Tightly-Secure Authenticated Key Exchange

Doctoral Thesis

Doreen Riepel

Fakultät für Informatik

Ruhr-Universität Bochum

Bochum, February 2023

Contents

Part I

Tightly-Secure Authenticated Key Exchange

Chapter 1

Introduction

Secure communication has become one of the most important goals of modern society. The use of digital communication media has seen an immense increase: instant messaging, online meetings and video calls have gained in importance, in personal life as well as in industry. Every day, we send and receive large amounts of data over the internet. These data streams are usually protected by encryption. More specifically, we use symmetric encryption algorithms where the same (secret) key is used for encryption and decryption. For this reason, symmetric encryption belongs to the broader field of *secret-key cryptography*. However, encrypting transmitted data is only one part to achieve secure communication.

Let us consider a setting with two parties, Alice and Bob. Before sending encrypted data, they need to choose and agree upon the key they want to use for encryption. Therefore, we also need to study mechanisms how this can be done in a secure way. In 1976, Diffie and Hellman proposed the first formal public-key key exchange scheme to solve this problem [DH76]. In such a scheme, both Alice and Bob hold their own secret value, their *secret key*, and publish some information related to that secret, a *public key*. Now Alice can use Bob's public key and her own secret key to compute a shared secret. Bob can do the same with Alice's public key and will be able to compute the same shared secret. This seminal paper initiated the study of asymmetric or *public-key cryptography*. Their scheme, today known as the Diffie-Hellman key exchange, is the basis for many cryptographic algorithms and protocols that have become an integral, though mostly invisible, part of our digital life.

In the above example, we are still missing an important piece. How does Alice know whether she is really using Bob's public key or possibly that of an adversarial third party? Therefore, *authentication* mechanisms have been proposed. Shortly after the work of Diffie and Hellman, the concept of digital signatures was formally defined by Rivest, Shamir and Adleman [RSA78]. They showed how to build signatures as well as public-key encryption based on the hardness of factoring large integers. Their work laid the foundation for the RSA cryptosystem which is named after the inventors and is yet another important building block of today's cryptography.

Authenticated Key Exchange. Authenticated key exchange (AKE) protocols combine the above ideas of exchanging a cryptographic key with additional authentication guarantees. Due to its high practical relevance, AKE can be considered one of the most

fundamental and important cryptographic primitives. In such a protocol, parties hold long-term public and secret keys which are used for authentication. When running the protocol, the involved parties interactively authenticate each other using their long-term keys, but also exchange ephemeral secrets to derive a shared secret which we call the session key. This shared session key can then be used to derive a symmetric key to encrypt further data, as described in the scenario above.

The most prominent real-world example is the Transport Layer Security (TLS) protocol [RD08, Res18]. Google reports that users of Google Chrome spend 99% of their browsing time on websites secured with TLS.[1] The TLS handshake that is responsible to set up the shared key between client and server is an AKE protocol. It is based on the Sign-and-MAC (SIGMA) paradigm [Kra03] which extends the plain Diffie-Hellman key exchange by signatures and message authentication codes. A whole framework of Diffie-Hellman key agreement protocols is provided by Noise [Per17]. The Noise token language allows to specify a variety of handshake patterns within a set of fundamental rules and thus it allows to capture a given scenario most accurately, e. g., whether parties are assumed to have preliminary knowledge of long-term keys. After the handshake, payload can be encrypted with the shared key using a symmetric cipher. Noise is also the basis for the Virtual Private Network (VPN) protocol WireGuard [Don17].

While the above applications use AKE in the most fundamental way, use cases can be found in various other settings. The Extended Triple Diffie-Hellman (X3DH) key agreement protocol [MP16] is widely used in messaging applications such as the Signal Messenger or WhatsApp. The main difference to the previous protocols is that it is designed to work in a scenario where the intended communication partner is offline. A server will therefore store not only messages but also ephemeral public keys which are used for the key exchange. The Messaging Layer Security (MLS) protocol [BBR+22] aims to provide secure messaging for groups of more than two parties. It allows multiple parties to jointly derive a shared session key. Moreover, it is designed to serve for continuous group key agreement, where group membership will change over time and new keys have to be derived.

Instead of using public keys for authentication, an alternative approach is to use pre-shared passwords. We call AKE protocols of this type *password-authenticated key exchange* (PAKE) protocols. This concept was first introduced by Bellovin and Merritt in 1992 [BM92]. While at first glance, it seems to capture real-world scenarios quite accurately, it is not nearly as widely used as traditional AKE. One reason for this may also be the lack of (secure) standardized protocols. Almost all protocols that were standardized by the Institute of Electrical and Electronics Engineers (IEEE) in 2009 [IEE09] turned out to have serious security issues. However, with recent standardization efforts by the Crypto Forum Research Group (CFRG), the adoption of PAKE protocols may see an increase in the future. The CFRG is a working group inside the Internet Research Task Force (IRTF). In 2020, they selected two protocols for standardization: CPace [HL19] and OPAQUE [JKX18], both of which have been thoroughly analyzed with formal security proofs.

[1] https://transparencyreport.google.com/https/overview

Provable Security and Tightness. Assessing the exact security of cryptographic primitives is a non-trivial task. In [GM82], Goldwasser and Micali formalize security for public-key encryption schemes and provide a methodological approach to analyze and prove the security of cryptographic schemes based on hardness assumptions. For this, we first need to define what security for a particular type of scheme, e. g., a public-key encryption scheme, means. We capture this in a *security game* between a challenger and an adversary $\mathcal{A}$. The challenger simulates the execution of a cryptographic scheme and $\mathcal{A}$ can interact with it via oracles. In the game, $\mathcal{A}$ has to solve some challenge which models the security of the scheme, e. g., distinguishing which of two messages was encrypted. Solving the challenge then translates to breaking the cryptosystem.

Once the security game is defined, we can prove security of a specific instantiation of the cryptographic scheme. This is done by describing a *security reduction*. In particular, we use the adversary $\mathcal{A}$ that breaks the cryptographic scheme to construct another adversary $\mathcal{B}$ that solves some computationally hard problem. The security follows from contradiction. If the problem that $\mathcal{B}$ solves is assumed to be hard, then such an adversary $\mathcal{A}$ cannot exist and the scheme must be secure.

We are also interested in quantifying the relation between $\mathcal{A}$ and $\mathcal{B}$. Therefore, if $\mathcal{A}$ runs in time $t_{\mathcal{A}}$ and has advantage $\epsilon_{\mathcal{A}}$ in breaking the scheme, then we can express the advantage of $\mathcal{B}$ in solving the underlying problem as $\epsilon_{\mathcal{B}} = \epsilon_{\mathcal{A}}/L$ for some L. We call L the *loss* of the security reduction. If $\mathcal{B}$'s running time $t_{\mathcal{B}}$ is about that of $\mathcal{A}$ and L is constant, then the reduction is considered *tight*. In particular, L should not depend on the number of oracle queries made by $\mathcal{A}$. Based on this analysis, we can now make statements about the concrete security which is usually stated in bits. In particular, if we have $\epsilon_{\mathcal{A}}/t_{\mathcal{A}} \leq L\epsilon_{\mathcal{B}}/t_{\mathcal{B}} \leq 2^{-\lambda}$, then the scheme offers λ bits of security.

The concrete security of a cryptographic scheme becomes important when choosing system parameters, e. g., the size of an RSA modulus or an elliptic curve group. The computation above illustrates the effect of the loss L on the concrete security: While a security proof usually attests security for polynomial-time adversaries, fixing system parameters may render such asymptotic security guarantees meaningless. In particular, a huge security loss implies that one should choose larger parameters in order to achieve the desired security level. Thus, a tight proof is always preferred over a non-tight proof and indeed, tightness has been considered for various cryptographic primitives, e. g., for digital signatures [BR96, Ber08, Sch11], public-key encryption [BBM00, HJ12, LJYP14] and variants like identity-based encryption [BR09, CW13, BKP14]. Depending on the cryptographic primitive, the tightness loss can manifest in several dimensions, e. g., the number of users in the system, the number of (challenge) ciphertexts the adversary obtains or the number of hash function evaluations. In some cases, it is even impossible to give a tight security proof. For example, Coron [Cor02] shows that security proofs for several signature schemes [BR96, GHR99, Pai99] are optimal in the sense that their security loss is unavoidable. These results were revisited and generalized to other classes of cryptographic primitives or specific schemes (e. g., [KK12, HJK12, LW14, BJLS16]).

Real-World Consequences of Tightness Gaps. When it comes to choosing system parameters for a scheme, practitioners often ignore the non-tightness of proofs since increasing the parameters comes with a significant efficiency penalty. Depending

on the application and cryptographic primitive in question, even a tight proof does not necessarily capture what we find in practice. This is due to the fact that many schemes are analyzed in a *single-user* and *single-challenge* setting. However, they are then used in systems with multiple users. Although the two settings are asymptotically equivalent in most cases (e. g., for public-key encryption schemes [BBM00]), exact security should be a concern, as the following examples demonstrate.

In the context of message authentication code (MAC) schemes, key collision attacks have shown to actually reduce the security by the number of keys [Bih02, CMS12], thus providing evidence that the tightness loss indeed plays a role for choosing system parameters. Furthermore, when combining a key encapsulation mechanism (KEM) with a data encapsulation mechanism (DEM) to obtain a hybrid public-key encryption scheme, it turns out that an attack against the multi-user security of the DEM tightly translates to an attack against the multi-user security of the hybrid scheme [Zav12, GKP18]. Since deterministic DEMs are subject to key collision attacks, similar to MACs, the combined scheme is only as secure as the weakest component and thus the actual security is also affected by the tightness gap.

Lattice-based cryptography has attracted a lot of interest over the past decades and also plays an important role in the post-quantum competition by the National Institute of Standards and Technology (NIST) that aims at standardizing public-key encryption and digital signatures. While a huge benefit of building public-key cryptography based on lattice problems is the existence of worst-case to average-case reductions for those problems (e. g., [Ajt96, Reg05]), the bounds are mainly asymptotic. For concrete parameters, this may incur a huge and sometimes exponential tightness loss, as pointed out in [CKMS16, KM19, KSSS22]. Considering that determining parameter sets for lattice-based schemes is a non-trivial task in itself, this makes it very hard to give good estimations on the exact security.

Modeling Security of AKE. Whereas for standard public-key encryption and signatures, the single-user setting is the most widely accepted security setting, all security models for AKE protocols consider an execution environment with multiple parties, mirroring their real-world application scenario. We want to formalize the security in terms of a game. Due to the interactivity, the security game needs to reflect strong adversarial capabilities. We want the adversary to be able to initiate multiple sessions between any pair of parties. It may read, send, modify or delete messages that are exchanged between the parties. With this, we model security against active adversaries, motivated by practical attacks, e. g., the Bleichenbacher attack [Ble98] against an old version of TLS where the padding of the used RSA encryption standard provides a decryption oracle to the adversary.

Another concern is that an adversary might be able to obtain secret information, e. g., if secret keys of parties get lost or stolen or if an adversary gets access to their devices. Thus, we allow the adversary to adaptively corrupt users or reveal session keys. So far, we have only described the capabilities of the adversary, but not what we consider a break of an AKE protocol. In game-based security notions such as [BR93a, BPR00, CK01, Kra05, LLM07], we define security based on key indistinguishability, i. e., the adversary's challenge is to distinguish a real non-revealed session key from

a uniformly random one. The session can be chosen adaptively, but trivial winning conditions have to be defined based on exposures. This challenge models an important cryptographic principle, namely that session keys of different sessions should indeed be independent of each other.

While this thesis focuses on security of AKE in the traditional and most fundamental sense, several other properties and security goals of AKE protocols have been analyzed. For example, privacy-preserving AKE [SSL20, RSW21, LLHG22] aims to hide the identities of involved parties. In some practical settings, it may also be desirable that only one party authenticates to the other. Thus, other security goals like one-way authentication and anonymity [GSU12, IY22] were considered, but also deniability [UG15] and leakage-resilience [ASB14, CMY+16].

AKE and Tightness. Due to the complexity of security models, security proofs of AKE protocols can be quite involved. Thus, a common proof technique in AKE is to let the reduction first guess the session for which the adversary will ask the challenge. Depending on the exact guessing strategy, this introduces a security loss that is at least linear in the total number of sessions. To estimate the security loss of this reduction, consider Google Chrome as an example. In 2021, 3.2 billion users had Google Chrome as their primary browser.[2] If each of these users establishes only one session secured over TLS, this makes already more than 2^{31} sessions. Thus, the security loss can be immense and suggests not to use standard system parameters. Instead, we should adapt the protocol's system parameters accordingly. Although we are not aware of generic attacks against AKE protocols as those mentioned above, we should aim for a tight proof in the first place to avoid unpleasant surprises. For this reason, there has been a lot of interest and effort in revisiting security proofs of protocols that are used in practice. In many cases, a more involved proof strategy can actually lead to a tight(er) security proof. Prominent examples are the security proof of TLS and SIGMA [DJ21, DG21, DDGJ22] as well as the signed Diffie-Hellman protocol [GJ18, PQR21]. We continue this line of research by analyzing the concrete security of well-established Diffie-Hellman based protocols. Whereas the HMQV protocol [Kra05] and several variants have received extensive security analyses, their *concrete* security has not been analyzed before. Apart from HMQV, we study the concrete security of the NAXOS protocol [LLM07] for which no tight proof is known. More details are provided in Section 1.3.1 and Appendix A.

Another direction of research is to consider tightness already when designing key exchange protocols. In particular, when constructing AKE from other cryptographic building blocks, it can be useful to extract the exact security definition for those building blocks which will lead to a tight security proof. The next step would be to find tightly-secure instantiations for each building block. This was done for example in [BHJ+15] and we follow a similar approach for different generic constructions of AKE (cf. Sections 1.3.2 and 1.3.3 and Appendices B and C).

When instantiating protocols or analyzing their security, one might encounter impossibility results. As described above for signatures, it turns out that specific types of protocols cannot be proven secure without a tightness loss. Cohn-Gordon et

[2] https://www.statista.com/statistics/543218/worldwide-internet-users-by-browser/

al. [CCG+19] show that as long as long-term public keys are uniquely determined by the corresponding secret key, we run into a *commitment problem* and cannot do better than losing a factor linear in the number of users.

Post-Quantum Secure AKE. Concrete instantiations of AKE protocols that come with a tight security proof mostly rely on the hardness of different variants of the Diffie-Hellman problem in prime-order groups. These problems allow for efficient re-randomization: we can take one problem instance and randomize it in such a way that we get another or multiple random problem instances. If an adversary solves one of them, we can then extract the solution for the initial problem instance. This is particularly useful when considering security in the multi-user multi-challenge setting. However, all of these Diffie-Hellman based protocols are not post-quantum secure since quantum computers can break the underlying hardness assumption in polynomial time using Shor's algorithm [Sho94]. Thus, the need has grown to design and standardize post-quantum secure cryptography that withstands attacks of quantum computers. In 2022, NIST announced the winners of the post-quantum competition. Three of the four selected schemes are based on computational hardness assumptions for lattices. While we know how to build AKE from public-key encryption and signatures generically, we do not know of any *tightly-secure* lattice-based construction of AKE.

A hopeful candidate for post-quantum security with similar properties as the original Diffie-Hellman key exchange lies in isogeny-based cryptography. While only shortly after the announcement of the winners of the NIST competition, one of the alternate candidates based on the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange [JD11] was completely broken [CD22, MM22, Rob22], the Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) protocol [CLM+18] is not affected by these attacks. CSIDH builds upon a group action which, compared to standard (e. g., elliptic-curve) groups, offers limited structure. On the one hand, this prevents polynomial-time quantum attacks as captured by Shor's algorithm. Instead, the best known quantum attack is subexponential [Kup05]. On the other hand, this limited structure prevents from simply transferring most of the existing protocols to the CSIDH setting.

One exception is the AKE protocol by [CCG+19] which has been analyzed for CSIDH in two independent works [dKGV20, KTAT20]. However, the analysis inherits the (optimal) tightness loss that is linear in the number of users due to the uniqueness of long-term keys. While the limited structure of the group action gives little hope to instantiate the generic constructions of tight AKE protocols, we show how to construct the first tightly-secure PAKE protocol from group actions (cf. Section 1.3.4 and Appendix D).

Overview of this Thesis. In the following section, we will provide necessary preliminaries and a more thorough look at tightness in general. We will then give an overview on the syntax, security and constructions of AKE protocols in Section 1.2. We will also cover the special case of password-authenticated key exchange and prior work on tightly-secure AKE. The main results of the thesis are summarized in Section 1.3 and full versions of the respective papers are given in Appendices A to D. In Chapter 2, we discuss these results and provide an outlook on future work and open questions.

1.1 Preliminaries

In this section, we introduce the necessary preliminaries to understand the contributions of this thesis. The AKE protocols considered or instantiations thereof are mostly based on prime-order groups and group actions. More details are given below in Section 1.1.1. In our generic constructions of AKE, we rely on key encapsulation mechanisms, hash proof systems and digital signatures, which are introduced in Section 1.1.2. Some proofs are carried out in idealized models which are discussed in Section 1.1.3. We further give an overview on tightness in Section 1.1.4.

1.1.1 Groups and Group Actions

Elliptic-Curve Cryptography and Prime-Order Groups. Most of today's public-key cryptography is based on elliptic curves over finite fields. For cryptographic applications, we are interested in a group of points on the elliptic curve which is abelian and cyclic. In the following, we will also assume that these groups are of prime order p, as is the case for curves P-256, P-384 and P-521 [Nat13] standardized by NIST. Other curves used in practice, such as Curve25519 and Curve448 [LHT16], are not of prime order, but they do have similar properties (cf. [ABH+21]). We will generally denote public parameters of a cryptographic scheme or protocol relying on prime-order groups by $(\mathcal{G}, p, g)$, where $\mathcal{G}$ is the group of prime order p with generator g. We will use multiplicative notation, where each element $h \in \mathcal{G}$ can be written as g^a for some $a \in \mathbb{Z}_p$. When sampling an element uniformly at random from $\mathbb{Z}_p$, we will write $a \xleftarrow{\$} \mathbb{Z}_p$. We can now define the *discrete logarithm* (DLOG) problem in the following way: given g^a, where $a \xleftarrow{\$} \mathbb{Z}_p$, an adversary must compute a. Similarly, we can define the Diffie-Hellman problem and its variants:

- The *computational Diffie-Hellman* (CDH) problem requires to compute g^{ab} given (g^a, g^b) for random $a, b \xleftarrow{\$} \mathbb{Z}_p$.
- The *decisional Diffie-Hellman* (DDH) problem requires to distinguish whether (g^a, g^b, g^c) is a Diffie-Hellman tuple (i. e., $g^c = g^{ab}$) or whether it is a random tuple (i. e., $c \xleftarrow{\$} \mathbb{Z}_p$).
- The *gap Diffie-Hellman* (GapCDH) problem requires to solve CDH when given a decision oracle DDH that on input $(x, y, z) \in \mathcal{G}^3$ will output 1 if $y^{\mathsf{DLOG}(x)} = z$ and 0 otherwise. We call the problem the *strong Diffie-Hellman* (StCDH) problem if the first input to the DDH oracle is fixed.

A useful property of these problems is their *random self-reducibility* which we will discuss in the context of tightness in Section 1.1.4.

Isogeny-based Cryptography and Group Actions. Since quantum algorithms can easily break traditional elliptic-curve cryptography [Sho94], alternative candidates have been proposed, e. g., isogeny-based cryptography. In general, an isogeny is a map between two elliptic curves. The two most prominent examples on how to use these maps to construct a Diffie-Hellman key exchange is given by the Supersingular Isogeny Diffie-Hellman (SIDH) protocol introduced by Jao and De Feo in 2011 [JD11] and the

Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) protocol introduced by Castryck et al. in 2018 [CLM+18]. The basic idea of the key exchange is that both parties choose a random path in a (specified) graph of elliptic curves, starting from a fixed known curve, and then use the resulting curve as public key while the path (represented by isogenies) remains secret. The shared secret can be obtained by taking the same path starting from the other party's public key. In general, it is not clear what it means to repeat a path: In SIDH, this is solved by providing auxiliary information to the other party, while in CSIDH, this is solved by restricting to a smaller set of elliptic curves. Being a promising candidate for post-quantum secure key exchange, recent works [CD22, MM22, Rob22] show that the SIDH protocol is completely broken by classical computers. This is due to the auxiliary information about points on the respective elliptic curves that is used to obtain the same shared key. This auxiliary information has been subject to various attacks before (e. g., [GPST16, Pet17]), but a non-trivial application of Kani's theorem [Kan97] then led to an efficient key recovery attack. The CSIDH protocol does not need the auxiliary information and is thus not affected by this attack. We can describe the underlying structure by a commutative group action.

A group action is a map $\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$, where $\mathcal{G}$ is a group and $\mathcal{X}$ is a set. Then, for any group elements $g, h \in \mathcal{G}$ and set element x, it holds that $g \star (h \star x) = (gh) \star x$. In the case of CSIDH, $\mathcal{G}$ is additionally abelian. The idea of constructing a key exchange protocol based on the group action on the set of *ordinary* elliptic curves was already observed by Couveignes [Cou06] and independently by Rostovtsev and Stolbunov [RS06]. However, CSIDH is the first practical instantiation that, in contrast to previous work, uses the group action on the set of *supersingular* elliptic curves (cf. [CLM+18]). Further, CSIDH is believed to be post-quantum secure. The best known quantum attack is subexponential and uses Kuperberg's algorithm [Kup05] to solve the underlying dihedral hidden subgroup problem. To accurately model CSIDH as a group action, we mainly follow the framework of (restricted) effective group actions as formalized by Alamati et al. in [ADMP20]. Thus, we will denote an effective group action by the tuple $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$, where $\tilde{x} \in \mathcal{X}$ additionally describes some fixed public set element. We can translate computational hardness assumptions like CDH and DDH from the prime-order group setting to group actions which we will label with the prefix GA, e. g., $\mathsf{GA\text{-}CDH}$.

1.1.2 Cryptographic Building Blocks

We will use key encapsulation mechanisms, hash proof systems and digital signature schemes to generically construct AKE protocols. In the following, we will introduce the syntax and standard security notions.

Key Encapsulation Mechanisms. A key encapsulation mechanism (KEM) is a cryptographic primitive which allows a sender, knowing the public key of the receiver, to encapsulate a key in a ciphertext. The receiver can then recover the key with their secret key. More formally, a KEM consists of three algorithms $\mathsf{Gen}_{\mathsf{KEM}}$, Encaps and Decaps, where $\mathsf{Gen}_{\mathsf{KEM}}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ of public key and secret key, Encaps takes as input the public key pk and outputs a tuple (c, K) of ciphertext c and encapsulated key K, and Decaps takes as input the secret key sk and c and outputs a key K.

The security game for indistinguishability under chosen-plaintext attacks (referred to as IND-CPA security) provides the adversary with a challenge (c^*, K^*), where K^* is either the real key output by Encaps or a key drawn uniformly at random from the key space. The adversary's goal is to distinguish which is the case. The security game for indistinguishability under chosen-ciphertext attacks (referred to as IND-CCA security) additionally provides a decapsulation oracle to the adversary such that it will receive decapsulated keys for ciphertexts of its choice (excluding the challenge ciphertext).

Hash Proof Systems. The notion of smooth projective hashing was initially defined by Cramer and Shoup [CS02]. Let $\mathcal{SK}$ and $\mathcal{PK}$ be sets. Further, let $\mathcal{C}$, $\mathcal{K}$ be sets, $\mathcal{V} \subset \mathcal{C}$ a language and $\Lambda_{\mathsf{sk}} : \mathcal{C} \to \mathcal{K}$ a hash function indexed by $\mathsf{sk} \in \mathcal{SK}$. We say that Λ_{sk} is projective if there exists a projection $\mu : \mathcal{SK} \to \mathcal{PK}$ such that $\mu(\mathsf{sk}) \in \mathcal{PK}$ defines the action of Λ_{sk} over $\mathcal{V}$. That is, for every $c \in \mathcal{V}$, $\mu(\mathsf{sk})$ and c uniquely determine $K = \Lambda_{\mathsf{sk}}(c)$. We do not have any guarantees for elements in $\mathcal{C} \setminus \mathcal{V}$. In particular, it may not be possible to compute $\Lambda_{\mathsf{sk}}(c)$ from $\mu(\mathsf{sk})$ and $c \in \mathcal{C} \setminus \mathcal{V}$. A hash proof system (HPS) can be seen as a special form of key encapsulation mechanism where $\mathcal{PK}$, $\mathcal{SK}$, $\mathcal{V}$, $\mathcal{K}$ are the sets of public keys, secret keys, ciphertexts and encapsulated keys, respectively. Key generation draws a random $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$ and derives $\mathsf{pk} \coloneqq \mu(\mathsf{sk})$. Encapsulation draws a ciphertext c together with a witness r of the fact that $c \in \mathcal{V}$. Using the witness, it is possible to compute $K \coloneqq \Lambda_{\mathsf{sk}}(c)$ from pk. Decapsulation can in turn compute K using the secret key.

We consider two main properties of hash proof systems: a subset membership problem and universality. The subset membership problem requires to distinguish whether an element c is drawn from $\mathcal{V}$ or from $\mathcal{C} \setminus \mathcal{V}$. The universality property makes statements about the entropy. In particular, we say a hash proof system is *perfectly universal*$_1$ if the value $\Lambda_{\mathsf{sk}}(c)$ for any ciphertext $c \notin \mathcal{V}$ is uniformly random in $\mathcal{K}$, even given $\mathsf{pk} = \mu(\mathsf{sk})$. This means that even a possibly unbounded adversary cannot distinguish the distributions $(\mathsf{pk}, \Lambda_{\mathsf{sk}}(c))$ and $(\mathsf{pk}, K \xleftarrow{\$} \mathcal{K})$, where the randomness is taken over the random choice of sk. Further, we can also define a strengthened form of universality. We say that a hash proof system is *perfectly universal*$_2$ if for all $c' \in \mathcal{C}$, the value $\Lambda_{\mathsf{sk}}(c)$ for any $c \notin \mathcal{V} \cup \{c'\}$ is uniformly random in $\mathcal{K}$, even given $\mathsf{pk} = \mu(\mathsf{sk})$ and $\Lambda_{\mathsf{sk}}(c')$.

Digital Signature Schemes. A digital signature scheme (SIG) is a cryptographic primitive which allows a sender to sign a message using their signing key. The resulting signature can be verified by anyone who is in possession of the verification key. More formally, a signature scheme consists of three algorithms $\mathsf{Gen}_{\mathsf{SIG}}$, Sign and Vrfy, where $\mathsf{Gen}_{\mathsf{SIG}}$ outputs a key pair $(\mathsf{ssk}, \mathsf{vk})$ of (secret) signing key and (public) verification key, Sign takes as input the signing key ssk and a message m and outputs a signature σ, and Vrfy takes as input the verification key vk, the message m and signature σ and outputs 1 if σ is a valid signature for m and 0 otherwise.

Security is captured by existential unforgeability against chosen-message attacks (referred to as EUF-CMA security). In this game, the adversary has access to the verification key as well as a signing oracle which outputs signatures on messages of the adversary's choice. The goal of the adversary is to forge a new pair of message and signature (m^*, σ^*) for which verification succeeds and which has not been output by

the signing oracle. In the slightly stronger notion of strong existential unforgeability (SUF-CMA), the adversary is also allowed to provide a signature forgery for a message that was already queried to the sign oracle, but the forged signature has to be different to the one output by the game. We will talk about this technicality and an extended security notion when talking about security of generic AKE constructions in Section 1.2.2.

1.1.3 Idealized Models

In the following, we will describe two abstract models of computation which are widely used tools in provable security: the random oracle model (ROM) and the generic group model (GGM).

The Random Oracle Model. The methodology of the ROM was first introduced by Bellare and Rogaway [BR93b]. It is used in security proofs to idealize hash functions as truly random functions. In particular, when proving security of a scheme in the ROM, the game provides oracle access to the hash function and simulates its output via lazy sampling. That is, each time the adversary (or the game itself) wants to evaluate the hash function, a uniformly random value from the output space is chosen. The game keeps a list of queries to remember outputs and to answer consistently whenever a value is queried multiple times. The ROM thus provides several properties that are useful in security proofs. The first one is that it allows the security reduction to *observe* the adversary's queries. The reduction always knows which queries have been made and as long as a query has not been made, the output is completely independent of the adversary's view. Many proofs rely on the fact that the solution to the computationally hard problem serves as input to the hash function. Thus, observing the queries made by the adversary can be used to *extract* the solution to that problem. A second useful property is *programmability*. As long as the output values are correctly distributed, the reduction can embed, e. g., its own challenge, or assign specific values at a later point in time.

The ROM has been used to prove security of many practical schemes, such as OAEP [BR95a] or DHIES [ABR01]. It also allows for useful transformations, e. g., the Fujisaki-Okamoto (FO) transformation [FO99] in the context of public-key encryption or the Fiat-Shamir transformation [PS96] in the context of signature schemes. The ROM can also be used as a tool to enable tight security proofs which we will discuss in more detail in Section 1.1.4.

Being a very powerful heuristic, the validity of the ROM has been questioned from the beginning. Though contrived, there exist several examples of schemes that can be proven secure in the ROM, but are actually insecure for any practical instantiation of the hash function [CGH98, GK03, BBP04, BFM15]. Despite these uninstantiability results, no attack against a practical scheme has been found which makes the ROM a well-established model to prove security for practical applications. Several variants of the ROM have been proposed to model hash functions and an adversary's capabilities more accurately, e. g., the global random oracle model [CJS14, CDG+18], the augmented random oracle model [Zha22] or the quantum random oracle model [BDF+11]. The latter is probably the most prominent variant since it has become the standard tool to argue about post-quantum security of schemes that use hash functions.

The Generic Group Model. The generic group model (GGM) was formally introduced by Shoup [Sho97] and later refined by Maurer [Mau05]. The GGM makes the assumption that most attacks against cryptographic groups are generic, i. e., they only exploit the group structure and not the actual representation of group elements. It is a useful model to give information-theoretic lower bounds on the computation complexity of (generic) attacks. In the GGM, an adversary gets oracle access to all group operations. In particular, each group element is represented by a handle which the adversary can use to perform computations. The running time of an adversary in the GGM is then defined by the number of group operations, i. e., the number of queries to the group operation oracle. If $\mathcal{A}$ is a generic adversary making at most q queries to the group operation oracle for a group of prime order p, then the lower bound of solving DLOG is $O(q^2/p)$. This bound matches the best known attack against DLOG in prime-order elliptic curve groups used in practice.

Although similar uninstantiability results as for the ROM exist (e. g., [Den02]), the GGM is a useful tool to justify the hardness of non-standard assumptions or security of a scheme. For example, it has been used to analyze the one-more discrete logarithm assumption [BFP21], the simultaneous Diffie-Hellman assumption [PW17] and attribute-based encryption schemes (e. g., [BSW07]). An analysis in the GGM is also interesting in the context of tightness. A tight bound in the GGM (i. e., one that matches the bound stated above) supports the use of standard group sizes for schemes when a tight bound is either unknown or impossible (e. g., [BD20]).

1.1.4 Tightness

Tightness has been considered for various cryptographic primitives, such as digital signatures and public-key encryption. Depending on the primitive, tight security may be relevant from different angles. When proving the security of a signature scheme, for example, it is natural to allow the adversary a polynomial number of queries to the signing oracle. A security bound would then be considered tight if it is independent of the number of signing queries. For IND-CCA security of public-key encryption schemes, the bound should be independent of the number of decryption queries. These two examples apply to the standard single-user setting. However, extending them to the multi-user setting requires the bound to be additionally independent of the number of users. Similarly, for encryption schemes, we can consider tightness for many challenges (i. e., encryption) queries.

This section aims to provide some insights into aspects of tight security in general, whereas tightness in the context of AKE is considered in Section 1.2.4.

Random Self-Reducibility. We call a problem random self-reducible if we can transform any instance of the problem into a random instance of the problem. This is not only useful for tight security proofs, but it also means that for fixed public parameters either (almost) all instances of the problem are hard or no instance.

Computationally hard problems over prime-order groups such as DLOG, CDH and DDH allow for a random self-reduction. By a re-randomization argument, we can thus show a tight reduction from a single-instance version of the problem to a multi-instance

version. This observation can be used in a security proof to actually embed multiple problem instances at the same time. Similarly, we can randomize instances of group action assumptions such as GA-DLOG and GA-CDH. Unfortunately, the limited structure of the group action prevents a random self-reduction for GA-DDH (cf. [KTAT20]).

Prior Work on Tight Security. In [BBM00], Bellare, Boldyreva and Micali give reductions for the multi-user and multi-challenge security of public-key encryption schemes showing the asymptotic equivalence to the standard (single-user single-challenge) security notion. However, the non-trivial direction of their proof uses a hybrid argument resulting in a loss that depends on the number of users and challenges. They also show that we can get a tight bound for specific schemes like the ElGamal encryption scheme [ElG84] and the Cramer-Shoup encryption scheme [CS98]. Both reductions use the random self-reducibility property of the DDH problem as described above. Tightly-secure public key encryption has been considered in several other works, e. g., [HJ12, LJYP14, GHKW16, GHK17, HLLG19], as well as in the identity-based setting [BR09, CW13, BKP14, LP20a, LP20b]. Some primitives seem particularly hard to construct, like tightly-secure non-interactive key exchange (NIKE) with adaptive security [FHKP13, HHK18, HHKL21]. One reason for this is that strong security models for NIKE (as those for AKE) consider adaptive corruptions of parties and are often subject to impossibility results. While tightly-secure signatures were considered in several works (e. g., [BR96, Ber08, Sch11]), we need an even stronger notion of security for signature schemes if we want to use them to construct tightly-secure AKE. We discuss the arising difficulties and constructions in more detail in Section 1.2.4.

In this thesis, we look at tightness by considering the success/time ratio as described earlier in the introduction. However, there also exist variants of the tightness definition that take into account the amount of memory. The study of *memory-tight* reductions was initiated by Auerbach et al. [ACFK17] and was used to analyze various primitives and schemes [Bha20, GJT20, GT20, DGJL21a] as well as to give lower bounds on the memory consumption of reductions [ACFK17, WMHT18].

Tightness in the ROM. It is important to note that impossibility results often do not apply in general but only to specific types of reductions or schemes. Whereas [BJLS16] show that unique signature schemes have an inherent tightness loss that is linear in the number of signing queries, Guo et al. [GCS+17] construct a scheme in the ROM that circumvents this tightness loss. In particular, they solve the underlying problem by extracting the solution from the hash queries. Their result does not contradict the impossibility result of [BJLS16], but rather illustrates its limitations.

In non-committing encryption, we assume the existence of a simulator that is able to generate fake ciphertexts which can later be opened to any message. As pointed out by Nielsen [Nie01, Nie02], such a non-committing encryption scheme can be efficiently constructed from any trapdoor permutation in the ROM. However, he also shows that we cannot hope to transfer those schemes to the standard model. The ROM allows to first create a random ciphertext and later, on corruption, the reduction can produce a valid explanation of the ciphertext by reprogramming the random oracle accordingly. For this reason, the random oracle provides a way to resolve the so-called commitment

problem that often arises in the context of tight security. We will further elaborate on this in Section 1.2.4.

Exploiting Tightness Gaps. The fact that a tight proof does not exist (or is even impossible) does not mean that there is indeed an actual practical attack. However, there are examples where attacks have shown to indeed exploit the tightness gap.

Chatterjee, Menezes and Sarkar [CMS12] show that the tightness gap of a MAC scheme in the multi-user setting, where keys and tags are of equal length, can lead to an attack against the scheme which is much better than exhaustive search. In this attack, the MAC scheme is modeled as an ideal MAC, i. e., it produces random outputs, and thus works generically for any MAC scheme. The attacker first asks for a tag of the same message for all users, then it tries to find a collision by computing the tag for this message with a subset of keys. A collision can either be caused due to a key collision or due to a tag collision. Since we assume both to be of equal length, it is a key collision with probability $1/2$. Thus, considering n keys with bit length r, a key collision happens with probability $2^r/n$. With this, we are not only able to produce a forgery, but it is also a complete key recovery attack.

Since operations of public-key encryption schemes are rather expensive compared to symmetric encryption schemes, those two primitives are in practice combined as hybrid encryption. The resulting public-key encryption scheme first encapsulates a key using a KEM and then uses this key to encrypt the message using a data encapsulation mechanism (DEM). Similar to the above key collision attacks, [Zav12, GKP18] show that deterministic DEMs also allow for (passive) attacks that allow to distinguish encryptions or even recover secret keys, making use of the number of keys. They also show that in order to get a tightly-secure hybrid public-key encryption scheme, all components must be tightly secure. In particular, any attack against the DEM (or an intermediate key derivation function) tightly implies an attack against the hybrid scheme.

1.2 Authenticated Key Exchange

This section aims at providing the background to understand the formal goals and guarantees of AKE. For this, we will first introduce some terminology and how security is modeled. We then describe how to generically construct AKE protocols from public-key primitives. Finally, we will discuss tightness in the context of AKE and prior work on tightly-secure AKE.

Terminology. An AKE protocol is executed between two parties where each party is in possession of a *long-term* key pair. One particular instance of an execution is called a *session*. The intended communication partner is called the *peer* of a session. Further, we will refer to the party which sends the first message of a session as the *initiator* and to the one that receives this message and continues the interaction as the *responder*. A protocol execution can consist of multiple *rounds*, where message that can be sent independently are counted as a single round. At the end of the session, both parties will output a shared *session key*. Between rounds, both parties may have to store some

intermediate session-specific values which can be public or secret. We will denote these values as the *state* of a session.

Correctness of an AKE protocol means that two honest parties compute the same session key when each party has received all messages sent by the peer of the session.

Instead of computing the session key, a session may also terminate early and output a special failure symbol $\perp$, e. g., when one of the parties detects an active attack. Based on this, we can broadly define two classes of AKE protocols, those with *implicit* authentication and those with *explicit* authentication. In explicitly authenticated protocols, the key exchange will fail and the session will terminate with $\perp$ if the involved parties do not authenticate each other successfully. In contrast to that, an implicitly authenticated protocol will only ensure that both parties agree on the same session key if they execute the protocol honestly and no active attack happens. In case of an active attack, we want to ensure that session keys are different (or rather independent) which can then be detected once the key is used in further communication.

1.2.1 Security

In this thesis, we focus on game-based security models for AKE which are based on key indistinguishability. Since the introduction of the first formal model by Bellare and Rogaway [BR93a], several security models have been proposed to capture different properties or to address shortcomings of previous models (e. g., [BR95b, BPR00, CK01, Kra05, LLM07]). We now recall the most important definitions, extensions and variants used in the literature. We provide a more general discussion on the variety of different models and the implications for tight security in Section 2.1.

Execution Environment. The AKE security experiment is described as a game between a challenger and an adversary $\mathcal{A}$. There are N parties $\mathsf{P}_1, \ldots, \mathsf{P}_N$, where each party P_i holds a pair of long-term keys $(\mathsf{lpk}_i, \mathsf{lsk}_i)$ and can interact in S sessions denoted by $\pi_i^1, \ldots, \pi_i^S$. At the beginning, $\mathcal{A}$ receives all long-term public keys $(\mathsf{lpk}_1, \ldots, \mathsf{lpk}_N)$. Then we give $\mathcal{A}$ full control over the network by providing the following oracles:

- Oracle Send is used to either initiate a new session or send messages to a previously initiated session. For this, the oracle takes as input a tuple (i, s) indicating the session π_i^s and a protocol message. During the execution, the session π_i^s stores session-specific variables, e. g., the intended communication partner as well as sent and received messages. The Send oracle allows $\mathcal{A}$ to send messages on behalf of other parties. In particular, $\mathcal{A}$ can decide to forward or replay messages, modify them in transit or also drop messages. Once the session key has been computed or the execution has stopped with the failure symbol, no more messages can be sent to the session π_i^s.
- Oracle Reveal takes as input a tuple (i, s) indicating the session π_i^s and outputs the session key K of that session if it exists.
- Oracle Corrupt takes as input an index i indicating party P_i and outputs the party's long-term secret key lsk_i.

– Oracle TEST takes as input a tuple (i, s) indicating the session π_i^s and outputs a challenge key which is determined by a random challenge bit b. If $b = 0$, $\mathcal{A}$ receives the session key of π_i^s. Otherwise, it receives a uniformly random key from the session key space. In the following, we denote the session subject to this query the *test session*.

All oracles can be queried adaptively. The adversary's goal is to guess the challenge bit. The fact that the adversary is allowed to reveal session keys of sessions other than the test session implies that session keys should in general be independent of each other. However, we need to specify in which cases a sequence of queries allows to trivially determine the challenge bit, e. g., the session key of the test session must not be revealed. Due to the complexity of the security game, restricting the adversary in order to prevent trivial wins is crucial and not always straightforward. Generally, we call a session *fresh* if it qualifies for a test session, i. e., it does not allow trivial wins. Before we are able to define freshness criteria, we need to introduce the concept of *partnering*.

Partnering. Whenever the adversary only forwards messages between two sessions π_i^s and π_j^t belonging to parties P_i and P_j, they will compute the same session key (by correctness). If one of the two sessions is the test session, we need to restrict the adversary from revealing the session key of the other session. For this, we need to formally define when a session is *partnered* to another session. The commonly used approaches to defining partnering are either based on matching conversations [BR93a] or on original keys [LS17].

The concept of matching conversations was already introduced in the first formal model for AKE [BR93a]. Partnering based on matching conversations is pretty simple: one session is partnered to another if they received those messages that were sent by the other one. However, the party sending the last message will never know if the intended partner also receives this message. Thus, the last message needs to be ignored when determining whether two sessions are partnered.

However, Li and Schäge [LS17] noticed that this sometimes puts unnecessary strong assumptions on protocols. They identify a class of "no-match" attacks. For example, if an adversary manages to randomize a signature, then sessions are not partnered in the sense of matching conversations. Thus, signatures need to be either deterministic or satisfy the stronger notion of SUF-CMA (cf. Section 1.1.2). To address this, they introduce the notion of original-key partnering. More specifically, the original key is the session key computed by the involved parties under a passive adversary and we say that two sessions are partnered if they both compute the original key.

Forward Security. The notion of perfect forward secrecy was originally used by Günther [Gün90] in the context of identity-based key exchange. It refers to the property that already established session keys are still secure, i. e., indistinguishable from a random session key, even if the long-term secret keys of the involved parties are leaked afterwards. Later, the relaxation of weak perfect forward secrecy was introduced in [Kra05]. In the following, we will denote the two properties by *forward security* and *weak forward security*, respectively. Whereas forward security guarantees key indistinguishability for all previous sessions, including those where an adversary may have been actively

interfering, weak forward security only provides guarantees for those sessions where the adversary was passive.

In the AKE security model, we capture (weak) forward security by allowing the adversary to corrupt long-term secret keys via oracle Corrupt and by defining freshness accordingly. For forward security, a session is fresh if at the time the session key was computed, none of the involved parties was corrupted. For weak forward security, a session is fresh if there additionally exists a partnered session. In general, implicitly authenticated AKE protocols can only achieve weak forward security [Kra05, BG11, Sch15].

Key-Compromise Impersonation Attacks. An additional property we want to model is security against key-compromise impersonation (KCI) attacks. Even if a party is corrupted, it should not be possible for an adversary to impersonate another (uncorrupted) party towards the corrupted party. It may be possible that a user does not know that their secret key was compromised, so resilience against KCI attacks can maintain at least some security guarantees for the corrupted party.

Protection against KCI attacks is captured by all common security models by allowing the adversary to first corrupt a party and then issue a test query against a session of this party.

Protecting against State Reveals. In some security definitions (e. g., [Kra05, LLM07]), the adversary is also allowed to reveal the secret state of a session. This is usually done via an additional oracle:

– Oracle RevealState outputs the secret state of the specified session.

The definition of the exact output of this oracle highly depends on the security model. There exist different flavors of definitions: the state could be the randomness that was used by the session to compute the message or it could also be all secret information related to a session. This makes the definition of trivial wins even more complex. Further, there exist generic constructions which are not secure when revealing ephemeral secrets, but applying simple techniques then allows to prove security. The most prominent example is probably the "NAXOS hashing trick" [LLM07]. The NAXOS protocol is a Diffie-Hellman based AKE, where the ephemeral public key (or rather the secret exponent) is derived by hashing an ephemeral secret string together with the long-term public key. While the proof of NAXOS relies on the ROM, Okamoto [Oka07] and Fujioka et al. [FSXY12] use a similar approach that works in the standard model based on pseudorandom functions. In the following, we will define the state as everything that needs to be stored to compute the response to the next message of the protocol. Further, we assume that the state of a session can be revealed independently of long-term keys since long-term keys are long-lived and usually better protected in memory.

Extension to Multiple Challenges. While early security models considered a single challenge query, more recent models extended this to multiple challenges. This is a natural extension, in particular, when considering tightness. Interestingly, two different flavors of multi-challenge security models have been proposed.

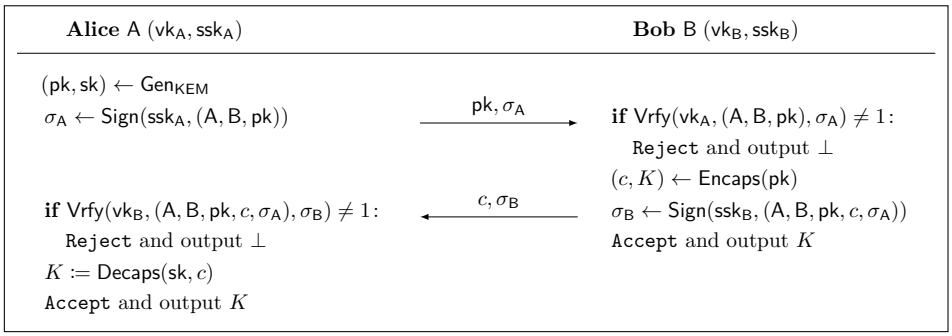

Figure 1: Explicitly authenticated AKE protocol using a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen_{KEM}}, \mathsf{Encaps}, \mathsf{Decaps})$ and a signature scheme $\mathsf{SIG} = (\mathsf{Gen_{SIG}}, \mathsf{Sign}, \mathsf{Vrfy})$. Long-term keys are signature key pairs, generated by Alice as $(\mathsf{vk_A}, \mathsf{ssk_A}) \leftarrow \mathsf{Gen_{SIG}}$ and analogously for Bob.

- Multi-Bit Guess (MBG): For each query to Test, the challenger draws an independent challenge bit b_i. The adversary's goal is to guess one of them.
- Single-Bit Guess (SBG): There is one global challenge bit b based on which all queries to Test are answered. The adversary's goal is to guess b.

Both approaches have been frequently used. The first approach was used in [BHJ+15, GJ18, LLGW20], and the second approach in [CCG+19, DJ21]. While the MBG approach is not known to allow a tight composition with a symmetric primitive, the SBG approach is tightly equivalent to a real-or-random style security notion, where the adversary has to distinguish a game where all challenge keys are real from one where all challenge keys are random. Thus, it exactly captures what is needed for a tight composition with symmetric primitives, as noted by [CCG+19].

1.2.2 Generic Constructions

We recall two paradigms to generically construct AKE protocols from the basic cryptographic building blocks that were introduced in Section 1.1.2. Parts of the results in this thesis build upon the following constructions by establishing security definitions for the building blocks that allow for a tight security proof.

Signatures for Explicit Authentication. A very simple and straightforward way to construct an AKE protocol with explicit authentication is to use a KEM in combination with a signature scheme. The generic construction is shown in Figure 1, illustrating a key exchange between Alice and Bob. Both have signature key pairs as long-term keys, i. e., $(\mathsf{vk_A}, \mathsf{ssk_A})$ and $(\mathsf{vk_B}, \mathsf{ssk_B})$, respectively, which are used to explicitly authenticate each other. The KEM is used to derive the shared session key.

The protocol is executed as follows. Alice (the initiator of the session) draws an ephemeral KEM key pair $(\mathsf{pk}, \mathsf{sk})$ and computes a signature σ_A on the public key along with her own and Bob's identity. She sends the tuple $(\mathsf{pk}, \sigma_\mathsf{A})$ to Bob. Bob verifies the signature and if verification succeeds, he computes a ciphertext and an encapsulated key

(c, K) using the ephemeral public key. He computes a signature σ_{B} on the ciphertext c as well as both their identities and Alice's previous message. The session key will be the KEM key K. If Alice verifies σ_{B} successfully, she decapsulates the key and outputs the result. If verification of the signature fails, then Alice (or Bob) will terminate the session and output the failure symbol $\perp$.

This protocol basically follows from generalizing the signed Diffie-Hellman protocol [Sho99] which is the basis for many practical AKE protocols [Kra03, KHN+14, Res18]. In this case, the original Diffie-Hellman key exchange protocol [DH76] is acting as a NIKE scheme which is known to imply KEMs [FHKP13]. One advantage of using a NIKE is that it allows for a one-round protocol where the parties' messages do not depend on each other. However, constructing (efficient) NIKE schemes from other assumptions than Diffie-Hellman is a challenging problem (e. g., [GKRS22]). KEM schemes, on the other hand, are easier to construct, thus we focus on the KEM-based protocol (as described above).

It might be desirable to tie the session key to the context of the session, e. g., the identities of the involved parties and the session transcript. While this is not necessary in general (i. e., it depends on the security requirements of the underlying building blocks), it is common practice to use an additional key derivation function. We will further elaborate on this below.

KEMs for Implicit Authentication. An alternative approach to using signatures is to construct AKE from KEMs directly. That is, the involved parties will (implicitly) authenticate each other with a KEM. The resulting protocol is shown in Figure 2. Here, Alice and Bob both hold long-term KEM key pairs, denoted by $(\mathsf{pk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{A}})$ and $(\mathsf{pk}_{\mathsf{B}}, \mathsf{sk}_{\mathsf{B}})$, respectively, which will be combined with an ephemeral KEM.

The protocol proceeds as follows. First, Alice chooses an ephemeral KEM key pair. For disambiguation, we will denote this key pair by $(\tilde{\mathsf{pk}}, \tilde{\mathsf{sk}})$. She then uses Bob's public key pk_{B} to compute a ciphertext and key $(c_{\mathsf{A}}, K_{\mathsf{A}})$ and sends $(\tilde{\mathsf{pk}}, c_{\mathsf{A}})$ to Bob. Bob computes two pairs of ciphertext and key $(\tilde{c}, \tilde{K})$ and $(c_{\mathsf{B}}, K_{\mathsf{B}})$ using $\tilde{\mathsf{pk}}$ and Alice's long-term key pk_{A}. He derives key K_{A} by decapsulating c_{A} using his secret key sk_{B}. Bob has now computed three keys, K_{A}, K_{B} and $\tilde{K}$, which allow him to derive the final session key K. The key derivation may also include the context of the session. Apart from this, we want to leave this procedure abstract for now. Finally, Bob sends $(\tilde{c}, c_{\mathsf{B}})$ to Alice and she can compute the two remaining keys $\tilde{K}$ and K_{B} using the ephemeral secret key $\tilde{\mathsf{sk}}$ and her long-term secret key sk_{A}. She derives the session key in the same way as Bob.

The above protocol allows for implicit authentication since active attacks can only be detected once the session key is computed. This generic construction was first proposed and analyzed in the identity-based setting [BCGP08]. Fujioka et al. [FSXY12] analyze the protocol above in the standard model using a pseudorandom function to derive the session key. Another approach would be to use a hash function which is then modeled as a (quantum) random oracle as considered in [HKSU20]. Constructing authenticated key exchange from KEMs is also interesting in the transition to post-quantum cryptography. Since known lattice-based signatures are rather expensive in terms of signature sizes,

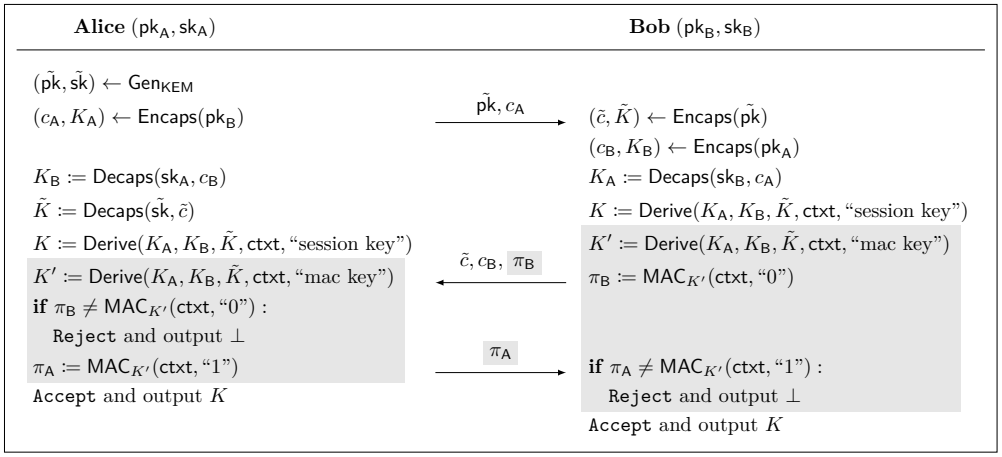

Figure 2: Implicitly authenticated AKE protocol using a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen_{KEM}}, \mathsf{Encaps}, \mathsf{Decaps})$. Long-term keys are KEM key pairs, generated by Alice as $(\mathsf{pk_A}, \mathsf{sk_A}) \leftarrow \mathsf{Gen_{KEM}}$ and analogously for Bob. An additional deterministic function Derive is used to compute the final session key from encapsulated keys $K_\mathsf{A}, K_\mathsf{B}, \tilde{K}$ and context $\mathsf{ctxt} = (\mathsf{A}, \mathsf{B}, \tilde{\mathsf{pk}}, c_\mathsf{A}, \tilde{c}, c_\mathsf{B})$. Explicit authentication can be achieved by adding a MAC as key confirmation step (gray boxes).

replacing the signatures with KEMs has been proposed for protocols like TLS or WireGuard [SSW20, HNS+21].

Explicit Authentication via Key Confirmation. The KEM approach above lacks explicit authentication and thus can only achieve weak forward security. To address this, we can add a *key confirmation* message to turn any implicitly authenticated AKE protocol into one with explicit authentication. This can be done, e. g., by using a message authentication code (MAC) as described by [Kra05] for the HMQV protocol.

Adding key confirmation to the above protocol extends it by an additional round, as illustrated in Figure 2. Instead of deriving only the shared session key, Bob will derive a second key K' using an alternative key derivation function. We will provide more details on this below. Then Bob uses K' to compute a (deterministic) tag π_B on the context and some fixed string. He sends π_B together with the two ciphertexts $\tilde{c}$ and c_B. Alice first derives the session key K and MAC key K'. Then she uses K' to verify π_B and if verification succeeds, she also computes a tag π_A using some different fixed string and sends it to Bob. Bob verifies the tag and only then, he accepts and outputs the session key. If any verification fails, Alice or Bob will terminate and output the failure symbol $\perp$.

The alternative key derivation function could be defined such that it only differs from the session key derivation function in the sense that it includes an additional fixed string (cf. Figure 2). This allows for domain separation, e. g., both can then be instantiated using the same hash function. For simplicity, we illustrate deterministic MACs here, e. g., instantiated using a hash function.

1.2.3 Password-Authenticated Key Exchange

The concept of password-authenticated key exchange (PAKE) was first introduced by Bellovin and Merritt [BM92]. In this setting, parties no longer have long-term keys, but we assume that two parties share the same password. Based on the knowledge of this (possibly low-entropy) password, they can compute a shared session key. We usually associate a role to the two parties interacting in a protocol, namely one party is a *user* and the other one is a *server*. In general, there exist two different settings or types of protocols. In the *balanced* setting, both user and server share the same password, whereas in the *unbalanced* setting, the server stores some augmented version of the password, e. g., a hash, which is why this type of protocol is also referred to as augmented PAKE. In this work, we will focus on balanced PAKE protocols.

Security. The game-based security model is similar to the standard AKE setting. The first formal model was given by Bellare, Pointcheval and Rogaway [BPR00]. Since parties have passwords instead of long-term keys, a corruption will reveal the password that is shared between two users in which case both parties are considered corrupted.

Passwords are considered to be chosen from a small set, thus the best attack that should be possible is an *online dictionary attack*. The adversary can always guess a password and initiate a session, but if the guess was wrong, it should not be able to rule out a larger number of passwords. In particular, we want to protect against *offline dictionary attacks* where the adversary can brute-force the password via an exhaustive search locally without further interaction. Forward security is defined as for the AKE setting. Thus, after the corruption of a password, all previously established session keys should still be secure. As for standard AKE, we consider multiple challenges as formalized by Abdalla, Fouque and Pointcheval [AFP05] to allow for tight composition.

Constructions of Balanced PAKE Protocols. There exist several PAKE protocols based on the Diffie-Hellman key exchange. In the Encrypted Key Exchange (EKE) paradigm introduced by [BM92], the password is used to encrypt the messages of the key exchange (i. e., the Diffie-Hellman public keys). A similar approach is to hash the password to a generator of the group and use this generator to perform the key exchange, as done in SPEKE [Jab96]. The SPAKE2 protocol [AP05] performs the Diffie-Hellman key exchange by involving additional group elements that are generated in a trusted setup phase in advance. These public parameters are also referred to as the common reference string (CRS).

The KOY protocol by Katz, Ostrovsky and Yung [KOY01] borrows techniques from the Cramer-Shoup cryptosystem to construct the first practical PAKE without random oracles, but it also relies on a CRS. This idea was generalized by Gennaro and Lindell [GL03, Gen08] who construct PAKE protocols from hash proof systems. Their construction builds upon a public-key encryption scheme and the language of ciphertexts for that scheme, i. e., a tuple of message and ciphertext (m, c) is in the language if there exists randomness r such that the encryption of m under r yields c, where we parameterize the language over the public key. These ideas were further extended to construct post-quantum secure PAKE based on lattices (e. g., [KV09, GK10]).

Canetti et al. [CDVW12] use oblivious transfer (OT) to generically construct PAKE protocols. In their two constructions, the password (interpreted as a bit string) is used in multiple executions of the OT protocol. In particular, for each password bit, the OT protocol is run twice: both parties once take the role of the sender and once that of the receiver. The efficiency of this protocol relies completely on that of the OT protocol and due to the "bit-by-bit" approach, it is rather unsuitable for practical applications.

1.2.4 Prior Work on Tightly-Secure AKE

The reason why constructing tightly-secure AKE is a challenging task is the underlying commitment problem. We will first illustrate the various manifestations of the commitment problem in the context of AKE. Then, we provide a summary of prior work on tightly-secure AKE and explain how they resolved or circumvented the challenges.

The Commitment Problem. Compared to standard public-key encryption schemes, security models for AKE are very strong in the sense that they allow the adversary to adaptively obtain secret values (cf. Section 1.2.1). The commitment problem arises when reducing to the hardness of a computationally hard problem or the security of a building block. In particular, the reduction has to decide in which session(s) it will embed the challenge instance. If a session is subject to a Test query, the reduction can only learn something from the adversary's behavior if it has indeed embedded a challenge instance in this session. However, if the adversary decides to later reveal the session key, then the reduction must output a valid session key. Depending on their behavior and queries, the adversary might be able to compute the key itself and notice whenever the simulation of the reduction fails. Since all queries can happen adaptively, the reduction cannot know for certain which of the cases will happen, so it must *commit* to either knowing the secret or not. The naive way to resolve this problem is to guess the test session and then embed the challenge instance only in that session, with the consequence of a non-tight proof.

We observe a similar manifestation of the commitment problem in the corruption of long-term secret keys. Since Corrupt queries also happen adaptively, the reduction must commit to which secret keys it actually knows or which it uses to embed a challenge. If the long-term public key uniquely determines the corresponding long-term secret key, we even run into an impossibility result. Based on the work from [BJLS16], Cohn-Gordon et al. [CCG+19] extend the impossibility result and show that for such AKE protocols, a security loss that is linear in the number of users is inherent. This applies in particular to a large class of Diffie-Hellman based protocols like HMQV [Kra05], NAXOS [LLM07] and variants such as [LM06, Ust08, YZ13]. In the context of KEMs, Han, Liu and Gu [HLG21] give further impossibility results in the multi-user setting with corruptions. They show that a security loss that is linear in the number of users is optimal for an even broader class of KEM schemes than those covered by previous impossibility results. This makes these KEMs insufficient to be used for implicit authentication in tightly-secure AKE. When using a signature scheme for authentication, the problem becomes even more paradoxical. The reduction must not only know signing keys for all parties (and be able to compute signatures), but it must also be able to solve its own challenge based on a valid signature forgery of the adversary. Due to this, we require

a very strong security definition for the signature scheme, which is multi-user strong unforgeability with corruptions ($\mathsf{MU\text{-}SUF\text{-}CMA}^{\mathsf{corr}}$). Although tightly-secure signature schemes in this setting are non-trivial, there exist several (more or less) efficient schemes in the random oracle model [Bad14, GJ18, DGJL21b] and even in the standard model [BHJ+15, HJK+21].

We can go further and describe yet another variant of the commitment problem when considering state reveals. In this case, the reduction must also know valid states for all sessions, while at the same time tackling the other two commitment problems. We now want to elaborate on how previous work resolved these problems.

AKE in the Random Oracle Model. In order to give a tight proof for the signed Diffie-Hellman key exchange protocol, Gjøsteen and Jager [GJ18] slightly tweak the protocol. In particular, the initiator must first send a hash of the ephemeral Diffie-Hellman public key that will be used for the key exchange. In the security proof, this hash function will be a programmable random oracle. Intended to serve as a commitment in the protocol, the ROM allows the reduction to actually *not* commit to the public key when the session is initiated, but only later when it knows whether the session qualifies for a test session. Due to this change, we get a 3-round protocol, whereas the original protocol can be executed in one round. They prove security in an MBG-style security model under the DDH assumption.

In the protocol analyzed by Cohn-Gordon et al. [CCG+19], the two parties compute three combinations of Diffie-Hellman keys based on long-term and ephemeral keys, similar to the X3DH protocol. Their proof relies on the StCDH assumption and the ROM which allows them to identify critical queries by the adversary and to "patch" the random oracle accordingly. A similar proof strategy is employed by Diemert and Jager [DJ21] as well as Davis and Günther [DG21] for TLS and SIGMA. Pan, Qian and Ringerud [PQR21] revisit the problem of proving tight security of the original signed Diffie-Hellman protocol. They show that the modification of Gjøsteen and Jager is not necessary to give a tight security proof when using the StCDH assumption. All these works rely on an SBG-style security model.

AKE in the Standard Model. The first tightly-secure AKE protocol was proposed by Bader et al. [BHJ+15]. They focus on security in the standard model and generically construct a 3-round AKE protocol from a (standard) signature scheme, a one-time signature scheme and a KEM. The signature scheme needs to achieve existential unforgeability in the multi-user setting with corruptions, whereas the KEM needs to satisfy CPA security in the multi-user setting with corruptions (in MBG style). The latter can, for example, be generically obtained by any CPA-secure PKE scheme using the Naor-Yung double-encryption paradigm [NY90].

Liu et al. [LLGW20] also give a generic construction in the standard model following the KEM and signature approach in Figure 1. The KEM needs to satisfy multi-user multi-challenge CCA security (in SBG style). However, both constructions, [BHJ+15] and [LLGW20], rely on an MBG-style AKE security model.

AKE from Group Actions. Due to the properties of the underlying group action structure, CSIDH is a good candidate to construct tightly-secure AKE based on isogenies. In [dKGV20], de Kock, Gjøsteen and Veroni give a security proof of the triple Diffie-Hellman protocol that was also considered in [CCG+19]. Their proof applies the same techniques by relying on the GA-StCDH assumption and also achieves optimal tightness (i. e., it loses an unavoidable factor linear in the number of users). The same result was independently shown by Kawashima et al. in [KTAT20].

PAKE. Tight security for PAKE protocols has not nearly attracted as much attention as for standard AKE. In [BIO+17], Becerra et al. show that the protocol PAK [BMP00] is tightly secure under the GapCDH assumption for a single test query and weak forward security. In a similar fashion, the result by Abdalla and Barbosa [AB19] shows that weak forward security of the SPAKE2 protocol [AP05] can be proven achieved tightly under the GapCDH assumption when considering multiple test queries. In contrast to that, CPace, which was chosen for standardization, only has a non-tight security proof (in fact, it loses a quadratic factor in the number of random oracle queries) and additionally relies on the non-standard simultaneous Diffie-Hellman assumption [AHH21].

1.3 Overview of Results

This thesis improves upon previous results on tightly-secure AKE in various directions. An illustrative overview of the main contributions is given in Figure 3. In particular, we answer the following questions:

(1) Can we justify the use of standard system parameters for *existing Diffie-Hellman based protocols*, in particular HMQV and NAXOS?

(2) How can we *generically* construct tightly-secure AKE in an SBG-style security model that additionally allows for state reveals ...

  (a) ... in the *random oracle model*?

  (b) ... in the *standard model*?

(3) Can we construct a *post-quantum* and tightly-secure PAKE protocol?

Since the implicitly-authenticated protocols HMQV and NAXOS are subject to known tightness impossibility results, we use the GGM to analyze their concrete security. We provide more details in Section 1.3.1 and the full paper is given in Appendix A. Both protocols achieve weak forward security, can be executed in one round and resist state reveal attacks.

In Section 1.3.2, we then show how to generically construct two AKE protocols: $\mathsf{AKE}_{\mathsf{wFS}}$ is built from non-committing key encapsulation (NCKE) mechanisms only and achieves weak forward security, whereas $\mathsf{AKE}_{\mathsf{FS}}$ also uses a $\mathsf{MU\text{-}SUF\text{-}CMA}^{\mathsf{corr}}$ secure signature scheme and achieves full forward security. Both protocols need two rounds of communication, allow for state reveals and rely on the random oracle model. We provide more details on how to construct NCKE from hash proof systems below. The full paper is given in Appendix B. Since the NCKE construction inherently relies on the random oracle model, we want to focus on protocols in the standard model next. By adding

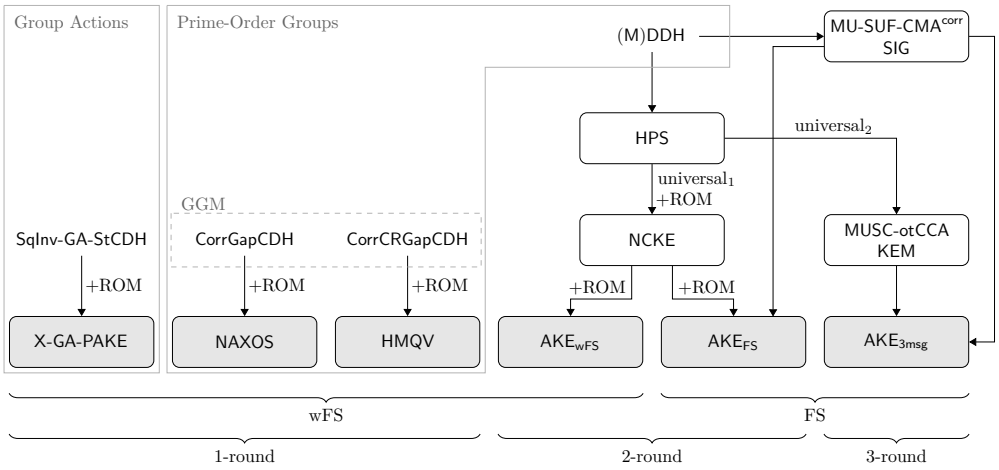

Figure 3: Overview of our results: we illustrate the construction of (P)AKE protocols (gray rounded boxes) based on their building blocks (white rounded boxes) and underlying hardness assumptions (no boxes). Below we indicate which level of forward security the protocols provide and the number of rounds.

an extra round, we show that we can construct an AKE protocol from a weaker KEM, namely a multi-user single-challenge one-time CCA (MUSC-otCCA) secure KEM. The resulting protocol is explained in Section 1.3.3 and the paper is given in Appendix C.

Finally, we show how to construct a PAKE protocol in the group action setting based on CSIDH, thus aiming for post-quantum security. For our tightly-secure one-round protocol X-GA-PAKE we require a new non-standard assumption, the square-inverse strong Diffie-Hellman (SqInv-GA-StCDH) assumption. We also discuss a second protocol which is based on a standard assumption, but requires three rounds and is non-tight. Details are provided in Section 1.3.4 and Appendix D.

1.3.1 Concrete Security of HMQV and NAXOS

In [KPRR23] (Appendix A), we analyze the concrete security of the HMQV [Kra05] and NAXOS [LLM07] protocols in the GGM. This is in particular interesting since we know that a security loss that is linear in the number of users is inherent [CCG+19]. Additionally, the original security proofs are carried out in the stronger security model with additional state reveals. Although no formal impossibility results have been shown, the underlying commitment problem indicates that a loss at least linear in the number of sessions can most likely not be avoided either. We also analyze the protocol studied in [CCG+19]. Due to the similarity to the Extended Triple Diffie-Hellman (X3DH) used in Signal, we denote this protocol by X3DH$^-$. It can also be described as the NAXOS protocol without the "NAXOS hashing trick" to derive the ephemeral secret exponent.

To give concrete security bounds in the GGM, we first extract the core of the protocols in terms of a multi-user variant of the CDH problem. With these assumptions, we can prove security in the ROM in a straightforward way with (mainly) a single

reduction. We can then analyze these new CDH problems in the GGM to derive concrete security bounds.

Multi-User CDH Problems with Corruptions. The reason why randomizing the (Gap)CDH problem is insufficient for tight security is that the reduction needs to answer adaptive Corrupt and RevealState queries of the adversary. Secret keys and states are exponents and must be known to the reduction in order to answer those queries. To address this, we define a multi-user version of the GapCDH problem with adaptive corruptions. We call this problem the CorrGapCDH problem. In order to win, the adversary has to provide the CDH solution to some combination of the input challenges which have not been corrupted. For X3DH$^-$, we tweak this problem by splitting the set of challenges into two disjoint subsets and the solution must contain one element from the first subset and one from the second subset. Further, only exponents from the first subset can be revealed via a corruption.

To mirror the HMQV protocol, where identities and messages are additionally hashed together to derive an exponent, we extend the CorrGapCDH problem by providing an additional challenge oracle. This challenge then needs to be embedded in the CDH solution. Due to the challenge-response character, we denote this problem by CorrCRGapCDH.

Proving Security of HMQV, NAXOS and X3DH$^-$. The variants of the GapCDH problem described above allow us to tightly prove the security of the three protocols in the random oracle model. By (non-tightly) relating the problems to the standard GapCDH problem, we additionally get an improvement of the original security proofs of HMQV and NAXOS which were proven in the single-challenge setting. In particular, we extend these results to the multi-challenge setting and show that the bound is independent of the number of test queries. An interesting question is whether this is due to our new proof technique or whether it would have been possible in the first place. While the typical strategy of guessing one out of S sessions in total is reasonable, guessing T possible test sessions would result in an exponential loss of $\binom{S}{T}$ for realistic choices of S and T. Thus, the naive approach of a hybrid argument and replacing challenge keys one by one seems to be the most natural proof strategy, but it also comes with a security loss linear in T.

Due to the weaker security model (without state reveals), the original bound of X3DH$^-$ does not depend on the number of (test) sessions. By using the asymmetric version of the problem, which is tightly implied by CorrGapCDH, and a case distinction in the proof, our analysis matches the (optimal) bound of [CCG+19].

Analysis in the GGM. We prove lower bounds in the GGM for both the CorrGapCDH problem and CorrCRGapCDH problem. The first one matches that of the discrete logarithm problem and is thus optimal. Although we cannot achieve the same bound for CorrCRGapCDH, we provide an analysis of the bit security for conservative numbers of oracle queries that justifies choosing standard parameters. Thus, assuming that the best attacks are indeed generic, it is not necessary to increase the size of elliptic curve groups.

1.3.2 Tightly-Secure AKE in the ROM

In [JKRS21] (Appendix B) we look at two generic AKE constructions. The first one is the generic construction from KEMs (cf. Figure 2) which we call $\mathsf{AKE}_{\mathsf{wFS}}$. The second one follows the explicitly authenticated two-round protocol in Figure 1, but for efficiency reasons, we combine it with a key confirmation hash so that only the sender needs to compute a signature. We call this protocol $\mathsf{AKE}_{\mathsf{FS}}$.

We are interested in determining a security notion for the underlying KEM when aiming for weak forward security, resistance against KCI attacks and state-reveal attacks for multiple test sessions, which allows us to prove tight security. Due to several manifestations of the commitment problem (cf. Section 1.2.4), we follow the approach of a non-committing KEM.

Non-Committing Key Encapsulation. The approach of mirroring the AKE security experiment as done in Section 1.3.1 suggests defining a multi-user multi-challenge security notion for KEMs with corruptions. However, in this naive approach, we encounter one important issue. Since we aim for security with a single challenge bit (as in the AKE SBG model), we need to restrict the adversary from getting challenges for users that are corrupted and vice versa. Otherwise, this allows the adversary to trivially distinguish the KEM keys. Looking at the AKE proof strategy, this restriction prevents us from solving the commitment problem. When a session is initiated and the involved parties are not (yet) corrupted, we cannot know whether this session will possibly be a test session. At the same time, long-term or ephemeral secrets could be corrupted at a later point. This makes it impossible to decide whether a challenge should be embedded.

To address this, we strengthen the definition of KEMs and propose the notion of (multi-receiver) non-committing key encapsulation (NCKE) schemes. An NCKE scheme is defined relative to a simulator and a random oracle. All algorithms (including those of the simulator) have access to the random oracle. The simulator ensures that, whenever the adversary obtains a challenge ciphertext and later decides to corrupt the user, the KEM key that was given as challenge is exactly the result of decapsulating the challenge ciphertext with the corrupted secret key. Although the ROM is a strong tool in itself (cf. Section 1.1.4), there must also be sufficient entropy in the secret key of the NCKE scheme when given the public key. We show that for a universal$_1$ hash proof system with a hard multi-instance subset membership problem, we can describe an NCKE simulator in the ROM. Such a hash proof system can in turn be tightly instantiated, e. g., based on the DDH assumption.

Constructing AKE from NCKE. In $\mathsf{AKE}_{\mathsf{wFS}}$ we simply replace the KEMs in Figure 2 with NCKE schemes. We use an independent random oracle to derive the shared session key from the three intermediate KEM keys and the context of the session. In the security proof, we can embed challenge keys of the NCKE scheme in all sessions at the same time. We then rely on the NCKE simulator to ensure that those keys are indistinguishable from random as long as the secret key is not corrupted and that they look valid as soon as a corruption happens.

In our second protocol $\mathsf{AKE}_{\mathsf{FS}}$, the sender uses a signature scheme and the receiver uses an NCKE scheme for authentication. Combining this with an additional key

confirmation hash allows to explicitly authenticate both parties in a two-round protocol. For forward security, we use an additional ephemeral NCKE instance in each session. The signature scheme needs to satisfy the strong notion of MU-SUF-CMA$^{\text{corr}}$ and can be instantiated using the signature scheme of Gjosteen and Jager [GJ18] which is based on CDH and DDH or the more recent DDH-based scheme of Diemert et al. [DGJL21b] which has even smaller signatures.

State Reveal Attacks. Even the strong notion of NCKE does not allow to protect against state reveal attacks. We use ideas from the NAXOS protocol and protect the ephemeral secrets additionally with the long-term key. For this, each party chooses a symmetric key as part of the long-term key which is then used to encrypt the secret state information. Since the NCKE scheme already relies on the ROM, we instantiate this encryption with a random oracle and a one-time pad which allows us to resolve the underlying commitment problem. Although this approach seems somehow generic, it is worth mentioning that on its own, a "state encryption" is not necessarily sufficient. As for long-term keys, we need some form of entropy in the ephemeral secrets which makes this approach not applicable to, e. g., the signed Diffie-Hellman protocol.

1.3.3 Tightly-Secure AKE in the Standard Model

While the approach described in the previous section highly relies on the random oracle model, our follow-up work [HJK+21] (Appendix C) focuses on tightly-secure AKE in the standard model. We start with the work by [LLGW20] which proposes an explicitly authenticated protocol based on the generic construction with a KEM and signature scheme. While their work achieves tight security in the Multi-Bit Guess security model, we develop new intermediate security notions that allow to prove security in the Single-Bit Guess model and can also be instantiated in the standard model.

In this work, we also propose a new MU-SUF-CMA$^{\text{corr}}$ secure signature scheme that addresses a flaw in the signature scheme of [BHJ+15]. The signature scheme relies on the Matrix Decisional Diffie-Hellman assumption (MDDH) in bilinear groups which can be seen as a generalization of the standard DDH assumption (cf. [EHK+17]).

Our Starting Point: Two-Round AKE. We start with the generic approach as shown in Figure 1. While it is clear that MU-SUF-CMA$^{\text{corr}}$ security is needed for the signature scheme, the exact security of the KEM depends on the model and protocol design. As long as we do not consider state reveal queries, we can show that a multi-user and multi-challenge secure KEM with exactly one decryption query (therefore denoted by MUC-otCCA security) is sufficient to achieve tight security. Unfortunately, it is unclear how to make use of the fact that only a single decryption query is allowed. Existing constructions of KEMs (e. g., [GHKW16, GHK17, HLLG19]) that are secure in the multi-user multi-challenge setting achieve full CCA security.

Adding a Nonce. By changing the protocol, we aim to weaken the security required for the KEM further. In particular, we want to reduce the number of challenges for each public key to a single one. In the previous protocol, we needed multi-challenge security

because the adversary can replay a message from the sender. Thus, the ephemeral public key contained in this message may be used multiple times. However, a simple trick can prevent this: we add a nonce that is chosen by the responder. This ties the ephemeral public key chosen by the sender to the nonce such that it cannot be replayed.

We show that the KEM security notion can now be relaxed to multi-user single-challenge one-time CCA (MUSC-otCCA) security to give a tight security proof in the standard model. We denote this protocol by $\mathsf{AKE}_{\mathsf{3msg}}$.

Adding a Symmetric Encryption Scheme. The above construction cannot achieve tight security in the presence of state reveals. Thus, when aiming for security in this model, we need to strengthen the security notion of the KEM again. Further, we borrow the "state-encryption" technique from the previous section, but we need to replace the random oracle with a symmetric encryption scheme. In order to perfectly work together with the symmetric encryption scheme, we identify two main properties for the KEM: (information-theoretic) uniformity and key indistinguishability, which we capture in the definition of multi-user simulatable (MU-SIM) security. The two properties are closely related to those of hash proof systems. In fact, we show that MU-SIM is implied by a universal$_2$ hash proof system with a multi-instance subset membership problem.

If the symmetric encryption scheme is secure against multi-challenge random plaintext attacks, the AKE protocol achieves security against state reveal attacks. The bound is tight in the advantage of the KEM and signature scheme and loses a factor linear in the number of users for the symmetric encryption scheme. In practice, increasing system parameters for symmetric-key primitives is tolerable since it does not come with a huge loss in efficiency.

Instantiating the MU-SIM KEM from MDDH and showing that every MU-SIM secure KEM is also MUSC-otCCA secure, we can instantiate both the KEM and signature scheme in $\mathsf{AKE}_{\mathsf{3msg}}$ from MDDH.

1.3.4 PAKE from Group Actions

All instantiations of AKE protocols so far relied on hardness assumptions in prime-order groups. In [AEK+22] (Appendix D), we show how to construct password-authenticated key exchange protocols from group actions. In particular, we are interested in the CSIDH group action since it is a promising candidate for post-quantum security.

However, CSIDH comes with some additional limitations, namely, we do not know how to sample a set element (i. e., a supersingular elliptic curve) obliviously. This translates to the problem of hashing into the graph of supersingular elliptic curves which has been studied in [BBD+22, MMP22] and remains an open problem. Thus, the only known way to sample an element of the set uniformly at random is to first sample a group element and then apply the group action. This prevents us from adapting known protocols such as SPEKE [Jab96] or CPace [HL19] to the CSIDH setting.

Instead of a hash function, we suggest using a CRS that consists of two random set elements. Applying a bit-by-bit approach, the parties perform a group action Diffie-Hellman key exchange for each password bit, where the bit indicates which of the two set elements in the CRS is used as a basis.

Group Actions with Twists. It turns out that the proposed protocol is insecure when instantiated with CSIDH. This is due to the additional structure of the specific group action. For each supersingular elliptic curve, we can easily compute its twist which is another related supersingular elliptic curve. Translating this property from the group action setting to the prime-order group setting, this would mean that we can efficiently compute $g^{\frac{1}{x}}$ from g^x. This property is useful for more efficient constructions (e. g., [LGd21]), but also poses a threat when relying on non-standard assumptions. In fact, twists allow for an offline dictionary attack against the proposed protocol. In the following, we describe two approaches to prevent the attack.

First Approach: Using a Commitment. The attack against the initial protocol relies on the fact that the responder can choose its message depending on the initiator's message. Thus, a natural way to prevent the attack is to let the responder commit to the message in the first flow of the protocol. This can be achieved by sending, e. g., a hash of the message. We call this protocol Com-GA-PAKE.

Modeling the hash function as a random oracle allows to prove security based on an interactive version of the simultaneous Diffie-Hellman problem which can be (non-tightly) reduced to the GA-GapCDH problem. However, we also need to apply a guessing technique to prove security of the protocol since we need to extract the adversary's commitment from the hash queries to embed our own challenge. Additionally, the protocol needs three rounds.

Second Approach: Using Cross-Terms. To improve upon the shortcomings of the first approach, we propose a second protocol that prevents the offline dictionary attack. While the protocol requires to double the number of elements that need to be sent, it can be executed in one round. The protocol, which we call X-GA-PAKE, relies on a new assumption, namely the Square-Inverse Group Action Strong Diffie-Hellman (SqInv-GA-StCDH) assumption. On input $g \star \tilde{x}$ for random $g \xleftarrow{\$} \mathcal{G}$, it requires to compute $(y, g^2 \star y, g^{-1} \star y)$, with additional access to a decision oracle. Although it is a non-standard assumption similar to the simultaneous Diffie-Hellman assumption, we can argue about its generic hardness even in the presence of the twisting functionality. The main advantage, however, is that it is a non-interactive assumption and re-randomization of the challenge allows us to tightly prove the security of X-GA-PAKE. This gives us the first tightly-secure PAKE protocol from isogenies.

Chapter 2

Discussion and Outlook

The results of this thesis mainly consider the most fundamental form of authenticated key exchange. We gave tightness results for established protocols from the literature, as well as new generic constructions. However, even for this rather simple form of protocol, there exists a large number of security models, each with slight differences and many of them are incomparable. We elaborate on the implications for security proofs and in particular tightness in Section 2.1.

We also considered password-authenticated key exchange, a variant of standard AKE. In Section 2.2, we will discuss other types of key exchange protocols with advanced functionalities, such as group key exchange or ratcheted key exchange. While our PAKE protocol relied on group actions, the other protocols were either designed for Diffie-Hellman groups directly or we instantiated the building blocks based on those. Thus, we conclude with an outlook on post-quantum tightly-secure key exchange protocols in Section 2.3. As part of this, we also want to give an intuition on how we can prove security of our PAKE protocol in the quantum random oracle model.

2.1 Variety of Security Models

Even for standard AKE, there exist numerous different security models. The main differences evolve around which level of authentication can be achieved, what information the adversary gets via reveal queries and when sessions are partnered. Thus, all of these differences influence the definition of trivial attacks as well as the exact security guarantees. While a definition that captures strong adversarial capabilities may be interesting from a theoretical point of view, one also always needs to find the balance between the practical instantiability of protocols and the motivation of attack scenarios.

Several works have tried to compare different models (e. g., [CBH05, XWF+08, Cre09a, Cre09b, dFW20]), most notably the two models by Bellare and Rogaway [BR93a, BR95b], the Canetti-Krawczyk (CK) model [CK01], its extension that was used to analyze HMQV [Kra05] and the extended Canetti-Krawczyk (eCK) model [LLM07]. It turns out that most of these models are syntactically as well as practically incomparable, making it hard to judge which is the strongest or the "correct" model. While it is clear that forward security is strictly stronger than weak forward security, leading to additional variants such as the eCK-PFS model [CF12], the most significant difference between models is which session-specific secrets can be revealed. This is

captured in a state reveal query or an ephemeral key reveal query. However, these two are incomparable [XWF+08, Cre09a, Cre09b]. Moreover, natural protocols often cannot achieve security in these models without relying on techniques that derive intermediate secrets by combining ephemeral secrets with long-term secrets as done in [LLM07, Oka07, FSXY12]. Similarly, we have to judge the implications for tight security. It is important to study the conditions for tight security in different models in order to allow for an objective evaluation of different protocols. For example, if strong assumptions are necessary to construct a more involved protocol with tight security and strong security guarantees, it might still be reasonable to employ a simpler protocol with tight security in a weaker model (e. g., without state reveals).

In this regard, we should always keep in mind that the overall goal of a key exchange protocol is to use the session key in some other application, i. e., we want to compose AKE with other primitives. So far, we only considered game-based security. However, simulation-based security is motivated exactly by strong composability guarantees. In simulation-based security notions such as [BCK98, Sho99, CK02], we consider a real-world execution and an ideal-world execution of the protocol, where the latter is modeled such that it is secure by definition. By constructing an ideal-world simulator from a real-world adversary, we can show that both worlds are computationally indistinguishable. There exist many different variants of simulation-based security like the universal composability (UC) framework [Can01], blackbox simulatability [PW01] and follow-up works addressing shortcomings of these early models for specific primitives (e. g., [CDPW07, HS15]).

Moreover, it gets a bit tricky when corruptions are considered. Simulation-based security models with static corruptions, where the adversary performs all corruptions before any honest party receives an input, are often equivalent to the corresponding game-based security notion with adaptive corruptions [Sho99, FHH14]. However, this equivalence comes with a non-tight bound since game-based security with static corruptions non-tightly implies security with adaptive corruptions. On the other hand, simulation-based security with adaptive corruptions is often strictly stronger than game-based security and may even be impossible to achieve in the standard model. This is due to the fact that the simulator in the ideal world has to resolve the commitment problem. Whenever this is considered impossible, security definitions make the restriction that no queries happen that would cause the commitment problem (cf. [FHH14]).

Especially for PAKE protocols, simulation-based security enjoys the advantage that it makes no assumption about the distribution of passwords and, most importantly, passwords can be correlated. Different formulations of UC-security for PAKE exist, e. g., [CHK+05, ABB+20, AHH21]. A programmable random oracle allows to prove security of the CPace protocol even in the presence of adaptive corruptions (cf. [AHH21]). They also provide concrete, though non-tight, bounds. Interestingly, most other works do not state concrete bounds when proving security in simulation-based models.

2.2 Advanced Key Exchange Protocols

There exist several use cases where traditional AKE is insufficient to provide the desired functionalities or does not accurately model the exact application. Examples include

the group AKE setting where multiple parties want to compute a shared session key or messaging protocols with long-lived sessions. Security models and proofs for protocols that provide these advanced functionalities are often even more complex than those for standard AKE. Thus, the focus is mostly on giving a formal proof in the first place instead of giving a *tight* proof.

Nevertheless, the techniques that have been established to achieve tight security may carry over to those settings. In [PQR22], Pan, Qian and Ringerud study the tight security of the signed Diffie-Hellman protocol in the group setting and are able to prove similar results as for the two-party case. Another promising direction is tightness for privacy-preserving, anonymous or deniable key exchange (cf. [GSU12, UG15, SSL20, RSW21, LLHG22, IY22]).

The analysis of real-world AKE protocols such as TLS is usually more complex due to different modes or additional steps that are often abstracted away, e. g., they consider pre-shared keys (PSK) and a key schedule. If a protocol consists of multiple stages, we ideally want to show security for intermediate keys and further secrets that are derived to be used in other applications. To capture this, a security model for multi-stage key exchange was introduced by Fischlin and Günther [FG14] who analyzed the security of the QUIC protocol [Ros13]. In the context of TLS, Davis et al. [DDGJ22] analyze the concrete security of the PSK mode and provide a new abstraction for the key schedule. While the key schedule is often modeled as random oracle(s), this needs additional care in order to account for dependencies between different components. A similar analysis would be interesting in the context of Signal. Cohn-Gordon et al. [CCD+20] thoroughly analyze the initial key exchange X3DH and the double ratchet protocol used in Signal [MP16, PM16]. They prove security in a multi-stage key exchange model under the GapCDH assumption and in the random oracle model. However, each key derivation function is modeled as a single random oracle and their proof is highly non-tight. The tight analysis of TLS gives hope that it may be possible to get a tight(er) bound by applying a similar approach to carefully model and patch the random oracles.

The double ratchet can also be considered as a standalone primitive. In ratcheted key exchange (RKE) as considered in [BSJ+17, PR18, JMM19, BRV20], parties do not have fixed long-term keys, but they keep a state that is updated over time. This way, we can not only get forward security but also post-compromise security, meaning that a party can recover from a corruption after some time. As security models for RKE usually model adaptive corruptions, tight security is probably hard to achieve, even in the two-party case. Considering that it is still a rather young primitive, the focus has been on formalizing the correct security properties. In this context, Alwen, Coretti and Dodis [ACD19] define the more general notion of secure messaging which they build from continuous key agreement (CKA). Recently, the group setting of CKA, denoted by continuous group key agreement (CGKA) [ACDT20], has attracted a lot of interest. As opposed to standard group key exchange, the groups can be dynamic, i. e., members can be added or removed. CGKA is the main component in the MLS protocol [BBR+22] that is being developed by the IETF as future standard for secure group messaging. It can be expected that as soon as security notions and concepts are more established, the focus will also shift towards improving the concrete security bounds of these protocols, as already raised as an open question in [ACDT20].

2.3 Post-Quantum Secure Protocols

Many tightly-secure AKE protocols rely either directly on Diffie-Hellman assumptions or define building blocks that are then instantiated based on those assumptions. Due to their limited structure, group actions are mostly not suitable to be used in these instantiations, e. g., our results from Sections 1.3.2 and 1.3.3 based on hash proof systems. The only existing hash proof system based on group actions [ADMP20] uses a similar bit-by-bit approach as our PAKE protocol and relies on a multi-instance version of GA-DDH for which we do not know a tight reduction to the single-instance version. It would be interesting to study hash proof systems from lattices in this multi-instance version, e. g., the scheme of [BBDQ18]. If we want to construct AKE from lattices using our generic constructions, it is also necessary to take into account correctness errors.

In [PW22], Pan and Wagner study tightly-secure signatures in the multi-user setting with corruptions from lattices and group actions. They refine the transformation from lossy identification schemes and sequential OR proofs by Diemert et al. [DGJL21b] such that it can also be applied in the lattice setting. However, their tightly-secure signature scheme relies on the plain Learning With Errors (LWE) assumption which, in contrast to the Module LWE or Ring LWE (cf. [Pei15]) assumption, requires larger parameter choices resulting in larger public key sizes and makes schemes rather inefficient. Their signature scheme based on group actions suffers from the same inefficiency as the hash proof system, but it avoids the loss arising from the multi-instance GA-DDH assumption by observing that lossy keys can be correlated. Thus, tight security can be achieved based on the single-instance GA-DDH assumption in the ROM. They leave it as an open problem to prove security of their transformation in the quantum random oracle model (QROM).

Our PAKE protocols as well as the AKE protocol based on CSIDH [dKGV20, KTAT20] are also only proven secure in the ROM. We believe that security in the QROM will require a similar approach as that used in our follow-up work where we prove security for hashed ElGamal based on group actions [DHK+22]. In this work, we show that a non-standard variant of the GA-StCDH assumption (where the DDH oracle can be queried in superposition) is needed to prove security of the (unmodified) scheme in the QROM. However, it is possible to prove security based on the standard GA-StCDH assumption when adding a key confirmation hash. Adding a key confirmation hash to our PAKE protocol thus not only allows a proof in the QROM, but also provides explicit authentication and (full) forward security.

Bibliography

[AB19] Michel Abdalla and Manuel Barbosa. Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194, 2019. https://eprint.iacr.org/2019/1194. 31

[ABB+20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020. 40

[ABH+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE standard. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 87–116. Springer, Heidelberg, October 2021. 15

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001. 18

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019. 41

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020. 41

[ACFK17] Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 101–132. Springer, Heidelberg, August 2017. 20

[ADMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 411–439. Springer, Heidelberg, December 2020. 16, 42

[AEK+22] Michel Abdalla, Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler, and Doreen Riepel. Password-authenticated key exchange from group actions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 699–728. Springer, Heidelberg, August 2022. 36

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Heidelberg, January 2005. 28

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Heidelberg, December 2021. 31, 40

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. 12

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, February 2005. 28, 31

[ASB14] Janaka Alawatugoda, Douglas Stebila, and Colin Boyd. Modelling after-the-fact leakage for key exchange. In Shiho Moriai, Trent Jaeger, and Kouichi Sakurai, editors, *ASIACCS 14*, pages 207–216. ACM Press, June 2014. 13

[Bad14] Christoph Bader. Efficient signatures with tight real world security in the random-oracle model. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14*, volume 8813 of *LNCS*, pages 370–383. Springer, Heidelberg, October 2014. 30

[BBD+22] Jeremy Booher, Ross Bowden, Javad Doliskani, Tako Boris Fouotsa, Steven D. Galbraith, Sabrina Kunzweiler, Simon-Philipp Merz, Christophe Petit, Benjamin Smith, Katherine E. Stange, Yan Bo Ti, Christelle Vincent, José Felipe Voloch, Charlotte Weitkämper, and Lukas Zobernig. Failing to hash into supersingular isogeny graphs. Cryptology ePrint Archive, Report 2022/518, 2022. https://eprint.iacr.org/2022/518. 36

[BBDQ18] Fabrice Benhamouda, Olivier Blazy, Léo Ducas, and Willy Quach. Hash proof systems over lattices revisited. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 644–674. Springer, Heidelberg, March 2018. 42

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000. 11, 12, 20

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 171–188. Springer, Heidelberg, May 2004. 18

[BBR+22] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-17, Internet Engineering Task Force, December 2022. Work in Progress. 10, 41

[BCGP08] Colin Boyd, Yvonne Cliff, Juan González Nieto, and Kenneth G. Paterson. Efficient one-round key exchange in the standard model. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 69–83. Springer, Heidelberg, July 2008. 26

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th ACM STOC*, pages 419–428. ACM Press, May 1998. 40

[BD20] Mihir Bellare and Wei Dai. The multi-base discrete logarithm problem: Tight reductions and non-rewinding proofs for Schnorr identification and signatures. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 529–552. Springer, Heidelberg, December 2020. 19

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011. 18

[Ber08] Daniel J. Bernstein. Proving tight security for Rabin-Williams signatures. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 70–87. Springer, Heidelberg, April 2008. 11, 20

[BFM15] Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Random-oracle uninstantiability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 428–455. Springer, Heidelberg, March 2015. 18

[BFP21] Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 587–617. Springer, Heidelberg, December 2021. 19

[BG11] Colin Boyd and Juan Manuel González Nieto. On forward secrecy in one-round key exchange. In Liqun Chen, editor, *13th IMA International Conference on Cryptography and Coding*, volume 7089 of *LNCS*, pages 451–468. Springer, Heidelberg, December 2011. 24

[Bha20] Rishiraj Bhattacharyya. Memory-tight reductions for practical key encapsulation mechanisms. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 249–278. Springer, Heidelberg, May 2020. 20

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658. Springer, Heidelberg, March 2015. 13, 25, 30, 35

[Bih02] Eli Biham. How to decrypt or even substitute des-encrypted messages in 228 steps. *Information Processing Letters*, 84(3):117–124, 2002. 12

[BIO+17] José Becerra, Vincenzo Iovino, Dimiter Ostrev, Petra Sala, and Marjan Skrobot. Tightly-secure PAK(E). In Srdjan Capkun and Sherman S. M. Chow, editors, *CANS 17*, volume 11261 of *LNCS*, pages 27–48. Springer, Heidelberg, November / December 2017. 31

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304. Springer, Heidelberg, May 2016. 11, 20, 29

[BKP14] Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (Hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 408–425. Springer, Heidelberg, August 2014. 11, 20

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 1–12. Springer, Heidelberg, August 1998. 12

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992. 10, 28

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2000. 31

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000. 12, 22, 28

[BR93a] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1993. 12, 22, 23, 39

[BR93b] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 18

[BR95a] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 92–111. Springer, Heidelberg, May 1995. 18

[BR95b] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th ACM STOC*, pages 57–66. ACM Press, May / June 1995. 22, 39

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Heidelberg, May 1996. 11, 20

[BR09] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 407–424. Springer, Heidelberg, April 2009. 11, 20

[BRV20] Fatih Balli, Paul Rösler, and Serge Vaudenay. Determining the core primitive for optimally secure ratcheting. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 621–650. Springer, Heidelberg, December 2020. 41

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650. Springer, Heidelberg, August 2017. 41

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007. 19

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. 40

[CBH05] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 585–604. Springer, Heidelberg, December 2005. 39

[CCD+20] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, October 2020. 41

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797. Springer, Heidelberg, August 2019. 14, 25, 29, 30, 31, 32, 33

[CD22] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive, Report 2022/975, 2022. https://eprint.iacr.org/2022/975. 14, 16

[CDG+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018. 18

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007. 40

[CDVW12] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Heidelberg, May 2012. 29

[CF12] Cas J. F. Cremers and Michele Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751. Springer, Heidelberg, September 2012. 39

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998. 18

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005. 40

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014. 18

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001. 12, 22, 39

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, Heidelberg, April / May 2002. 40

[CKMS16] Sanjit Chatterjee, Neal Koblitz, Alfred Menezes, and Palash Sarkar. Another look at tightness II: practical issues in cryptography. In Raphael C.-W. Phan and Moti Yung, editors, *Paradigms in Cryptology - Mycrypt 2016*, volume 10311 of *LNCS*, pages 21–55. Springer, Heidelberg, Germany, 2016. 12

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018. 14, 16

[CMS12] Sanjit Chatterjee, Alfred Menezes, and Palash Sarkar. Another look at tightness. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 293–319. Springer, Heidelberg, August 2012. 12, 21

[CMY+16] Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, and Fuchun Guo. Strongly leakage-resilient authenticated key exchange. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 19–36. Springer, Heidelberg, February / March 2016. 13

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 272–287. Springer, Heidelberg, April / May 2002. 11

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291. 16

[Cre09a] Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 20–33. Springer, Heidelberg, June 2009. 39, 40

[Cre09b] Cas J.F. Cremers. Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange. Cryptology ePrint Archive, Report 2009/253, 2009. https://eprint.iacr.org/2009/253. 39, 40

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, August 1998. 20

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002. 17

[CW13] Jie Chen and Hoeteck Wee. Fully, (almost) tightly secure IBE and dual system groups. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 435–460. Springer, Heidelberg, August 2013. 11, 20

[DDGJ22] Hannah Davis, Denis Diemert, Felix Günther, and Tibor Jager. On the concrete security of TLS 1.3 PSK mode. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 876–906. Springer, Heidelberg, May / June 2022. 13, 41

[Den02] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109. Springer, Heidelberg, December 2002. 19

[dFW20] Cyprien de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 288–303. IEEE Computer Society Press, 2020. 39

[DG21] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21, Part II*, volume 12727 of *LNCS*, pages 448–479. Springer, Heidelberg, June 2021. 13, 30

[DGJL21a] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. Digital signatures with memory-tight security in the multi-challenge setting. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 403–433. Springer, Heidelberg, December 2021. 20

[DGJL21b] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. More efficient digital signatures with tight multi-user security. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 1–31. Springer, Heidelberg, May 2021. 30, 35, 42

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 9, 26

[DHK+22] Julien Duman, Dominik Hartmann, Eike Kiltz, Sabrina Kunzweiler, Jonas Lehmann, and Doreen Riepel. Group Action Key Encapsulation and Non-Interactive Key Exchange in the QROM. In *ASIACRYPT 2022*, *LNCS*. Springer, Heidelberg, December 2022. 42

[DJ21] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments. *Journal of Cryptology*, 34(3):30, July 2021. 13, 25, 30

[dKGV20] Bor de Kock, Kristian Gjøsteen, and Mattia Veroni. Practical isogeny-based key-exchange with optimal tightness. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 451–479. Springer, Heidelberg, October 2020. 14, 31, 42

[Don17] Jason A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *NDSS 2017*. The Internet Society, February / March 2017. 10

[EHK+17] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Luis Villar. An algebraic framework for Diffie-Hellman assumptions. *Journal of Cryptology*, 30(1):242–288, January 2017. 35

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984. 20

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1193–1204. ACM Press, November 2014. 41

[FHH14] Eduarda S. V. Freire, Julia Hesse, and Dennis Hofheinz. Universally composable non-interactive key exchange. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 1–20. Springer, Heidelberg, September 2014. 40

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, February / March 2013. 20, 26

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999. 18

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 467–484. Springer, Heidelberg, May 2012. 24, 26, 40

[GCS+17] Fuchun Guo, Rongmao Chen, Willy Susilo, Jianchang Lai, Guomin Yang, and Yi Mu. Optimal security reductions for unique signatures: Bypassing impossibilities with a counterexample. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 517–547. Springer, Heidelberg, August 2017. 20

[Gen08] Rosario Gennaro. Faster and shorter password-authenticated key exchange. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 589–606. Springer, Heidelberg, March 2008. 28

[GHK17] Romain Gay, Dennis Hofheinz, and Lisa Kohl. Kurosawa-desmedt meets tight security. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 133–160. Springer, Heidelberg, August 2017. 20, 35

[GHKW16] Romain Gay, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Tightly CCA-secure encryption without pairings. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 1–27. Springer, Heidelberg, May 2016. 20, 35

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, Heidelberg, May 1999. 11

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 95–125. Springer, Heidelberg, August 2018. 13, 25, 30, 35

[GJT20] Ashrujit Ghoshal, Joseph Jaeger, and Stefano Tessaro. The memory-tightness of authenticated encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 127–156. Springer, Heidelberg, August 2020. 20

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003. 18

[GK10] Adam Groce and Jonathan Katz. A new framework for efficient password-based authenticated key exchange. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 516–525. ACM Press, October 2010. 28

[GKP18] Federico Giacon, Eike Kiltz, and Bertram Poettering. Hybrid encryption in a multi-user setting, revisited. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 159–189. Springer, Heidelberg, March 2018. 12, 21

[GKRS22] Siyao Guo, Pritish Kamath, Alon Rosen, and Katerina Sotiraki. Limits on the efficiency of (ring) LWE-based non-interactive key exchange. *Journal of Cryptology*, 35(1):1, January 2022. 26

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Heidelberg, May 2003. https://eprint.iacr.org/2003/032.ps.gz. 28

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982. 11

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 63–91. Springer, Heidelberg, December 2016. 16

[GSU12] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, 67, 02 2012. 13, 41

[GT20] Ashrujit Ghoshal and Stefano Tessaro. On the memory-tightness of hashed ElGamal. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 33–62. Springer, Heidelberg, May 2020. 20

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990. 23

[HHK18] Julia Hesse, Dennis Hofheinz, and Lisa Kohl. On tightly secure non-interactive key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 65–94. Springer, Heidelberg, August 2018. 20

[HHKL21] Julia Hesse, Dennis Hofheinz, Lisa Kohl, and Roman Langrehr. Towards tight adaptive security of non-interactive key exchange. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 286–316. Springer, Heidelberg, November 2021. 20

[HJ12] Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 590–607. Springer, Heidelberg, August 2012. 11, 20

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 66–83. Springer, Heidelberg, May 2012. 11

[HJK+21] Shuai Han, Tibor Jager, Eike Kiltz, Shengli Liu, Jiaxin Pan, Doreen Riepel, and Sven Schäge. Authenticated key exchange and signatures with tight security in the standard model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 670–700. Springer, Heidelberg, August 2021. 30, 35

[HKSU20] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 389–422. Springer, Heidelberg, May 2020. 26

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384. 10, 36

[HLG21] Shuai Han, Shengli Liu, and Dawu Gu. Key encapsulation mechanism with tight enhanced security in the multi-user setting: Impossibility result and optimal tightness. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 483–513. Springer, Heidelberg, December 2021. 29

[HLLG19] Shuai Han, Shengli Liu, Lin Lyu, and Dawu Gu. Tight leakage-resilient CCA-security from quasi-adaptive hash proof system. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 417–447. Springer, Heidelberg, August 2019. 20, 35

[HNS+21] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy*, pages 304–321. IEEE Computer Society Press, May 2021. 27

[HS15] Dennis Hofheinz and Victor Shoup. GNUC: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, July 2015. 40

[IEE09] IEEE Standards Association. IEEE standard specification for password-based public-key cryptographic techniques. *IEEE Std 1363.2-2008*, pages 1–140, 2009. 10

[IY22] Ren Ishibashi and Kazuki Yoneyama. Post-quantum anonymous one-sided authenticated key exchange without random oracles. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022*, volume 13178 of *LNCS*, pages 35–65. Springer, Heidelberg, Germany, 2022. 13, 41

[Jab96] David P Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996. 28, 36

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011. 14, 15

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146. Springer, Heidelberg, October 2021. 34

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018. 10

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Heidelberg, May 2019. 41

[Kan97] Ernst Kani. The existence of curves of genus two with elliptic differentials. *Journal of Number Theory*, 64(1):130–161, 1997. 16

[KHN+14] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, October 2014. 26

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 537–553. Springer, Heidelberg, April 2012. 11

[KM19] Neal Koblitz and Alfred Menezes. Critical perspectives on provable security: Fifteen years of "another look" papers. Cryptology ePrint Archive, Report 2019/1336, 2019. https://eprint.iacr.org/2019/1336. 12

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, Heidelberg, May 2001. 28

[KPRR23] Eike Kiltz, Jiaxin Pan, Doreen Riepel, and Magnus Ringerud. Multi-user CDH problems and the concrete security of NAXOS and HMQV. In *CT-RSA 2023*, *LNCS*. Springer, Heidelberg, 2023. 32

[Kra03] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, Heidelberg, August 2003. 10, 26

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005. 12, 13, 22, 23, 24, 27, 29, 32, 39

[KSSS22] Neal Koblitz, Subhabrata Samajder, Palash Sarkar, and Subhadip Singha. Concrete analysis of approximate ideal-SIVP to decision ring-LWE reduction. Cryptology ePrint Archive, Report 2022/275, 2022. `https://eprint.iacr.org/2022/275`. 12

[KTAT20] Tomoki Kawashima, Katsuyuki Takashima, Yusuke Aikawa, and Tsuyoshi Takagi. An efficient authenticated key exchange from random self-reducibility on CSIDH. In Deukjo Hong, editor, *ICISC 20*, volume 12593 of *LNCS*, pages 58–84. Springer, Heidelberg, December 2020. 14, 20, 31, 42

[Kup05] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005. 14, 16

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, Heidelberg, December 2009. 28

[LGd21] Yi-Fu Lai, Steven D. Galbraith, and Cyprien de Saint Guilhem. Compact, efficient and UC-secure isogeny-based oblivious transfer. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 213–241. Springer, Heidelberg, October 2021. 37

[LHT16] A. Langley, M. Hamburg, and S. Turner. Elliptic curves for security. RFC 7748, RFC Editor, January 2016. 15

[LJYP14] Benoît Libert, Marc Joye, Moti Yung, and Thomas Peters. Concise multi-challenge CCA-secure encryption and signatures with almost tight security. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 1–21. Springer, Heidelberg, December 2014. 11, 20

[LLGW20] Xiangyu Liu, Shengli Liu, Dawu Gu, and Jian Weng. Two-pass authenticated key exchange with explicit authentication and tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 785–814. Springer, Heidelberg, December 2020. 25, 30, 35

[LLHG22] You Lyu, Shengli Liu, Shuai Han, and Dawu Gu. Privacy-preserving authenticated key exchange in the standard model. In *ASIACRYPT 2022*, *LNCS*. Springer, Heidelberg, December 2022. 13, 41

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007. 12, 13, 22, 24, 29, 32, 39, 40

[LM06] Kristin Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 378–394. Springer, Heidelberg, April 2006. 29

[LP20a] Roman Langrehr and Jiaxin Pan. Tightly secure hierarchical identity-based encryption. *Journal of Cryptology*, 33(4):1787–1821, October 2020. 20

[LP20b] Roman Langrehr and Jiaxin Pan. Unbounded HIBE with tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 129–159. Springer, Heidelberg, December 2020. 20

[LS17] Yong Li and Sven Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1343–1360. ACM Press, October / November 2017. 23

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 58–76. Springer, Heidelberg, May 2014. 11

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005. 19

[MM22] Luciano Maino and Chloe Martindale. An attack on SIDH with arbitrary starting curve. Cryptology ePrint Archive, Report 2022/1026, 2022. https://eprint.iacr.org/2022/1026. 14, 16

[MMP22] Marzio Mula, Nadir Murru, and Federico Pintore. Random sampling of supersingular elliptic curves. Cryptology ePrint Archive, Report 2022/528, 2022. https://eprint.iacr.org/2022/528. 36

[MP16] Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol. 2016. https://signal.org/docs/specifications/x3dh/x3dh.pdf. 10, 41

[Nat13] National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS Publication 186-4, July 2013. 15

[Nie01] Jesper Buus Nielsen. Non-committing encryption is too easy in the random oracle model. 8, Dec. 2001. 20

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002. 20

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. 30

[Oka07] Tatsuaki Okamoto. Authenticated key exchange and key encapsulation in the standard model (invited talk). In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 474–484. Springer, Heidelberg, December 2007. 24, 40

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999. 11

[Pei15] Chris Peikert. A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939, 2015. https://eprint.iacr.org/2015/939. 42

[Per17] Trevor Perrin. The noise protocol framework, 2017. http://noiseprotocol.org/noise.html. 10

[Pet17] Christophe Petit. Faster algorithms for isogeny problems using torsion point images. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 330–353. Springer, Heidelberg, December 2017. 16

[PM16] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. 2016. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf. 41

[PQR21] Jiaxin Pan, Chen Qian, and Magnus Ringerud. Signed diffie-hellman key exchange with tight security. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 201–226. Springer, Heidelberg, May 2021. 13, 30

[PQR22] Jiaxin Pan, Chen Qian, and Magnus Ringerud. Signed (group) diffie–hellman key exchange with tight security. *Journal of Cryptology*, 35, 2022. 41

[PR18] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018. 41

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996. 18

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *2001 IEEE Symposium on Security and Privacy*, pages 184–200. IEEE Computer Society Press, May 2001. 40

[PW17] David Pointcheval and Guilin Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17*, pages 301–312. ACM Press, April 2017. 19

[PW22] Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022*, volume 13178 of *LNCS*, pages 347–378. Springer, Heidelberg, Germany, 2022. 42

[RD08] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. 10

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. 12

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. 10, 26

[Rob22] Damien Robert. Breaking SIDH in polynomial time. Cryptology ePrint Archive, Report 2022/1038, 2022. https://eprint.iacr.org/2022/1038. 14, 16

[Ros13] Jim Roskind. QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport over UDP, 2013. https://docs.google.com/document/d/1RNHkx$_{\text{V}}$vKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/. 41

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145. 16

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. 9

[RSW21] Sebastian Ramacher, Daniel Slamanig, and Andreas Weninger. Privacy-preserving authenticated key exchange: Stronger privacy and generic constructions. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part II*, volume 12973 of *LNCS*, pages 676–696. Springer, Heidelberg, October 2021. 13, 41

[Sch11] Sven Schäge. Tight proofs for signature schemes without random oracles. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 189–206. Springer, Heidelberg, May 2011. 11, 20

[Sch15] Sven Schäge. TOPAS: 2-pass key exchange with full perfect forward secrecy and optimal communication complexity. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1224–1235. ACM Press, October 2015. 24

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994. 14, 15

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 19

[Sho99] Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120, IBM, 1999. 26, 40

[SSL20] Sven Schäge, Jörg Schwenk, and Sebastian Lauer. Privacy-preserving authenticated key exchange and the case of IKEv2. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 567–596. Springer, Heidelberg, May 2020. 13, 41

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1461–1480. ACM Press, November 2020. 27

[UG15] Nik Unger and Ian Goldberg. Deniable key exchanges for secure messaging. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1211–1223. ACM Press, October 2015. 13, 41

[Ust08] Berkant Ustaoglu. Obtaining a secure and efficient key agreement protocol for (h)mqv and naxos. *Designs, Codes, and Cryptography*, 46:329–342, 03 2008. 29

[WMHT18] Yuyu Wang, Takahiro Matsuda, Goichiro Hanaoka, and Keisuke Tanaka. Memory lower bounds of reductions revisited. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 61–90. Springer, Heidelberg, April / May 2018. 20

[XWF+08] Jinyue Xia, Jiandong Wang, Liming Fang, Yongjun Ren, and Shizhu Bian. Formal proof of relative strengths of security between ECK2007 model and other proof models for key agreement protocols. Cryptology ePrint Archive, Report 2008/479, 2008. https://eprint.iacr.org/2008/479. 39, 40

[YZ13] Andrew Chi-Chih Yao and Yunlei Zhao. OAKE: a new family of implicitly authenticated Diffie-Hellman protocols. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1113–1128. ACM Press, November 2013. 29

[Zav12] G.M. Zaverucha. Hybrid encryption in the multi-user setting. Cryptology ePrint Archive, Report 2012/159, 2012. https://eprint.iacr.org/2012/159. 12, 21

[Zha22] Mark Zhandry. Augmented random oracles. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 35–65. Springer, Heidelberg, August 2022. 18

Part II

Publications

List of Publications

Below is the list of papers in this thesis which are also included in Appendices A to D. A list of additional contributions is given on the next page.

Publications in this Thesis

[AEK+22] Michel Abdalla, Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler, and Doreen Riepel. Password-Authenticated Key Exchange from Group Actions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 699–728. Springer, Heidelberg, August 2022.

[HJK+21] Shuai Han, Tibor Jager, Eike Kiltz, Shengli Liu, Jiaxin Pan, Doreen Riepel, and Sven Schäge. Authenticated Key Exchange and Signatures with Tight Security in the Standard Model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 670–700. Springer, Heidelberg, August 2021.

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-Secure Authenticated Key Exchange, Revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146. Springer, Heidelberg, October 2021.

[KPRR23] Eike Kiltz, Jiaxin Pan, Doreen Riepel, and Magnus Ringerud. Multi-User CDH Problems and the Concrete Security of NAXOS and HMQV. In *CT-RSA 2023*, *LNCS*. Springer, Heidelberg, 2023.

Other Contributions

[ABH+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE Standard. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 87–116. Springer, Heidelberg, October 2021.

[DHK+22] Julien Duman, Dominik Hartmann, Eike Kiltz, Sabrina Kunzweiler, Jonas Lehmann, and Doreen Riepel. Group Action Key Encapsulation and Non-Interactive Key Exchange in the QROM. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of LNCS, pages 36–66. Springer, Heidelberg, December 2022.

[DHK+23] Julien Duman, Dominik Hartmann, Eike Kiltz, Sabrina Kunzweiler, Jonas Lehmann, and Doreen Riepel. Generic Group Action Models. In *PKC 2023*, *LNCS*. Springer, Heidelberg, 2023.

[DHRR22] Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly Anonymous Ratcheted Key Exchange. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of LNCS, pages 119–150. Springer, Heidelberg, December 2022.

[ERC+22] Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. Verifiable and Provably Secure Machine Unlearning. *CoRR*, abs/2210.09126, 2022.

[RW22] Doreen Riepel and Hoeteck Wee. FABEO: Fast Attribute-Based Encryption with Optimal Security. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2491–2504. ACM Press, November 2022.

Appendix A

Multi-User CDH Problems and the Concrete Security of NAXOS and HMQV

An extended abstract of this article will appear in the proceedings of CT-RSA 2023. The following version is the full version of this article which is also available in the IACR ePrint archive, ia.cr/2023/115.

Original Publication

E. Kiltz, J. Pan, D. Riepel, M. Ringerud. Multi-User CDH Problems and the Concrete Security of NAXOS and HMQV. In *CT-RSA 2023*, *LNCS*. Springer, Heidelberg, 2023.

Multi-User CDH Problems and the Concrete Security of NAXOS and HMQV

Eike Kiltz[1], Jiaxin Pan[2], Doreen Riepel[1,3], Magnus Ringerud[2]

[1] Ruhr-Universität Bochum, Bochum, Germany
{eike.kiltz,doreen.riepel}@rub.de
[2] NTNU – Norwegian University of Science and Technology, Trondheim, Norway
{jiaxin.pan,magnus.ringerud}@ntnu.no
[3] University of California San Diego, San Diego, USA

Abstract. We introduce CorrGapCDH, the Gap Computational Diffie-Hellman problem in the multi-user setting with Corruptions. In the random oracle model, our assumption *tightly* implies the security of the authenticated key exchange protocols NAXOS in the eCK model and (a simplified version of) X3DH without ephemeral key reveal. We prove hardness of CorrGapCDH in the generic group model, with *optimal bounds* matching the one of the discrete logarithm problem. We also introduce CorrCRGapCDH, a stronger Challenge-Response variant of our assumption. Unlike standard GapCDH, CorrCRGapCDH implies the security of the popular AKE protocol HMQV in the eCK model, *tightly* and *without rewinding*. Again, we prove hardness of CorrCRGapCDH in the generic group model, with (almost) optimal bounds.
Our new results allow implementations of NAXOS, X3DH, and HMQV without having to adapt the group sizes to account for the tightness loss of previous reductions. As a side result of independent interest, we also obtain modular and simple security proofs from standard GapCDH with tightness loss, improving previously known bounds.

Keywords: Authenticated key exchange, HMQV, NAXOS, X3DH, generic hardness.

1 Introduction

Authenticated key exchange (AKE) is a fundamental cryptographic protocol where two users agree on a joint session key. In a simple and efficient blueprint of Diffie-Hellman protocols, Alice (holding long-term key g^a) sends a random ephemeral key g^x to Bob; Bob (holding long-term key g^b) sends a random ephemeral key g^y to Alice. After receiving their input, both users derive the joint session key K from the four Diffie-Hellman values $g^{ab}, g^{ay}, g^{xy}, g^{bx}$. The practically relevant protocols HMQV [Kra05], NAXOS [LLM07], and X3DH$^-$ [CCG+19] (a simplification of Extended Triple Diffie-Hellman X3DH [MP16]) fall into this class of Diffie-Hellman protocols, see Figure 1. They are all two message protocols with implicit authentication, namely, only the designated users can share the same key and together with a MAC they can confirm their session keys and authenticate each other explicitly.

We highlight that HMQV is the well-known "provably secure" variant of MQV [MQV95, LMQ+03] which is included in the IEEE P1363 standard for key exchange [P1300]. X3DH$^-$ is essentially the Extended Triple Diffie-Hellman (X3DH) key exchange protocol without involving any signature and ignoring the server. The original X3DH protocol is used for the initial key exchange in Signal, where the receiver publishes (signed) prekeys on a server which can be retrieved (asynchronously) by the sender. The NAXOS protocol is X3DH$^-$ combined with the "NAXOS hashing trick" which is marked with a dashed box in Figure 1.

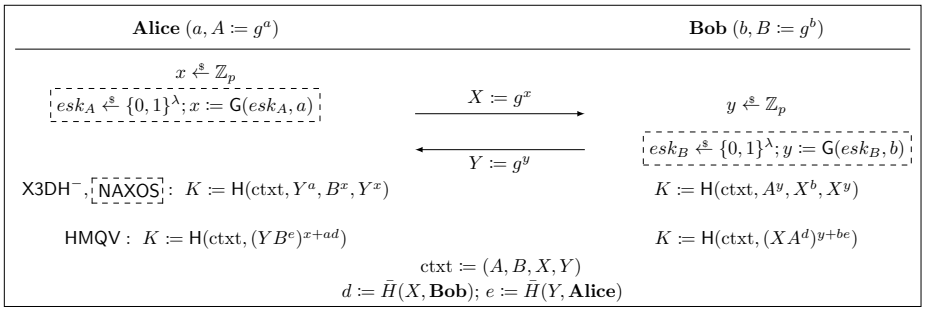

Figure 1: Overview of different AKE protocols, HMQV, X3DH$^-$, and NAXOS. NAXOS computes exponents x and y as shown in the dashed box. We make a small twist to HMQV that includes the context ctxt in computing the session key K. This twist is to avoid the trivial winning of an adversary in the eCK model (see Section 6) and is also applied in the analysis of [BCLS15].

AKE Security Model. Adversaries against AKE protocols can control all messages transferred among involved users, and they can also reveal some of the shared session keys and the long-term secret keys of honest users. These capabilities are captured by security models such as [BR94, CK01, LLM07]. The goal of an adversary is to distinguish a non-revealed session key from a random key of the same length. We use the extended Canetti-Krawczyk (eCK) model [BR94, CK01, LLM07] in a game-based formulation of [JKRS21] that allows adversaries to register dishonest users, corrupt long-term secret keys of the $N \geq 2$ honest users, reveal ephemeral states and session keys of the S sessions. The adversary is allowed to make T test queries based on the same random bit b. It captures weak forward secrecy (which is the strongest forward secrecy a two-pass implicit AKE protocol can achieve [Kra05]) and security against key-compromise impersonation (KCI) attacks and reflection attacks. We stress that our model is using a single challenge bit and hence allows for tight composition of the AKE with symmetric primitives [CCG+19].

Tightness. The security of AKE protocols is usually established by a security reduction. More precisely, for any adversary $\mathcal{A}$ against an AKE protocol with success probability $\varepsilon^{\mathsf{AKE}}$, there exists an adversary $\mathcal{B}$ with roughly the same running time that breaks the underlying assumption with probability $\varepsilon^{\mathsf{Ass}} = \varepsilon^{\mathsf{AKE}}/\ell$. The security loss ℓ plays an important role in choosing the system parameters. If ℓ is large, one has to increase the

size of the underlying group $\mathbb{G}$ to account for the security loss. Optimally, ℓ is a small constant in which case we call the reduction *tight*.

Security proofs for AKE protocols are rather complex and the resulting bounds are highly non-tight [Kra05, LLM07, CCG+19, Ust08, SE16, PW11]. A reduction $\mathcal{B}$ usually makes several case distinctions and, by guessing the behavior of an adversary in each case, $\mathcal{B}$ embeds a problem instance into either the protocol transcripts or the users' public keys. In the end, this guessing strategy ends up with a large security loss. Most of the AKE protocols lose a linear (or even quadratic) factor in the number of users N, the number of sessions S, and the number of test sessions T. Even worse, HMQV and its variants (such as [Ust08, SE16, PW11]) additionally require the Forking Lemma [PS00] to rewind the adversary and bound its success probability, which ends up with an even larger security loss. X3DH$^-$ is a noteworthy exception because it loses only a linear factor in N [CCG+19]. This linear loss in N is shown to be optimal for a large class of Diffie-Hellman protocols [CCG+19], including our simple blueprint of Diffie-Hellman protocols.

1.1 Our Contributions

In this paper, we simplify the difficulty of proving AKE protocols by introducing new variants of the Computational Diffie-Hellman (CDH) problem in the multi-user setting:

- We introduce n-CorrGapCDH, the Gap Computational Diffie-Hellman problem in an n-user setting with Corruptions. The hardness of $(N + S)$-CorrGapCDH *tightly* implies the security of NAXOS and X3DH$^-$.
- We introduce (n, Q_{CH})-CorrCRGapCDH, a stronger Challenge-Response variant of n-CorrGapCDH. The hardness of $(N + S, Q_{\mathrm{RO}})$-CorrCRGapCDH *tightly* implies the security of HMQV without rewinding.

Recall that in the eCK model the variables N, S, T, and Q_{RO} correspond to the number of users, sessions, test queries, and random oracle queries, respectively. For NAXOS and HMQV, we prove security with state corruptions. For X3DH$^-$, state corruption is not allowed, since it will lead to a trivial attack.

We prove our new assumptions based on the Gap Diffie-Hellman (GapCDH) assumption [OP01, ABR01] via non-tight reductions. Combined with these non-tight reductions, we give simple, intuitive and modular security proofs of X3DH$^-$, NAXOS and HMQV. For NAXOS and HMQV, we obtain tighter security bounds, and for X3DH$^-$ we match the optimal bound from [CCG+19]. Our results in the random oracle model are summarized in Figure 2.[1]

The main novelty of our new multi-user CDH assumptions lies in their practical applicability. We show the quantitative hardness of CorrGapCDH in the Generic Group Model (GGM) [Sho97, Mau05], which is optimal and matches the one of the discrete

[1] Our new and previously known bounds for HMQV in Figure 2 are stated in the eCK model disallowing reflection attacks. The reason is that for reflection attacks, one additionally requires the hardness of Square Diffie-Hellman (i.e., compute g^{a^2} from g^a) which is non-tightly equivalent to CDH. We remark that our generic group bounds from Figure 3 can be shown in the full eCK model allowing reflection attacks.

	wFS	St	Security tightly implied by	Security loss wrt. GapCDH Old	New
NAXOS	√	√	$(N+S)$-CorrGapCDH	$T(N+S)^2$	$(N+S)^2$
X3DH$^-$	√	–	$(N+S,N)$-CorrAGapCDH	N	N
HMQV	√	√	$(N+S,Q_{\mathrm{RO}})$-CorrCRGapCDH	$Q_{\mathrm{RO}}T(N+S)^2$	$Q_{\mathrm{RO}}(N+S)^2$

Figure 2: Security of the AKE protocols NAXOS, X3DH$^-$, and HMQV in the eCK model. St stands for state reveal attacks and wFS stands for weak forward secrecy. The "Security tightly implied by" column names the *new multi-user problem* which tightly implies the AKE's security. The last two columns contain old and new security loss for the AKE protocols relative to the *standard* GapCDH *problem*, ignoring constants. HMQV additionally incorporates the $\sqrt{\varepsilon^{\mathsf{GapCDH}}}$ loss due to the Forking Lemma.

logarithm problem. We also prove the hardness of CorrCRGapCDH in the GGM and it is (almost) optimal. Our new results in the GGM support the implementation of NAXOS, X3DH$^-$, and HMQV without increasing the group sizes to compensate the security loss of the previous reductions. Our results in the generic group model are summarized in Figure 3.

1.2 Multi-User CDH with Corruptions

Let $\mathsf{par} = (p, g, \mathbb{G})$ be system parameters that describe a group $\mathbb{G}$ of prime order $p = |\mathbb{G}|$ and a generator g of $\mathbb{G}$. Given g^{a_1}, g^{a_2}, the standard GapCDH problem (over par) requires to compute the Diffie-Hellman key $g^{a_1 a_2}$ [OP01, ABR01]. Here Gap stands for the presence of a (decisional) Gap Oracle which on input $(X = g^x, Y = g^y, Z = g^z)$ returns 1 iff $xy = z \bmod p$. We now describe our new assumptions in more details. Formal definitions will be given in Section 3.

Multi-User GapCDH With Corruptions. For $n \geq 2$, the n-user GapCDH problem with Corruptions (n-CorrGapCDH) is a natural generalization of GapCDH to the n-user setting. The adversary is given the n-tuple $(g^{a_1}, \ldots, g^{a_n})$ and is allowed to corrupt any user i to obtain its secret a_i. In order to win, it must output any of the $n(n-1)$ possible Diffie-Hellman keys $g^{a_i a_j}$ for two non-corrupted users $i \neq j$. Even though the two assumptions are asymptotically equivalent, they are quantitatively different: Due to the corruptions, one can only prove the non-tight bound $\varepsilon^{\mathsf{CorrGapCDH}} \leq O(n^2) \cdot \varepsilon^{\mathsf{GapCDH}}$.

For $n_1 \leq n$, we also consider an Asymmetric version of this assumption called (n, n_1)-CorrAGapCDH. It is asymmetric in the sense that n_1 splits the set of users $[n]$ in two disjoint sets $[n_1]$ and $[n_1 + 1, n]$, where only the first n_1 users can be corrupted. The adversary has to output any of the Diffie-Hellman keys $g^{a_i a_j}$ for two non-corrupted users $i \in [n_1]$ and $j \in [n_1 + 1, n]$. Note that CorrGapCDH tightly implies CorrAGapCDH. However, the fact that the challenge set is split asymmetrically allows us to give a tighter relation to GapCDH. In particular, we prove that $\varepsilon^{\mathsf{CorrAGapCDH}} \leq O(n_1) \cdot \varepsilon^{\mathsf{GapCDH}}$.

Multi-User Challenge-Response GapCDH With Corruptions. The (n, Q_{CH})-CorrCRGapCDH problem is a generalization of n-CorrGapCDH, where the adversary is

additionally given Q_{CH} many challenge-response pairs (R_k, h_k), for adaptively chosen $R_k \in \mathbb{G}$. To win, the adversary must output any of the $n(n-1)Q_{\text{CH}}$ possible Diffie-Hellman Challenge-Response keys $g^{a_i a_j h_k} \cdot R_k^{a_j}$ for two non-corrupted users $i \neq j$.

Another interpretation of the CorrCRGapCDH problem stems from canonical (three-round) identification schemes (a.k.a. Σ protocols) with a designated Verifier, where the Prover (holding secret key a_j) sends commitment R_k, the Verifier (holding secret key a_i) responds with a random challenge h_k, and finally the Prover sends the response $C = g^{a_i a_j h_k} \cdot R_k^{a_j}$. In this setting, the CorrCRGapCDH problem can be seen as an n-user version with corruptions of Parallel IMPersonification against Key-Only Attack (PIMP-KOA) [KMP16].

The interpretation in the context of identification schemes gives a hint that the (n, Q_{CH})-CorrCRGapCDH problem is again of qualitatively different nature than GapCDH and n-CorrGapCDH. Using techniques from [KMP16], one can prove that GapCDH and (n, Q_{CH})-CorrCRGapCDH are asymptotically equivalent. However, since the proof involves the Forking Lemma [PS00], the resulting bound $\varepsilon^{\mathsf{CorrCRGapCDH}} \leq Q_{\text{CH}} n^2 \cdot \sqrt{\varepsilon^{\mathsf{GapCDH}}}$ is highly non-tight.

Generic Hardness. In the generic group model (GGM) [Sho97], the running time of an adversary is captured by the number of queries to a group operation oracle. Ignoring constants, the advantages of an adversary making Q_{OP} group operations to a generic group of order p are upper bounded by

$$\varepsilon^{\mathsf{CorrCRGapCDH}} \leq \frac{(Q_{\text{OP}} + n)^2}{p} + \frac{n^2 Q_{\text{CH}}}{p} \tag{1}$$

$$\varepsilon^{\mathsf{CorrGapCDH}} \leq \frac{(Q_{\text{OP}} + n)^2}{p}. \tag{2}$$

We note that $\varepsilon^{\mathsf{CorrGapCDH}}$ is the same as the generic hardness of the standard discrete logarithm (DL) problem in [Sho97]. The generic hardness of CorrAGapCDH follows from that of CorrGapCDH.

1.3 Concrete Security of AKE Protocols

We will now state the concrete security bounds of the AKE protocols in the eCK model which depend on the number of users $N \geq 2$, the total number of sessions $S \geq 0$, the total number of test queries $T \geq 0$, and the number of random oracle queries Q_{RO}.

Concrete Bounds from GapCDH. We summarize the previously known and our security loss of NAXOS, X3DH$^-$, and HMQV relative to GapCDH in Figure 2. For HMQV [Kra05], we could not identify a concrete security bound in the literature so we had to estimate it from [Kra05, BCLS15] and the one of CMQV [Ust08]. The original bounds of NAXOS and HMQV are proven in a model that allows only a single test query. The bounds from Figure 2 are derived using a hybrid argument inducing a multiplicative factor of T, the number of test queries.

We stress that the multiplicative factor T seems to be unavoidable using the original proof strategies of NAXOS [LLM07] and HMQV [Kra05]. Even using the random self reducibility of CDH, these strategies still need to guess T possible test sessions out of

	Old GGM Bounds		New GGM Bounds	
	$\varepsilon^{\mathsf{AKE}}(t_{\text{OFF}}, t_{\text{ON}})$	**Bit security**	$\varepsilon^{\mathsf{AKE}}(t_{\text{OFF}}, t_{\text{ON}})$	**Bit security**
NAXOS	$\frac{t_{\text{ON}}^3 t_{\text{OFF}}^2}{p}$	32	$\frac{t_{\text{OFF}}^2}{p}$	128
X3DH$^-$	$\frac{t_{\text{ON}} t_{\text{OFF}}^2}{p}$	96	$\frac{t_{\text{OFF}}^2}{p}$	128
HMQV	$\frac{t_{\text{ON}}^3 t_{\text{OFF}}^2}{\sqrt{p}}$	0	$\frac{t_{\text{OFF}}^2 + t_{\text{ON}}^2 t_{\text{OFF}}}{p}$	128

Figure 3: Security bounds in the GGM, where $t_{\text{OFF}} = Q_{\text{OP}} + Q_{\text{RO}}$ counts the number of offline queries and $t_{\text{ON}} = N + S + T$ counts the number of online queries. The "Bit security" columns refer to the bit security supported by the respective bounds over generic groups of order $p \approx 2^{256}$ and assuming $t_{\text{ON}} \approx 2^{32}$ and $t_{\text{OFF}} \lesssim 2^{128}$.

S many sessions in total, resulting in an exponential loss of $\binom{S}{T}$. Thus, the best way is to apply a hybrid argument and replace the keys one by one for each test query, which results in the security loss T. Our new assumptions resolve this issue and allow us to get rid of the factor T. In particular, we can replace the session keys of all T test sessions at once as the reduction can embed challenge instances in all sessions and then adaptively choose which instance to solve, while allowing corruptions from adversaries.

We believe that improving the bound by the factor T is relevant in practice. When combining session keys with a symmetric primitive, security should still hold for many sessions, thus T can be about 2^{30}, e.g. in modern messaging applications.

Concrete Bounds in the GGM. The main novelty of our multi-user CDH problems is that they allow us to give *optimal* security bounds for NAXOS, X3DH$^-$, and HMQV in the GGM. Our bounds in the eCK security model depend on the number of honest users N, sessions S, test sessions T, random oracle queries Q_{RO}, and generic group operations Q_{OP} made by the adversary. Since N, S, and T correspond to "online queries", we will merge them into one single value $t_{\text{ON}} = N + S + T$, the time adversary $\mathcal{A}$ spends on online queries. Similarly, $t_{\text{OFF}} = Q_{\text{RO}} + Q_{\text{OP}}$ counts the time that adversary $\mathcal{A}$ spends on "offline queries". (The reason is that offline queries are considerably less expensive than online queries, see below.) Figure 3 summarizes the security bounds in the GGM expressed as functions in $t_{\text{ON}}, t_{\text{OFF}}$.

We now explain the bounds for NAXOS in more detail. According to Figure 2, its security is tightly implied by $(N+S)$-CorrGapCDH. This means that in practice one can just pick a group $\mathbb{G}$ where the $(N+S)$-CorrGapCDH problem is hard (say, with 128-bit security) and implementing NAXOS in $\mathbb{G}$ directly gives us the same level of security (namely, 128-bit security) without increasing the group size. Applying (2) and using that $Q_{\text{OP}} \geq (N+S)$, the quantitative hardness of NAXOS in the GGM is $(Q_{\text{OP}}+N+S)^2/p = t_{\text{OFF}}^2/p$. This is *optimal* in the sense that it matches the generic bounds on the best attack on NAXOS (which computes one DL and breaks the scheme). From previously known reductions [LLM07], one can only obtain the weaker GGM bound $T(N+S)^2(Q_{\text{OP}}+N+S)^2/p = t_{\text{ON}}^3 t_{\text{OFF}}^2/p$. As for a concrete comparison, we compute the bit security offered by NAXOS when implemented over prime-order elliptic curves with $\log(p) = 256$. According to [CKMS16], a scheme offers a security level of κ bits if $\varepsilon/(t_{\text{ON}}+t_{\text{OFF}}) \leq 2^{-\kappa}$ for all adversaries running in time $t_{\text{ON}} + t_{\text{OFF}}$ where $1 \leq t_{\text{ON}} + t_{\text{OFF}} \leq 2^{\kappa}$.

A simple computation shows that our new bounds offer $\kappa = 128$ bits security as long as $t_{\text{ON}} + t_{\text{OFF}} \leq 2^{128}$. Using the bound from previously known proofs, one obtains a provable security guarantee of $128 - 3\log_2(t_{\text{ON}})$ bits. Using the conservative $t_{\text{ON}} = 2^{32}$ [CCG+19], this makes only 32 bits. Since $(N+S, N)$-CorrAGapCDH implies $(N+S)$-CorrGapCDH, the computations for X3DH$^-$ are similar. The old GGM bound is obtained from the bound in [CCG+19] which has a security loss linear in N.

The same computation shows that the quantitative hardness of HMQV in the GGM is $(Q_{\text{OP}}+N+S)^2/p + (N+S)^2(Q_{\text{RO}}+1)/p = (t_{\text{OFF}}^2 + t_{\text{ON}}^2 t_{\text{OFF}})/p$. Hence HMQV over prime-order elliptic curves of size $\log(p) = 256$ offers a security of 128 bits as long as $t_{\text{ON}} \leq 2^{64}$. In contrast, from previously known proofs one can only obtain $t_{\text{ON}}^3 t_{\text{OFF}}^2/\sqrt{p}$ which means that we are left with -96 bits of security (meaning zero). If, to guarantee 128 bits of security, group sizes were chosen according to this bound, they would be quite large, and the scheme correspondingly slow.

1.4 Discussion and Prior Work

We showed that for HMQV, X3DH$^-$, and NAXOS one can pay the price of stronger cryptographic assumptions for the benefit of getting tighter bounds. One might argue that our new assumptions partly "abstract away" the looseness of prior proofs and moreover come very close to a tautology of the AKE's security. While there is certainly some truth to the first statement, we would like to stress that our AKE security proofs are still rather complex and non-trivially relate the AKE experiment involving multiple oracles to the much simpler multi-user CDH experiment. Our new assumptions are purely algebraic and do not involve any hash function. Hence, they precisely characterize the "algebraic complexity" of the AKEs' security which certainly improves our understanding of their security. As a matter of fact, as a side result our approach also led to improved security reductions from the standard GapCDH assumption. Furthermore, our new generic bounds are the only known formal argument supporting the security of HMQV in 256-bit groups, cf. Figure 3.

Another point of criticism might be that our new assumptions are non-falsifiable. We remark that the full Gap oracle (i.e., oracle DDH in Figure 4) is the only reason why our new assumptions (such as CorrGapCDH) are non-falsifiable. Previous (non-tight) proofs for HMQV and NAXOS also relied on the non-falsifiable GapCDH, whereas X3DH$^-$ was proved from the weaker and falsifiable Strong CDH assumption, where the first input of the DDH oracle is fixed. For simplicity we decided to analyze all protocols with respect to a gap assumption. But we would like to stress that for NAXOS and X3DH$^-$ we actually do not need the full power of the gap oracle in our proofs (see our comment in the beginning to Section 5). This way we can prove the security of NAXOS and X3DH$^-$ from falsifiable assumptions. Proving HMQV with respect to a falsifiable assumption remains an interesting open problem.

We analyzed the tightness of *existing* AKE protocols of practical relevance. The works [KMP16, BD20, FPS20] took a similar approach in the context of the Schnorr (blind) signature scheme. For example, [KMP16] proved that UF-CMA security of Schnorr signatures in the multi-user setting is tightly implied by the interactive Q_{RO}-IDLOG assumption which in turn has optimal bounds in the GGM. In a different line of work,

new AKE protocols with a tight security reduction from standard assumptions were created from scratch, for example [BHJ+15, CCG+19, JKRS21]. All those schemes are considerably less efficient than NAXOS, X3DH$^-$, and HMQV.

Open Problems. We note that there are several variants of HMQV and NAXOS, such as [Ust08, PW11, Ust09, YZ13]. We are optimistic that our analysis will carry over in a straightforward manner but leave the concrete analysis as an open problem. While we only use our assumptions to analyze two-message DH-based AKE protocols in this paper, we believe that our framework can be extended to analyze the Noise framework [Per17, DRS20] in combination of suitable symmetric primitives. Another interesting open problem is to improve the generic bound for HMQV to t_{OFF}^2/p, or to show an attack matching our slightly worse bound from Figure 3.

2 Preliminaries

Notation. For integers $N, M \in \mathbb{N}^+$, we define $[N, M] := \{N, N+1, \ldots, M\}$ (which is the empty set for $M < N$) and $[N] := [1, N]$. For an adversary $\mathcal{A}$, we write $a \leftarrow \mathcal{A}(b)$ as the output of $\mathcal{A}$ on input b. To express $\mathcal{A}$'s random tape ρ explicitly, we write $a := \mathcal{A}(b; \rho)$. In this case, $\mathcal{A}$'s execution is deterministic. The notation $[\![B]\!]$, where B is a boolean statement, refers to a bit that is 1 if the statement is true and 0 otherwise.

Games. We use code-based games in this paper, following [BR06]. In every game, Boolean values are all initialized to false, numerical values to 0, sets to $\emptyset$, strings to undefined $\perp$. For the empty string, we use a special symbol ϵ. A procedure terminates once it has returned an output.

Idealized Models. In the Generic Group Model (GGM) [Sho97, Mau05], group operations in group $\mathbb{G}$ can only be computed via an oracle Op (Op stands for operation) provided by the GGM, and adversaries only receive unique handles for the corresponding group elements. The GGM internally identifies elements in $\mathbb{G}$ with elements in $\mathbb{Z}_p$, since $(\mathbb{G}, \cdot)$ of order p is isomorphic to $(\mathbb{Z}_p, +)$. Moreover, the GGM maintains an internal list that keeps track of all elements that have been issued. In this paper, our GGM proofs follow the work of Kiltz et al. [KMP16] which essentially uses the Maurer model [Mau05]. In the Random Oracle Model (ROM) [BR93], a hash function is modeled as a perfectly random function. That is, an adversary is only given access to the hash functions via an oracle H which (consistently) outputs uniform random elements in the hash function's range.

The running time of an adversary $\mathcal{A}$ in the GGM and ROM counts the number of calls to the Op and H oracles. We define such calls to the hash and group operation oracles as *offline* queries, since these operations can in practice be performed by an adversary offline, without any interaction with a server. In contrast, we define all queries that require interaction with a server as *online* queries. (For example, queries to a signing oracle in a digital signature scheme.) Adversary $\mathcal{A}$'s offline (or online) running time t_{OFF} (or t_{ON}) is the time $\mathcal{A}$ spends on offline (or online) queries.

BIT SECURITY. According to [CKMS16], a scheme has κ-bit security if $\varepsilon/(t_{\text{ON}}+t_{\text{OFF}}) \leq 2^{-\kappa}$ for all adversaries that run in time $t_{\text{ON}} + t_{\text{OFF}}$ where $1 \leq t_{\text{ON}} + t_{\text{OFF}} \leq 2^{\kappa}$.

3 Multi-User CDH Problems

We formally define our new multi-user CDH problems CorrGapCDH and CorrCRGapCDH, discuss their relation to the standard CDH problem and analyze their generic bounds.

For the rest of this section, we fix parameters $\mathsf{par} = (p, g, \mathbb{G})$ that describe a group $\mathbb{G}$ of prime order $p = |\mathbb{G}|$ and a generator g of $\mathbb{G}$. For $g, A \in \mathbb{G}$, we define $\mathsf{DL}_g(A)$ as the unique $a \in \mathbb{Z}_p$ satisfying $g^a = A$.

STANDARD CDH. We first recall the standard CDH problem which is to compute $g^{a_1 a_2}$ given g^{a_1} and g^{a_2} for randomly chosen $a_1, a_2 \xleftarrow{\$} \mathbb{Z}_p$. A popular variant for proving security of encryption and key exchange protocols is the Gap CDH GapCDH [OP01, ABR01] problem. In GapCDH, the adversary can make queries to a gap oracle $\text{DDH}(A, Y, Z)$ returning the Boolean value $[\![Y^{\mathsf{DL}_g(A)} = Z]\!]$.

MULTI-USER GapCDH. We now consider natural generalizations of GapCDH to a setting with $n \geq 2$ users where the adversary is given the n-tuple $(g^{a_1}, \ldots, g^{a_n})$ and in order to win, it must output any of the $n(n-1)$ possible CDH tuples in the winning set $\mathsf{Win} = \{g^{a_i a_j} \mid i \neq j\}$. Formally, to $n \geq 2$ and $Q_{\text{DDH}} \geq 0$, we associate game $\mathsf{GapCDH}_{n,Q_{\text{DDH}}}$ of Figure 4 and define the advantage function of $\mathcal{A}$ as $\text{Adv}^{\mathsf{GapCDH}}_{n,Q_{\text{DDH}}}(\mathcal{A}) := \Pr[\mathsf{GapCDH}^{\mathcal{A}}_{n,Q_{\text{DDH}}} \Rightarrow 1]$. We let n-GapCDH be the problem with parameters $n \geq 2$ such that GapCDH $= 2$-GapCDH. (To simplify notation we ignore the value Q_{DDH} when naming assumptions.) By a standard re-randomization argument [NR97] over the users, one can show that n-GapCDH is tightly equivalent to GapCDH $= 2$-GapCDH .

MULTI-USER GapCDH WITH CORRUPTION. We now generalize the n-GapCDH problem to allow for user corruptions. Corruptions are modeled by oracle $\text{CORR}_n(i \in [n])$ which returns a_i, the discrete logarithm of $A_i = g^{a_i}$. To win, the adversary must output one of the Diffie-Hellman keys $g^{a_i a_j}$ for two distinct, non-corrupted users i and j. More formally, to $n \geq 2$, and $Q_{\text{DDH}} \geq 0$, we associate game $\mathsf{CorrGapCDH}_{n,Q_{\text{DDH}}}$ of Figure 4 and define the advantage function of $\mathcal{A}$ as $\text{Adv}^{\mathsf{CorrGapCDH}}_{n,Q_{\text{DDH}}}(\mathcal{A}) := \Pr[\mathsf{CorrGapCDH}^{\mathcal{A}}_{n,Q_{\text{DDH}}} \Rightarrow 1]$. We let n-CorrGapCDH be the problem with parameters $n \geq 2$ and Q_{DDH}. We note that due to the corruption oracle a re-randomization argument as for the case without corruptions can no longer be applied and therefore we can not prove tight equivalence between GapCDH and n-CorrGapCDH.

MULTI-USER ASYMMETRIC GapCDH WITH CORRUPTION. This problem is like the n-CorrGapCDH problem, where the corruption oracle $\text{CORR}_{n_1}(i \in [n_1])$ is restricted to users $i \in [n_1]$, where parameter $0 \leq n_1 \leq n$ splits interval $[n]$ in $[n_1]$ and $[n_1 + 1, n]$. To win, the adversary has to return one of the $\leq n_1(n-n_1)$ asymmetric Diffie-Hellman values $A_i^{a_j}$ for non-corrupted users $i \in [n_1]$ and $j \in [n_1 + 1, n]$. More formally, to $n \geq 2$, $0 \leq n_1 \leq n$, and $Q_{\text{DDH}} \geq 0$, we associate game $\mathsf{CorrAGapCDH}_{n,n_1,Q_{\text{DDH}}}$ of Figure 4 and define the advantage function of $\mathcal{A}$ as $\text{Adv}^{\mathsf{CorrAGapCDH}}_{n,n_1,Q_{\text{DDH}}}(\mathcal{A}) := \Pr[\mathsf{CorrAGapCDH}^{\mathcal{A}}_{n,n_1,Q_{\text{DDH}}} \Rightarrow 1]$. We let (n, n_1)-CorrAGapCDH be the problem with parameters $n \geq 2$ and $0 \leq n_1 \leq n$.

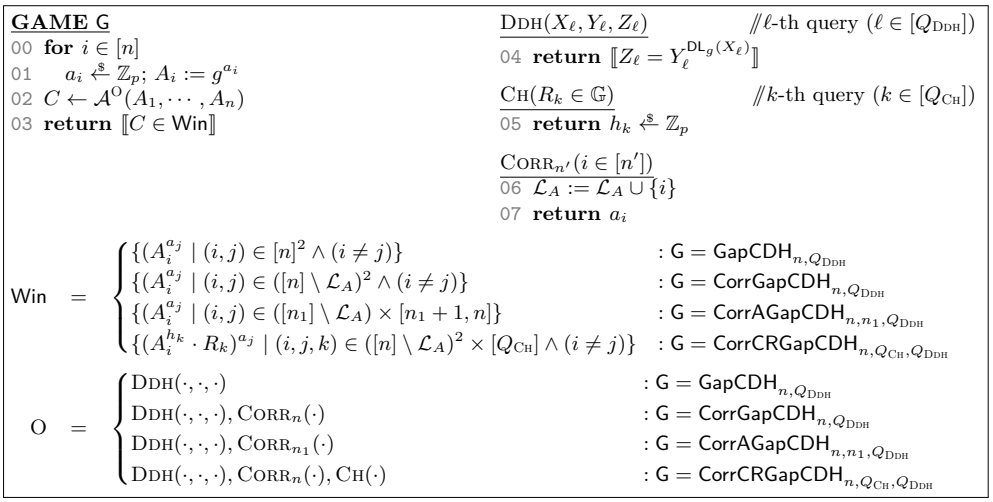

Figure 4: Game $\mathsf{G} \in \{\mathsf{GapCDH}_{n,Q_{\mathrm{DDH}}}, \mathsf{CorrGapCDH}_{n,Q_{\mathrm{DDH}}}, \mathsf{CorrAGapCDH}_{n,n_1,Q_{\mathrm{DDH}}}, \mathsf{CorrCRGapCDH}_{n,Q_{\mathrm{CH}},Q_{\mathrm{DDH}}}\}$ for defining our Multi-User CDH problems.

MULTI-USER CHALLENGE-RESPONSE GapCDH WITH CORRUPTION. Our final problem is a generalization of the n-CorrGapCDH problem. The adversary is given access to a challenge oracle $\mathrm{CH}(R_k \in \mathbb{G})$ ($k \in [Q_{\mathrm{CH}}]$) which returns a response $h_k \xleftarrow{\$} \mathbb{Z}_p$. In the winning condition, the adversary is required to output any of the at most $n(n-1)Q_{\mathrm{CH}}$ elements of the winning set $\mathsf{Win} = \{(A_i^{h_k} \cdot R_k)^{a_j} \mid i \neq j \text{ uncorrupted}\}$. Furthermore, we will give the adversary access to the full gap oracle DDH. More formally, to integers $n \geq 2$, $Q_{\mathrm{CH}} \geq 0$, and $Q_{\mathrm{DDH}} \geq 0$, we associate game $\mathsf{CorrCRGapCDH}_{n,Q_{\mathrm{CH}},Q_{\mathrm{DDH}}}$ of Figure 4 and define the advantage function $\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,Q_{\mathrm{CH}},Q_{\mathrm{DDH}}}(\mathcal{A}) := \Pr[\mathsf{CorrCRGapCDH}^{\mathcal{A}}_{n,Q_{\mathrm{CH}},Q_{\mathrm{DDH}}} \Rightarrow 1]$. We let (n, Q_{CH})-CorrCRGapCDH be the problem with parameters $n \geq 2$ and Q_{CH}.

RELATIONS. Figure 5 summarizes the relations between the multi-user CDH problems. We only state the important ones for our analysis here, all other formal statement and proofs are postponed to Appendix A.

Theorem 1 ($\mathsf{GapCDH} \xrightarrow{\text{non-tightly}} (n, Q_{\mathrm{CH}})$-$\mathsf{CorrCRGapCDH}$)**.** *For any adversary $\mathcal{A}$ against (n, Q_{CH})-CorrCRGapCDH, there exist an adversary $\mathcal{B}$ against GapCDH such that*

$$\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,Q_{\mathrm{CH}},Q_{\mathrm{DDH}}}(\mathcal{A}) \leq Q_{\mathrm{CH}} \cdot n^2 \left(\sqrt{\mathrm{Adv}^{\mathsf{GapCDH}}_{Q_{\mathrm{DDH}}}(\mathcal{B})} + \frac{1}{p} \right), \text{ and } \mathbf{T}(\mathcal{B}) \approx 2\mathbf{T}(\mathcal{A}), \quad (3)$$

where $\mathbf{T}(\mathcal{A})$ and $\mathbf{T}(\mathcal{B})$ are the running times of adversaries $\mathcal{A}$ and $\mathcal{B}$, respectively.

The proof of Theorem 1 and Lemmas 6 to 8 referred to in Figure 5 can be found in Appendix A.1. Proofs of the following lemmas can be found in Appendix A.2.

Lemma 1 ($(n,1)$-CorrCRGapCDH $\to$ n-CorrGapCDH)**.** *For any adversary $\mathcal{A}$ against n-CorrGapCDH, there exists an adversary $\mathcal{B}$ against $(n,1)$-CorrCRGapCDH with*

$$\mathrm{Adv}^{\mathsf{CorrGapCDH}}_{n,Q_{\mathrm{DDH}}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,1,Q_{\mathrm{DDH}}}(\mathcal{B}).$$

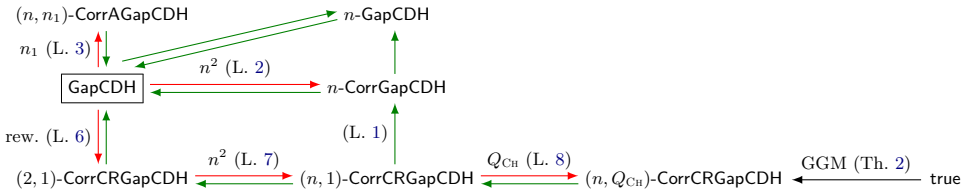

Figure 5: Standard model relations between the standard problem GapCDH (CDH with full gap oracle) and our new problems n-GapCDH, n-CorrGapCDH, and (n, Q_{CH})-CorrCRGapCDH. Red arrows denote non-tight implications with tightness loss as indicated; Green arrows denote tight implications; The black arrow denotes an unconditional statement in the GGM. Formal statements and proofs (unless trivial) are referenced.

Lemma 2 (GapCDH $\xrightarrow{n^2}$ n-CorrGapCDH)**.** *For any adversary $\mathcal{A}$ against n-CorrGapCDH, there exists an adversary $\mathcal{B}$ against* GapCDH *with*

$$\text{Adv}^{\text{CorrGapCDH}}_{n,Q_{\text{DDH}}}(\mathcal{A}) \leq n^2 \cdot \text{Adv}^{\text{GapCDH}}_{Q_{\text{DDH}}}(\mathcal{B}).$$

Lemma 3 (GapCDH $\xrightarrow{n_1}$ (n, n_1)-CorrAGapCDH)**.** *For any adversary $\mathcal{A}$ against (n, n_1)-CorrAGapCDH, there exists an adversary $\mathcal{B}$ against* GapCDH *with*

$$\text{Adv}^{\text{CorrAGapCDH}}_{n,n_1,Q_{\text{DDH}}}(\mathcal{A}) \leq n_1 \cdot \text{Adv}^{\text{GapCDH}}_{Q_{\text{DDH}}}(\mathcal{B}).$$

Theorem 2 (Generic Hardness of CorrCRGapCDH)**.** *For an adversary $\mathcal{A}$ against (n, Q_{CH})-CorrCRGapCDH in the GGM that makes at most Q_{OP} queries to the group oracle* OP*, n' queries to the corruption oracle* CORR*, Q_{DDH} queries to the gap oracle* DDH*, and Q_{CH} queries to the challenge oracle* CH*, $\mathcal{A}$'s advantage is*

$$\text{Adv}^{\text{CorrCRGapCDH}}_{n,Q_{\text{CH}},Q_{\text{DDH}},\text{GGM}}(\mathcal{A}) \leq \frac{(Q_{\text{OP}}+n+1)^2}{2p} + \frac{2Q_{\text{DDH}}}{p} + \frac{(n-n')^2 Q_{\text{CH}}}{2p} + \frac{Q_{\text{CH}}(n-n')}{p} \ .$$

We analyze the hardness of (n, Q_{CH})-CorrCRGapCDH in the generic group model (GGM) [Sho97, Mau05]. In particular, our GGM proofs follow the work of Kiltz et al. [KMP16] which essentially uses the Maurer model [Mau05]. Theorem 2 presents the hardness of (n, Q_{CH})-CorrCRGapCDH in the GGM. Before proving it, we recall a useful lemma.

Lemma 4 (Schwartz–Zippel Lemma)**.** *Let $f(x_1, .., x_n)$ be a non-zero multivariate polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$. Let $\alpha_1, \ldots, \alpha_n$ be chosen uniformly at random from S. Then*

$$\Pr[f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}.$$

Proof (of Theorem 2). We construct a simulator $\mathcal{B}$ who interacts and plays game $\text{CorrCRGapCDH}_{n,Q_{\text{CH}},Q_{\text{DDH}}}$ with $\mathcal{A}$ in the GGM. Group operation, corruption and DDH oracle queries are simulated as in Figure 6.

```
B                                          //simulating in the GGM
00 L_E := {(x_0 := 1, P_{x_0} := 1)}   //set of polynomials
01 for i ∈ [n]
02    α_i <-$ Z_p; L_E := L_E ∪ {(x_i, P_{x_i} := i + 1)}
03 x⃗ := (x_1, ..., x_n)
04 α⃗ := (α_1, ..., α_n)
05 cnt := n + 1                            //size of L_E
06 C ← A^O(P_{x_0}, ..., P_{x_n})
07 if C ∉ [cnt]
08    return 0
09 fetch (z*(x⃗), C) ∈ L_E
10 if ∃(f(x⃗), P), (g(x⃗), P') ∈ L_E
11    and f(x⃗) ≠ g(x⃗) and f(α⃗) = g(α⃗)
12    BAD_G := 1; Abort
13 if z*(α⃗) = (α_{i*} h_k + r_k(α⃗)) α_{j*}
14    if (i*, j*, k) ∈ ([n] \ L_A)^2 × [Q_CH] and i* ≠ j*
15       return 1
16 return 0

DDH(P_i, P_j, P_k)
17 if (P_i, P_j, P_k) ∉ [cnt]^3
18    return ⊥
19 fetch (a(x⃗), P_i), (b(x⃗), P_j), (c(x⃗), P_k) ∈ L_E
20 if c(x⃗) = a(x⃗) · b(x⃗)
21    return 1
22 if c(α⃗) = a(α⃗) · b(α⃗)
23    BAD_DDH := 1; Abort
24 return 0

CORR(i)
25 if i ∉ [n]
26    return ⊥
27 L_A := L_A ∪ {i}
28 return α_i

CH(R_k)          //k-th query (k ∈ [Q_CH])
29 if ∄(r_k(x⃗), R_k) ∈ L_E
30    return ⊥
31 h_k <-$ Z_p
32 return h_k

OP(P, P')
33 if (P, P') ∉ [cnt]^2
34    return ⊥
35 fetch (a(x⃗), P), (b(x⃗), P') ∈ L_E
36 z(x⃗) := a(x⃗) + b(x⃗)
37 if ∃(z(x⃗), P_{z(x⃗)}) ∈ L_E
38    return P_{z(x⃗)}
39 cnt ++
40 P_{z(x⃗)} := cnt
41 L_E := L_E ∪ {(z(x⃗), P_{z(x⃗)})}
42 return P_{z(x⃗)}
```

Figure 6: $\mathcal{B}$ simulates $\mathsf{CorrCRGapCDH}_{n,Q_{\text{CH}},Q_{\text{DDH}}}$ in the Generic Group Model (GGM) and interacts with $\mathcal{A}$. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{DDH}, \text{CORR}, \text{CH}, \text{OP}\}$.

Our overall idea is to simulate the $\mathsf{CorrCRGapCDH}_{n,Q_{\text{CH}},Q_{\text{DDH}}}$ game in a symbolic way using degree-1 polynomials. More precisely, during the simulation our simulator keeps an internal list $\mathcal{L}_E$ with entries of the form $(z(\vec{x}), P_{z(\vec{x})})$ where z is a degree-1 polynomial and $P_{z(\vec{x})} \in \mathbb{N}$ identifies which entry it is. After $\mathcal{A}$ outputs a forgery, our simulator assigns variables $(x_1, \ldots, x_n)$ with $(\alpha_1, \ldots, \alpha_n) \xleftarrow{\$} \mathbb{Z}_p^n$.

Now we note that the simulator perfectly simulates the $\mathsf{CorrCRGapCDH}_{n,Q_{\text{CH}},Q_{\text{DDH}}}$ in the GGM if both BAD_{DDH} and $\text{BAD}_{\mathbb{G}}$ are equal to 0. To bound the probability that one of the bad events happens, we use Lemma 4:

For each DDH query, $\Pr_{\vec{\alpha}}[c(\vec{x}) \neq a(\vec{x}) \cdot b(\vec{x}) \text{ and } c(\vec{\alpha}) = a(\vec{\alpha}) \cdot b(\vec{\alpha})] \leq 2/p$, since $c(\vec{x}) - a(\vec{x}) \cdot b(\vec{x})$ is a non-zero polynomial of degree two. By the union bound, $\Pr[\text{BAD}_{\text{DDH}}] \leq 2Q_{\text{DDH}}/p$, where Q_{DDH} is $\mathcal{A}$'s maximum number of DDH queries.

If $\text{BAD}_{\mathbb{G}}$ happens, there are two distinct degree-1 polynomials $z_i(\vec{x})$ and $z_j(\vec{x})$ in $\mathcal{L}_E$ that collide on input $\vec{\alpha} \xleftarrow{\$} \mathbb{Z}_p^n$. By the union bound, we get

$$\begin{aligned} \Pr[\text{BAD}_{\mathbb{G}}] &:= \Pr_{\vec{\alpha}}[\exists (i,j) \in [\text{cnt}]^2 : z_i(\vec{x}) \neq z_j(\vec{x}) \text{ and } z_i(\vec{\alpha}) = z_j(\vec{\alpha})] \\ &\leq \binom{Q_{\text{OP}} + n + 1}{2} \cdot \frac{1}{p} \leq \frac{(Q_{\text{OP}} + n + 1)^2}{2p}, \end{aligned}$$

where the $1/p$ factor comes from Lemma 4, and the fact that all our polynomials have degree one.

Let $n' := |\mathcal{L}_A|$ be the size of $\mathcal{L}_A$.

The advantage function of $\mathcal{A}$ in the GGM can be bounded as

$$\begin{aligned}
&\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,Q_{\mathrm{CH}},Q_{\mathrm{DDH}},\mathrm{GGM}}(\mathcal{A}) \leq \Pr[\mathrm{BAD}_{\mathbb{G}}] + \Pr[\mathrm{BAD}_{\mathrm{DDH}}] \\
&+ \Pr_{\vec{\alpha}}[\exists (i^*, j^* \neq i^*, k) \in ([n] \setminus \mathcal{L}_A)^2 \times [Q_{\mathrm{CH}}] \colon z^*(\vec{\alpha}) = (\alpha_{i^*} h_k + r_k(\vec{\alpha}))\alpha_{j^*}] \\
&\leq \frac{(Q_{\mathrm{OP}} + n + 1)^2}{2p} + \frac{2Q_{\mathrm{DDH}}}{p} + \frac{(n-n')^2 Q_{\mathrm{CH}}}{2p} + \frac{(n-n')Q_{\mathrm{CH}}}{p}.
\end{aligned}$$

To bound the third probability statement above, we use the following general inequality for events A and B:

$$\Pr[A] = \Pr[A \mid B] \cdot \Pr[B] + \Pr[A \wedge \neg B] \cdot \Pr[\neg B] \leq \Pr[A \mid B] + \Pr[\neg B].$$

This allows us to split the statement into two terms, for which we can apply Lemma 4 to both and get

$$\begin{aligned}
&\Pr_{\vec{\alpha}}[\exists (i^*, j^* \neq i^*, k) \in ([n] \setminus \mathcal{L}_A)^2 \times [Q_{\mathrm{CH}}] \colon z^*(\vec{\alpha}) = (\alpha_{i^*} h_k + r_k(\vec{\alpha}))\alpha_{j^*}] \\
\leq &\Pr_{\vec{\alpha}}[\exists (i^*, j^*, k) \colon z^*(\vec{\alpha}) = (\alpha_{i^*} h_k + r_k(\vec{\alpha}))\alpha_{j^*} \mid \alpha_{i^*} h_k + r_k(\vec{\alpha}) \neq 0] \\
&+ \Pr_{\vec{\alpha}}[\exists (i^*, k) \colon \alpha_{i^*} h_k + r_k(\vec{\alpha}) = 0] \\
\leq &\binom{n-n'}{2} \cdot \binom{Q_{\mathrm{CH}}}{1} \cdot \frac{1}{p} + \binom{n-n'}{1} \cdot \binom{Q_{\mathrm{CH}}}{1} \cdot \frac{1}{p} \\
= &\frac{(n-n')^2 Q_{\mathrm{CH}}}{2p} + \frac{(n-n')Q_{\mathrm{CH}}}{p}.
\end{aligned}$$

□

The following corollary is obtained by applying Lemma 1 to Theorem 2.

Corollary 1 (Generic Hardness of $\mathsf{CorrGapCDH}$)**.** *For an adversary $\mathcal{A}$ against n-$\mathsf{CorrGapCDH}$ in the GGM that makes at most Q_{OP} queries to the group oracle* OP*, n' queries to the corruption oracle* CORR*, and Q_{DDH} queries to the gap oracle* DDH*, $\mathcal{A}$'s advantage is*

$$\mathrm{Adv}^{\mathsf{CorrGapCDH}}_{n,Q_{\mathrm{DDH}},\mathrm{GGM}}(\mathcal{A}) \leq \frac{(Q_{\mathrm{OP}} + n + 1)^2}{2p} + \frac{2Q_{\mathrm{DDH}}}{p} + \frac{(n-n')^2}{2p} + \frac{n-n'}{p} \quad .$$

4 Two-Message Authenticated Key Exchange

A two-message key exchange protocol $\mathsf{AKE} = (\mathsf{Gen_{AKE}}, \mathsf{Init_I}, \mathsf{Init_R}, \mathsf{Der_R}, \mathsf{Der_I})$ consists of five algorithms which are executed interactively by two parties as shown in Figure 7. We denote the party which initiates the session by P_i and the party which responds to the session by P_r. The key generation algorithm $\mathsf{Gen_{AKE}}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ for one party. The initialization algorithms $\mathsf{Init_I}$ and $\mathsf{Init_R}$ input the long-term secret key of

the party running the algorithm and the corresponding peer's long-term public key and output a message I or R and a state st_I or st_R. The derivation algorithms Der_I and Der_R take as input the corresponding long-term secret key, the peer's public key, a message I or R and the state. It computes a session key K. Note that the terms initiator and responder are used to identify the parties, but the notation does not enforce an order of execution. In particular, the protocols we are looking at here allow that messages can be sent simultaneously and both parties may store a state.

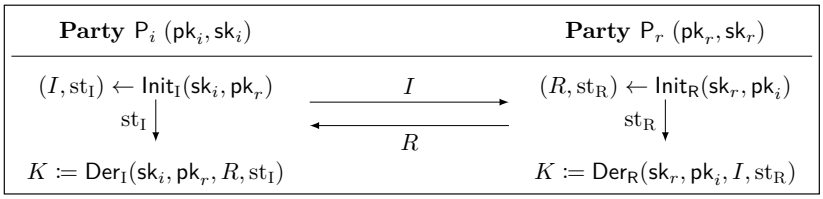

Figure 7: Running a key exchange protocol between two parties.

We give a security game written in pseudocode in the style of [JKRS21]. We define two models for *implicitly authenticated* protocols achieving weak forward secrecy, where one is without and one is with state reveals. The latter models the same security as the eCK model [LLM07], extended by multiple test queries with respect to the same random bit b. The games IND-wFS and IND-wFS-St are given in Figures 8 and 9.

Execution Environment. We consider N parties $\mathsf{P}_1, ..., \mathsf{P}_N$ with long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$, $n \in [N]$. Each session between two parties has a unique identification number sID and variables which are defined relative to sID:

- init[sID] $\in [N]$ denotes the initiator of the session.
- resp[sID] $\in [N]$ denotes the responder of the session.
- type[sID] $\in$ {"In", "Re"} denotes the session's view, i. e., whether the initiator or the responder computes the session key.
- I[sID] denotes the message that was computed by the initiator.
- R[sID] denotes the message that was computed by the responder.
- state[sID] denotes the (secret) state information, i. e., ephemeral secret keys.
- sKey[sID] denotes the session key.

To establish a session between two parties, the adversary is given access to oracles $\textsc{Session}_\mathrm{I}$ and $\textsc{Session}_\mathsf{R}$, where the first one starts a session of type "In" and the second one of type "Re". In order to complete the session, the oracle $\textsc{Der}_\mathrm{I}$ or $\textsc{Der}_\mathsf{R}$ has to be queried. At any time, the adversary can register an *adversarially controlled* party by providing a long-term public key via the oracle RegisterLTK. The adversary does not need to know the corresponding secret key, but the party will be corrupted by definition. Note that oracles $\textsc{Session}_\mathrm{I}$ and $\textsc{Session}_\mathsf{R}$ cannot take an adversarially controlled party as owner. Furthermore, the adversary has access to oracles Corrupt and Reveal to obtain secret information. In game IND-wFS-St, the adversary has additional access to Rev-State. We use the following boolean values to keep track of which queries the adversary made:

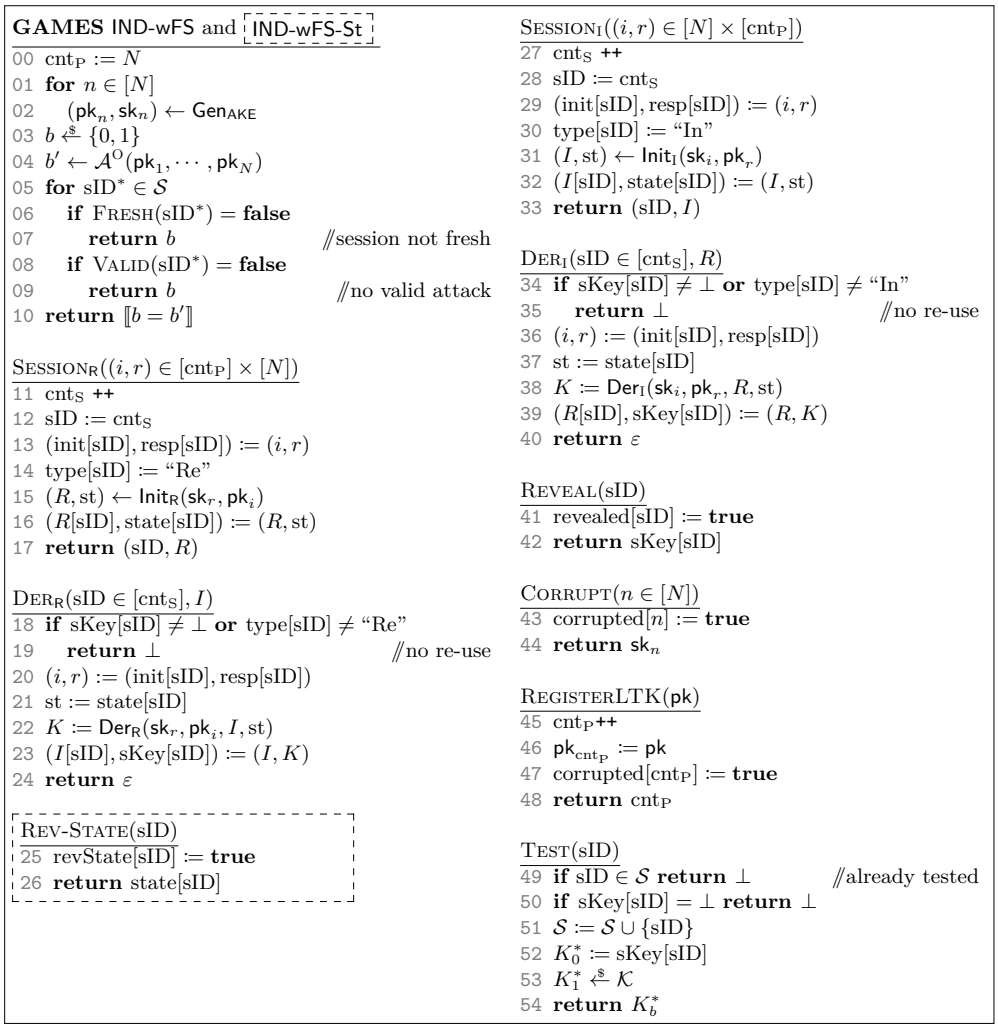

Figure 8: Games IND-wFS and IND-wFS-St for AKE. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_\mathrm{I}, \textsc{Session}_\mathsf{R}, \textsc{Der}_\mathrm{I}, \textsc{Der}_\mathsf{R}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{RegisterLTK}, \textsc{Test}\}$. In game IND-wFS-St, $\mathcal{A}$ has additionally access to oracle Rev-State. Helper procedures Fresh and Valid are defined in Figure 9. If there exists any test session which is not both fresh and valid, the game will return the random bit b.

- corrupted[n] denotes whether the long-term secret key of party P_n was given to the adversary.
- revealed[sID] denotes whether the session key was given to the adversary.
- revState[sID] denotes whether the state information of that session was given to the adversary.

```
FRESH(sID*)
00 (i*, r*) := (init[sID*], resp[sID*])
01 M(sID*) := {sID | (init[sID], resp[sID]) = (i*, r*) ∧ (I[sID], R[sID]) =
                     (I[sID*], R[sID*]) ∧ type[sID] ≠ type[sID*]}          //matching sessions
02 if revealed[sID*] or (∃sID ∈ M(sID*) : revealed[sID] = true)
03   return false                                  //A trivially learned the test session's key
04 if ∃sID ∈ M(sID*) s.t. sID ∈ S
05   return false                                  //A also tested a matching session
06 return true

VALID(sID*)
07 (i*, r*) := (init[sID*], resp[sID*])
08 M(sID*) := {sID | (init[sID], resp[sID]) = (i*, r*) ∧ (I[sID], R[sID]) =
                     (I[sID*], R[sID*]) ∧ type[sID] ≠ type[sID*]}          //matching sessions
09 for attack ∈ Table 2 Table 1
10   if attack = true return true
11 return false
```

Figure 9: Helper procedures FRESH and VALID for games IND-wFS and IND-wFS-St defined in Figure 10. Procedure FRESH checks if the adversary performed some trivial attack. In procedure VALID, each attack is evaluated by the set of variables shown in Table 1 (IND-wFS-St, excluding trivial attacks) or Table 2 (IND-wFS) and checks if an allowed attack was performed. If the values of the variables are set as in the corresponding row, the attack was performed, i. e., attack = **true**, and thus the session is valid.

The adversary can forward messages between sessions or modify them. By that, we can define the relationship between two sessions:

- **Matching Session**: Two sessions sID and sID′ *match* if the same parties are involved (init[sID] = init[sID′] and resp[sID] = resp[sID′]), the messages sent and received are the same (I[sID] = I[sID′] and R[sID] = R[sID′]) and they are of different types (type[sID] ≠ type[sID′]).

As we look at implicitly authenticated protocols that consist only of group elements, they are not vulnerable to no-match attacks described in [LS17].

Finally, the adversary is given access to oracle TEST, which can be queried multiple times and which will return either the session key of the specified session or a uniformly random key. We use one bit b for all test queries. We store test sessions in a set $\mathcal{S}$. In general, the adversary can disclose the complete interaction between two parties by querying the long-term secret keys, the state information and the session key. However, for each test session, we require that the adversary does not issue queries such that the session key can be trivially computed. We define the properties of freshness and validity which all test sessions have to satisfy:

- **Freshness**: A (test) session is called *fresh* if the session key was not revealed. Furthermore, if there exists a matching session, we require that this session's key is not revealed and that this session is not also a test session.
- **Validity**: A (test) session is called *valid* if it is fresh and the adversary performed any attack which is defined in the security model. We capture this with attack tables (cf. Tables 1 and 2).

$\mathcal{A}$ gets (Initiator, Responder)		corrupted[i^*]	corrupted[r^*]	type[sID*]	revState[sID*]	$\exists$sID $\in \mathfrak{M}$(sID*) : revState[sID]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$
0a.	**multiple matching sessions**	–	–	–	–	–	> 1
*0b.	**trivial attack**	–	–	–	–	–	–
1.+2.	**(long-term, long-term)**	–	–	–	**F**	**F**	1
3.+4.	**(state, state)**	**F**	**F**	–	–	–	1
5.	**(long-term, state)**	–	**F**	"In"	**F**	–	1
6.	**(long-term, state)**	–	**F**	"Re"	–	**F**	1
7.	**(state, long-term)**	**F**	–	"In"	–	**F**	1
8.	**(state, long-term)**	**F**	–	"Re"	**F**	–	1
*9.+10.	**(long-term, long-term)**	–	–	–	**F**	n/a	0
11.+12.	**(state, state)**	**F**	**F**	–	–	n/a	0
13.	**(long-term, state)**	–	**F**	"In"	**F**	n/a	0
*14.	**(long-term, state)**	–	**F**	"Re"	–	n/a	0
*15.	**(state, long-term)**	**F**	–	"In"	–	n/a	0
16.	**(state, long-term)**	**F**	–	"Re"	**F**	n/a	0

Table 1: Table of possible attacks for adversaries against implicitly authenticated two-message protocols with ephemeral state reveals. Trivial attacks are highlighted in blue color (and additionally marked with an asterisk *) and thus are NOT valid in our security definition. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. **F** means "false" and "n/a" indicates that there is no state which can be revealed as no matching session exists.

Attack Tables. We define validity of different attack strategies. All attacks are defined using variables to indicate which queries the adversary may (not) make. We consider three dimensions:

- whether the test session is on the initiator's (type[sID*] ="In") or the responder's side (type[sID*] ="Re"),
- all combinations of long-term secret key and state reveals (corrupted and revState variables),
- whether the adversary acted passively (matching session) or actively (no matching session).

This way, we capture all kind of combinations which are possible. From the 16 attacks in total, four are trivial wins for the adversary and thus they are excluded:

- Attack (9.)+(10.): no implicitly authenticated key exchange can achieve full forward security, so that we cannot reveal the long-term keys of both parties when there is no matching session.

	$\mathcal{A}$ gets (Initiator, Responder)	corrupted$[i^*]$	corrupted$[r^*]$	type[sID*]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$
0a.	**multiple matching sessions**	–	–	–	> 1
1.+2.	**(long-term, long-term)**	–	–	–	1
13.	**(long-term, –)**	–	**F**	"In"	0
16.	**(–, long-term)**	**F**	–	"Re"	0

Table 2: Distilled table of attacks for adversaries against implicitly authenticated two-message protocols without ephemeral state reveals. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value and **F** means "false".

– Attack (14.)+(15.): an adversary cannot reveal the long-term secret key of the test session's peer when there is no matching session, otherwise it can simply impersonate the party.

Instead of black-listing these trivial attacks, our model captures what the adversary is allowed to do. Hence, all non-trivial attacks are covered in our model, in particular capturing *weak forward secrecy* (wFS), *key compromise impersonation* (KCI) and *maximal exposure* (MEX) attacks. In more detail, wFS covers passive adversaries that are allowed to corrupt both parties' long-term keys after the session is completed (1.+2.). KCI covers adversaries that will try to impersonate an honest party to a corrupted party (13., 16.). MEX covers adversaries that have revealed any pair of long-term secret key and state, except for both the long-term key and state of one party (5.-8., 11.+12.).

An attack is performed if the variables are set to the corresponding values in the table. Table 1 is used for the IND-wFS-St security game, excluding trivial attacks highlighted in blue. For completeness, we add the trivial attack in row (0b.), where an adversary may query all secret information of a session. When not considering states, most of the attacks are redundant. This way, we obtain the *distilled* table for the IND-wFS security game given in Table 2.

However, if the protocol does not use appropriate randomness, it should not be considered secure. Thus, if the adversary is able to create more than one matching session to a test session, he may also run a trivial attack. We model this in row (0a.) of Tables 1 and 2.

How to read the tables. As an example, we choose row (1.+2.) of Table 1. Then, if the test session is an initiating session, the state was not revealed (revState[sID*] = **false**) and there is a matching session ($\lvert\mathfrak{M}(\text{sID}^*)\rvert = 1$) whose state was also not revealed, this row will evaluate to true. In this scenario, the adversary is allowed to query both long-term secret keys.

For all test sessions, at least one attack has to evaluate to true. If not, the game will return a random bit. The adversary wins if he does not make a trivial attack and distinguishes the session keys from uniformly random keys which he obtains through queries to the TEST oracle.

When proving the security of a protocol, the success probability for each attack strategy listed in the corresponding table will have to be analyzed, thus showing that independently of which queries the adversary makes, he cannot distinguish the session key from a uniformly random key.

In the protocols we look at, the state is defined as the ephemeral secret key (e.g., the exponent of a group element) and thus equivalent with the randomness which is used to compute the first message. Thus IND-wFS-St is exactly the same level of security as captured by the eCK model, extended by multiple test queries to the same random bit b.

Definition 1 (Key Indistinguishability of AKE)**.** *We define games* IND-wFS *and* IND-wFS-St *as in Figures 8 and 9. The advantage of an adversary $\mathcal{A}$ against* AKE *in these games is defined as*

$$\mathrm{Adv}^{\mathsf{IND\text{-}wFS}}_{\mathsf{AKE}}(\mathcal{A}) := \left|2\Pr[\mathsf{IND\text{-}wFS}^{\mathcal{A}} \Rightarrow 1] - 1\right| \quad \textit{and}$$
$$\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}}(\mathcal{A}) := \left|2\Pr[\mathsf{IND\text{-}wFS\text{-}St}^{\mathcal{A}} \Rightarrow 1] - 1\right| \quad .$$

5 Protocols X3DH$^-$ and NAXOS

In this section, we want to analyze the X3DH$^-$ and NAXOS protocols (see Figure 1 in the introduction). The protocols are defined relative to fixed parameters $(p, g, \mathbb{G})$ that describe a group $\mathbb{G}$ of prime order $p = |\mathbb{G}|$ and a generator g of $\mathbb{G}$. G and H are hash functions with $\mathsf{G} : \{0,1\}^\lambda \times \mathbb{Z}_p \to \mathbb{Z}_p$ and $\mathsf{H} : \mathbb{G}^7 \to \{0,1\}^\lambda$, where $\lambda \geq \log(p)$.

We note that the original proof by Cohn-Gordon et al. [CCG+19] for X3DH$^-$ is based on the strong Diffie Hellman Assumption, where the first input of the DDH oracle is fixed. Our proof strategy does not allow for that as we handle multiple attacks at a time and avoid guessing. However, we want to stress that we do not require the full power of the gap oracle, but could restrict ourselves to queries to DDH, where the first value is one of the input elements of the corresponding multi-user CDH problem. The same applies to the proof of NAXOS.

Also note that X3DH$^-$ is insecure under ephemeral key reveals, so we prove security in a weaker model as done in the original proof by [CCG+19].

Theorem 3 ($(N+S, N)$-CorrAGapCDH + S-GapCDH $\xrightarrow{\text{tight, ROM}}$ X3DH$^-$ IND-wFS)**.** *For any* IND-wFS *adversary $\mathcal{A}$ against* X3DH$^-$ *with N parties that establishes at most S sessions and issues at most T queries to the* TEST *oracle and at most Q_H queries to the random oracle* H*, there exist an adversary $\mathcal{B}$ against $(N+S, N)$-CorrAGapCDH and an adversary $\mathcal{C}$ against S-GapCDH with running times $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{C})$ such that*

$$\mathrm{Adv}^{\mathsf{IND\text{-}wFS}}_{\mathsf{X3DH}^-}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{CorrAGapCDH}}_{N+S,\, N,\, 3Q_\mathsf{H}}(\mathcal{B}) + \mathrm{Adv}^{\mathsf{GapCDH}}_{S,\, Q_\mathsf{H}}(\mathcal{C}) + \frac{(N+S)^2}{p} \quad .$$

The proof is given in Appendix B.

Theorem 4 ($(N+S)$-CorrGapCDH $\xrightarrow{\text{tight, ROM}}$ NAXOS IND-wFS-St)**.** *For any* IND-wFS-St *adversary* $\mathcal{A}$ *against* NAXOS *with* N *parties that establishes at most* S *sessions and issues at most* T *queries to the* Test *oracle, at most* Q_{G} *queries to random oracle* G *and at most* Q_{H} *queries to random oracle* H*, there exists an adversary* $\mathcal{B}$ *against* $(N+S)$-CorrGapCDH *with running time* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *such that*

$$\mathrm{Adv}^{\text{IND-wFS-St}}_{\text{NAXOS}}(\mathcal{A}) \leq \mathrm{Adv}^{\text{CorrGapCDH}}_{N+S,\,3Q_{\mathsf{H}}}(\mathcal{B}) + \frac{(N+S)^2}{p} + \frac{S^2}{p} + \frac{2Q_{\mathsf{G}}S}{p} \ .$$

Proof. Let $\mathcal{A}$ be an adversary against IND-wFS-St security of NAXOS, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is the maximum number of test sessions. Consider the sequence of games in Figure 10.

Game G_0. This is the original IND-wFS-St game. In this game, we implicitly assume that all long-term keys, all messages output by $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathsf{R}}$, and all ephemeral secret keys are different. If such a collision happens, the game will abort. Using the birthday paradox, the probability for that can be upper bounded by $(N+S)^2/(2p)$ for N long-term key pairs and at most S messages, where exponents are chosen uniformly at random from $\mathbb{Z}_p$, and $S^2/(2p)$ for ephemeral secret keys esk, which are chosen uniformly at random from $\{0,1\}^\lambda$ and $\lambda \geq \log(p)$. This rules out attack (0a.), as there will be no two sessions having the same transcript. We get

$$\Pr[\text{IND-wFS-St}^{\mathcal{A}} \Rightarrow 1] \leq \Pr[G_0^{\mathcal{A}} \Rightarrow 1] + \frac{(N+S)^2}{2p} + \frac{S^2}{2p} \ . \tag{4}$$

Game G_1. In game G_1, we define event $\mathrm{BAD}_{\textsc{State}}$ which occurs if the adversary makes a query to random oracle G on a string $esk \in \{0,1\}^\lambda$ which was used in any session, but was not revealed to the adversary yet (line 53). This will become important in the next game hop since we need to be able to reprogram G in case there is a Rev-State query and Corrupt query for the party involved. If $\mathrm{BAD}_{\textsc{State}}$ happens, the game aborts. The probability for this event to happen can be upper bounded by the number of oracle queries and the number of sessions:

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathrm{BAD}_{\textsc{State}}] \leq \frac{Q_{\mathsf{G}} \cdot S}{p} \ .$$

Game G_2. In game G_2, the challenge oracle Test always outputs a uniformly random key, independent from the bit b (line 31). We use that

$$\begin{aligned}
\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| &= \frac{1}{2}\Big|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b=0] + \Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b=1] \\
&\quad - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b=0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b=1]\Big| \\
&= \frac{1}{2}\Big|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b=0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b=0]\Big| \ , \qquad (5)
\end{aligned}$$

where the last equation holds because $\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b=1] = \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b=1]$.

GAMES G_0, $\boxed{G_1}$, $\boxed{G_2}$

00 $\text{cnt}_\text{P} \coloneqq N$
01 **for** $n \in [N]$
02 $\quad a_n \xleftarrow{\$} \mathbb{Z}_p$; $A_n \coloneqq g^{a_n}$
03 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) \coloneqq (A_n, a_n)$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
06 **for** $\text{sID}^* \in \mathcal{S}$
07 $\quad$ **if** $\textsc{Fresh}(\text{sID}^*) = \textbf{false}$ **return** b
08 $\quad$ **if** $\textsc{Valid}(\text{sID}^*) = \textbf{false}$ **return** b
09 **return** $[\![b = b']\!]$

$\textsc{Session}_\text{R}((i,r) \in [\text{cnt}_\text{P}] \times [N])$
10 cnt_S ++
11 $\text{sID} \coloneqq \text{cnt}_\text{S}$
12 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) \coloneqq (i,r)$
13 $\text{type}[\text{sID}] \coloneqq$ "Re"
14 $esk_r \xleftarrow{\$} \{0,1\}^\lambda$
15 $y \coloneqq \mathsf{G}(esk_r, a_r)$; $Y \coloneqq g^y$
16 $(R[\text{sID}], \text{state}[\text{sID}]) \coloneqq (Y, esk_r)$
17 **return** (sID, Y)

$\textsc{Der}_\text{R}(\text{sID} \in [\text{cnt}_\text{S}], X)$
18 **if** $\text{sKey}[\text{sID}] \neq \perp$ **or** $\text{type}[\text{sID}] \neq$ "Re"
19 $\quad$ **return** $\perp$
20 $(i,r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
21 $(Y, esk_r) \coloneqq (R[\text{sID}], \text{state}[\text{sID}])$
22 $y \coloneqq \mathsf{G}(esk_r, a_r)$
23 $\text{ctxt} \coloneqq (A_i, A_r, X, Y)$
24 $K \coloneqq \mathsf{H}(\text{ctxt}, A_i^y, X^{a_r}, X^y)$
25 $(I[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (X, K)$
26 **return** ε

$\textsc{Test}(\text{sID})$
27 **if** $\text{sID} \in \mathcal{S}$ **return** $\perp$
28 **if** $\text{sKey}[\text{sID}] = \perp$ **return** $\perp$
29 $\mathcal{S} \coloneqq \mathcal{S} \cup \{\text{sID}\}$
30 $K_0^* \coloneqq \text{sKey}[\text{sID}]$
$\boxed{31\ K_0^* \xleftarrow{\$} \mathcal{K}}$
32 $K_1^* \xleftarrow{\$} \mathcal{K}$
33 **return** K_b^*

$\textsc{Session}_\text{I}((i,r) \in [N] \times [\text{cnt}_\text{P}])$
34 cnt_S ++
35 $\text{sID} \coloneqq \text{cnt}_\text{S}$
36 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) \coloneqq (i,r)$
37 $\text{type}[\text{sID}] \coloneqq$ "In"
38 $esk_i \xleftarrow{\$} \{0,1\}^\lambda$
39 $x \coloneqq \mathsf{G}(esk_i, a_i)$; $X \coloneqq g^x$
40 $(I[\text{sID}], \text{state}[\text{sID}]) \coloneqq (X, esk_i)$
41 **return** (sID, X)

$\textsc{Der}_\text{I}(\text{sID} \in [\text{cnt}_\text{S}], Y)$
42 **if** $\text{sKey}[\text{sID}] \neq \perp$ **or** $\text{type}[\text{sID}] \neq$ "In"
43 $\quad$ **return** $\perp$
44 $(i,r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
45 $(X, esk_i) \coloneqq (I[\text{sID}], \text{state}[\text{sID}])$
46 $x \coloneqq \mathsf{G}(esk_i, a_i)$
47 $\text{ctxt} \coloneqq (A_i, A_r, X, Y)$
48 $K \coloneqq \mathsf{H}(\text{ctxt}, Y^{a_i}, A_r^x, Y^x)$
49 $(R[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (Y, K)$
50 **return** ε

$\mathsf{G}(esk, a)$
51 **if** $\mathsf{G}[esk, a] = z$
52 $\quad$ **return** z
53 **else if** $\exists \text{sID}$ s. t. $esk = \text{st}[\text{sID}]$
$\quad$ **and** $\text{revState}[\text{sID}] = \textbf{false}$
54 $\quad \text{BAD}_\text{STATE} \coloneqq \textbf{true}$
55 $\quad$ **abort**
56 **else**
57 $\quad z \xleftarrow{\$} \mathbb{Z}_p$
58 $\quad \mathsf{G}[esk, a] \coloneqq z$
59 $\quad$ **return** z

$\mathsf{H}(A_i, A_r, X, Y, Z_1, Z_2, Z_3)$
60 **if** $\mathsf{H}[A_i, A_r, X, Y, Z_1, Z_2, Z_3] = K$
61 $\quad$ **return** K
62 **else**
63 $\quad K \xleftarrow{\$} \mathcal{K}$
64 $\quad \mathsf{H}[A_i, A_r, X, Y, Z_1, Z_2, Z_3] \coloneqq K$
65 $\quad$ **return** K

Figure 10: Games G_0-G_2 for the proof of Theorem 4. $\mathcal{A}$ has access to oracles $\text{O} \coloneqq \{\textsc{Session}_\text{I}, \textsc{Session}_\text{R}, \textsc{Der}_\text{I}, \textsc{Der}_\text{R}, \textsc{Rev-State}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{RegisterLTK}, \textsc{Test}, \mathsf{G}, \mathsf{H}\}$, where $\textsc{RegisterLTK}$, $\textsc{Corrupt}$, $\textsc{Rev-State}$ and $\textsc{Reveal}$ are defined as in the original $\mathsf{IND\text{-}wFS\text{-}St}$ game (Fig. 8). G_0 implicitly assumes that no long-term keys or messages generated by the experiment collide.

Due to the exclusion of collisions, a particular (test) session cannot be recreated, i.e., the adversary cannot create two sessions sID, sID' of the same type that compute the same session key. Thus, the adversary must query the random oracle H on the correct input to distinguish a session key from a random key. We construct adversary $\mathcal{B}$ against

$(N+S)$-CorrGapCDH in Figures 11 and 12 to interpolate between the two games. We now describe adversary $\mathcal{B}$ in detail.

$\mathcal{B}$ gets as input $(N+S)$ group elements and has access to oracles Corr and Ddh. The first N group elements $(A_1, ..., A_N)$ are used as public keys for the parties $\mathsf{P}_1, ..., \mathsf{P}_N$ (line 02). The remaining group elements $(B_1, ..., B_S)$ will be used as outputs for Session_I and Session_R. This means that whenever $\mathcal{A}$ initiates a session sID, $\mathcal{B}$ increments the session counter and chooses the secret random string esk. Instead of evaluating G, it outputs the group element B_{sID} (lines 22, 14). Note that as long as esk is unknown to $\mathcal{A}$, this is a perfect simulation.

To identify queries to the random oracle with correct Diffie-Hellman tuples, $\mathcal{B}$ uses a flag f which is added as additional entry in the list of queries to H. This helps to reduce the number of Ddh queries in oracles Der_I or Der_R. In particular, whenever $\mathcal{A}$ calls one of the two oracles, $\mathcal{B}$ first checks the list of queries to H (lines 58, 41) and if there is an entry with $f = 1$, it outputs the corresponding session key. If this is not the case, it checks if there is an entry with unknown Diffie-Hellman tuples (lines 60, 43). This is to keep session keys of matching sessions consistent. If there is no such entry, $\mathcal{B}$ chooses a session key uniformly at random (lines 63, 46) and adds an entry with unknown Diffie-Hellman tuples to the list. If $\mathcal{A}$ issues a query to H which has not been asked before, $\mathcal{B}$ checks if the Diffie-Hellman tuples are correct using the Ddh oracle (Fig. 12, line 02). In this case, it sets the flag f to 1. Furthermore, if there is an entry with unknown values, it updates the entry (line 05) and outputs the corresponding key. Otherwise, f is set to 0. $\mathcal{B}$ chooses a key uniformly at random (line 09), adds an entry with f to the list and outputs the key.

We now describe how we patch random oracle G. As soon as the adversary has queried both Rev-State and Corrupt for the owner of the session (i.e., the initiator in a session of type "In" or the responder in a session of type "Re"), then it can query G on the respective inputs. Thus, we fix the output value of G at exactly that time, i.e., on a corrupt query (after a state reveal query) as well as on a state reveal query (after a corrupt query).

That is, whenever $\mathcal{A}$ calls Rev-State on sID, $\mathcal{B}$ checks if the owner of the session is corrupted (Fig. 11, lines 27, 30). If this is the case, we have to patch the random oracle G by querying the Corr oracle on B_{sID} which is the message output by this session (lines 28, 31). Note that the corresponding input has not been queried to G before because then event $\text{BAD}_{\text{State}}$ would have occurred.

Further, whenever $\mathcal{A}$ corrupts a party P_n, $\mathcal{B}$ queries the Corr oracle on n (line 69). We then have to patch G for all sessions where P_n is the owner and the state of that session was revealed (line 71). Note that the corresponding input has not been queried to G before because then $\mathcal{B}$ would have already aborted.

If $\mathcal{A}$ makes a query to G, where the input a equals the secret key of any user which was not corrupted before (Fig. 12, line 17), i.e., $g^a = A_n$ for some $n \in [N]$, then $\mathcal{B}$ is able to compute a solution for the CorrGapCDH problem. It just looks for some $A_{n'}$ such that n' was not queried to Corrupt or B_{sID} such that b_{sID} has not been revealed via a Corr query. Then it can output $C = (A_{n'})^a$ or $C = (B_{\text{sID}})^a$ as valid solution. Note that such an $A_{n'}$ or B_{sID} must exist. Note also that in this case, the adversary $\mathcal{A}$ can trivially compute the session key for a valid test session.

$\mathcal{B}^{\text{Corr,Ddh}}(A_1, ..., A_N, B_1, ..., B_S)$
00 $\text{cnt}_\text{P} := N$
01 **for** $n \in [N]$
02 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) := (A_n, \bot)$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
05 **for** $\text{sID}^* \in \mathcal{S}$
06 $\quad$ **if** $\text{Fresh}(\text{sID}^*) = \textbf{false}$ **return** b
07 $\quad$ **if** $\text{Valid}(\text{sID}^*) = \textbf{false}$ **return** b
08 **return** $C \in \mathsf{Win}$ (see text)

$\text{Session}_\text{I}((i,r) \in [N] \times [\text{cnt}_\text{P}])$
09 cnt_S ++
10 $\text{sID} := \text{cnt}_\text{S}$
11 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i, r)$
12 $\text{type}[\text{sID}] := \text{“In”}$
13 $esk_i \xleftarrow{\$} \{0,1\}^\lambda$
14 $X := B_{\text{sID}}$
15 $(I[\text{sID}], \text{state}[\text{sID}]) := (X, esk_i)$
16 **return** (sID, X)

$\text{Session}_\text{R}((i,r) \in [\text{cnt}_\text{P}] \times [N])$
17 cnt_S ++
18 $\text{sID} := \text{cnt}_\text{S}$
19 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i, r)$
20 $\text{type}[\text{sID}] := \text{“Re”}$
21 $esk_r \xleftarrow{\$} \{0,1\}^\lambda$
22 $Y := B_{\text{sID}}$
23 $(R[\text{sID}], \text{state}[\text{sID}]) := (Y, esk_r)$
24 **return** (sID, Y)

$\text{Rev-State}(\text{sID})$
25 $\text{revState}[\text{sID}] := \textbf{true}$
26 $(i,r) := (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
27 **if** $\text{type}[\text{sID}] = \text{“In”}$ **and** $\text{corrupted}[i]$
28 $\quad b_{\text{sID}} := \text{Corr}(N + \text{sID})$
29 $\quad \mathsf{G}[\text{state}[\text{sID}], a_i] := b_{\text{sID}}$
30 **else if** $\text{type}[\text{sID}] = \text{“Re”}$ **and** $\text{corrupted}[r]$
31 $\quad b_{\text{sID}} := \text{Corr}(N + \text{sID})$
32 $\quad \mathsf{G}[\text{state}[\text{sID}], a_r] := b_{\text{sID}}$
33 **return** $\text{state}[\text{sID}]$

$\text{Der}_\text{I}(\text{sID} \in [\text{cnt}_\text{S}], Y)$
34 **if** $\text{sKey}[\text{sID}] \neq \bot$ **or** $\text{type}[\text{sID}] \neq \text{“In”}$
35 $\quad$ **return** $\bot$
36 $(i,r) := (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
37 $X := I[\text{sID}]$
38 $\text{ctxt} := (A_i, A_r, X, Y)$
39 **if** $\exists \text{sID}'$ s. t. $(\text{type}[\text{sID}'], R[\text{sID}']) = (\text{“Re”}, Y)$
40 $\quad \mathcal{P} := \mathcal{P} \cup \{\text{sID}\}$
41 **if** $\exists Z_1, Z_2, Z_3$ s. t. $\mathsf{H}[\text{ctxt}, Z_1, Z_2, Z_3, 1] = K$
42 $\quad \text{sKey}[\text{sID}] := K$
43 **else if** $\mathsf{H}[\text{ctxt}, \bot, \bot, \bot, \bot] = K$
44 $\quad \text{sKey}[\text{sID}] := K$
45 **else**
46 $\quad K \xleftarrow{\$} \mathcal{K}$
47 $\quad \mathsf{H}[\text{ctxt}, \bot, \bot, \bot, \bot] := K$
48 $\quad \text{sKey}[\text{sID}] := K$
49 $(R[\text{sID}], \text{sKey}[\text{sID}]) := (Y, K)$
50 **return** ε

$\text{Der}_\text{R}(\text{sID} \in [\text{cnt}_\text{S}], X)$
51 **if** $\text{sKey}[\text{sID}] \neq \bot$ **or** $\text{type}[\text{sID}] \neq \text{“Re”}$
52 $\quad$ **return** $\bot$
53 $(i,r) := (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
54 $Y := R[\text{sID}]$
55 $\text{ctxt} := (A_i, A_r, X, Y)$
56 **if** $\exists \text{sID}'$ s. t. $(\text{type}[\text{sID}'], I[\text{sID}']) = (\text{“In”}, X)$
57 $\quad \mathcal{P} := \mathcal{P} \cup \{\text{sID}\}$
58 **if** $\exists Z_1, Z_2, Z_3$ s. t. $\mathsf{H}[\text{ctxt}, Z_1, Z_2, Z_3, 1] = K$
59 $\quad \text{sKey}[\text{sID}] := K$
60 **else if** $\mathsf{H}[\text{ctxt}, \bot, \bot, \bot, \bot] = K$
61 $\quad \text{sKey}[\text{sID}] := K$
62 **else**
63 $\quad K \xleftarrow{\$} \mathcal{K}$
64 $\quad \mathsf{H}[\text{ctxt}, \bot, \bot, \bot, \bot] := K$
65 $\quad \text{sKey}[\text{sID}] := K$
66 $(I[\text{sID}], \text{sKey}[\text{sID}]) := (X, K)$
67 **return** ε

$\text{Corrupt}(n \in [N])$
68 $\text{corrupted}[n] := \textbf{true}$
69 $a_n \leftarrow \text{Corr}(n)$
70 $\mathsf{sk}_n := a_n$
71 $\forall \text{sID}$ with $((\text{init}[\text{sID}], \text{type}[\text{sID}]) = (n, \text{“In”})$
$\quad$ **or** $(\text{resp}[\text{sID}], \text{type}[\text{sID}]) = (n, \text{“Re”}))$
$\quad$ **and** $\text{revState}[\text{sID}]$
72 $\quad b_{\text{sID}} \leftarrow \text{Corr}(N + \text{sID})$
73 $\quad \mathsf{G}[\text{state}[\text{sID}], a_n] := b_{\text{sID}}$
74 **return** sk_n

Figure 11: Adversary $\mathcal{B}$ against $(N + S)$-CorrGapCDH for the proof of Theorem 4. $\mathcal{A}$ has access to oracles $\text{O} := \{\text{Session}_\text{I}, \text{Session}_\text{R}, \text{Der}_\text{I}, \text{Der}_\text{R}, \text{Rev-State}, \text{Reveal}, \text{Corrupt}, \text{RegisterLTK}, \text{Test}, \mathsf{G}, \mathsf{H}\}$, where RegisterLTK, Reveal and Test are defined as in game G_2 of Figure 10. Oracles H and G are defined in Figure 12. Lines written in blue color highlight how $\mathcal{B}$ simulates G_1 and G_2, respectively.

```
H(A_i, A_r, X, Y, Z_1, Z_2, Z_3)
00 if H[A_i, A_r, X, Y, Z_1, Z_2, Z_3, · ] = K
01    return K
02 if DDH(A_i, Y, Z_1) = 1
     and DDH(A_r, X, Z_2) = 1
     and DDH(X, Y, Z_3) = 1
03    f := 1
04    if H[A_i, A_r, X, Y, ⊥, ⊥, ⊥, ⊥] = K
05       replace (⊥, ⊥, ⊥, ⊥) with (Z_1, Z_2, Z_3, f)
06       return K
07 else
08    f := 0
09 K ←$ 𝒦
10 H[A_i, A_r, X, Y, Z_1, Z_2, Z_3, f] := K
11 return K
```

```
G(esk, a)
12 if G[esk, a] = z
13    return z
14 else if ∃sID s. t. esk = st[sID]
     and revState[sID] = false
15    BAD_STATE := true
16    abort
17 else if ∃n ∈ [N] s. t. A_n = g^a
     and corrupted[n] = false
18    abort and return C ∈ Win (see text)
19 else
20    z ←$ ℤ_p
21    G[esk, a] := z
22    return z
```

Figure 12: Oracles H and G for adversary $\mathcal{B}$ in Figure 11.

We now show that if $\mathcal{A}$ queries to the random oracle on the correct input for at least one test session, $\mathcal{B}$ is able to output a solution $C \in \mathsf{Win}$ to the CorrGapCDH problem. Let $\mathrm{sID}^* \in \mathcal{S}$ be any test session and $\mathsf{H}[A_{i^*}, A_{r^*}, X^*, Y^*, Z_1^*, Z_2^*, Z_3^*, 1] = \mathrm{sKey}[\mathrm{sID}^*]$ be the corresponding entry in the list of hash queries. $\mathcal{B}$ has to find this query in the list and depending on which reveal queries $\mathcal{A}$ has made (i.e., which attack was performed), $\mathcal{B}$ returns either Z_1^*, Z_2^* or Z_3^* as described below. Therefore, we will now argue that for each possible attack listed in Table 1, there will be a correct solution for CorrGapCDH.

ATTACK (1.)+(2.). There is a matching session sID' and $\mathcal{A}$ has queried both long-term secret keys a_{i^*} and a_{r^*}. $\mathcal{A}$ is not allowed to query the state of those sessions. W.l.o.g. assume the test session is of type "Re". Then, messages X^* and Y^* are chosen by the reduction $\mathcal{B}$ as $B_{\mathrm{sID}'}$ and B_{sID^*}. Thus, in order to distinguish the session key, $\mathcal{A}$ has to compute $Z_3^* = \mathsf{DH}(X^*, Y^*) = \mathsf{DH}(B_{\mathrm{sID}'}, B_{\mathrm{sID}^*})$.

ATTACK (3.)+(4.). There is a matching session sID' and $\mathcal{A}$ has queried both states esk_{i^*} and esk_{r^*}. $\mathcal{A}$ is not allowed to query the long-term secret keys of both parties. Again, we assume that the test session is of type "Re" (w.l.o.g). The states do not reveal any information about the exponents of X^* and Y^* (i.e., $B_{\mathrm{sID}'}$ and B_{sID^*}), as $\mathcal{A}$ has not made a query to G specifying the correct long-term secret key. Also note that $\mathcal{B}$ never queried the CORR oracle to reveal the exponents of $B_{\mathrm{sID}'}$ and B_{sID^*} or A_{i^*} and A_{r^*}. Thus, in order to distinguish the session key, $\mathcal{A}$ has to compute all of the Diffie-Hellman tuples $Z_1^* = \mathsf{DH}(A_{i^*}, B_{\mathrm{sID}^*})$, $Z_2^* = \mathsf{DH}(A_{r^*}, B_{\mathrm{sID}'})$ and $Z_3^* = \mathsf{DH}(B_{\mathrm{sID}^*}, B_{\mathrm{sID}'})$.

ATTACK (5.)+(6.). There is a matching session sID' and $\mathcal{A}$ has queried the initiator's long-term secret key a_{i^*} and the responder's state esk_{r^*}, but neither the responder's long-term secret key a_{r^*} nor the initiator's state esk_{i^*}. Again, assume the test session is of type "Re" (w.l.o.g.). Message X^* is chosen as $B_{\mathrm{sID}'}$. In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_2^* = \mathsf{DH}(A_{r^*}, X^*) = \mathsf{DH}(A_{r^*}, B_{\mathrm{sID}'})$.

ATTACK (7.)+(8.). This is the same as the case before, only that the adversary queried the other party's long-term key or state. Message Y^* is chosen as B_{sID^*} and in order to distinguish the session key, $\mathcal{A}$ has to compute $Z_1^* = \mathsf{DH}(A_{i^*}, Y^*) = \mathsf{DH}(A_{i^*}, B_{\mathrm{sID}^*})$.

ATTACK (11.). The test session is of type "In" and there is no matching session. $\mathcal{A}$ has queried the initiator's state esk_{i^*}. Message X^* is chosen as B_{sID^*}, whereby Y^* is chosen by $\mathcal{A}$. The state does not reveal any information about the exponent of X^* (B_{sID^*}) as $\mathcal{A}$ has not made a query to G on (esk_{i^*}, a_{i^*}). In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_2^* = \mathsf{DH}(A_{r^*}, B_{\mathrm{sID}^*})$.

ATTACK (12.). The test session is of type "Re" and there is no matching session. $\mathcal{A}$ has queried the responder's state esk_{r^*}. Message Y^* is chosen as B_{sID^*}, whereby X^* is chosen by $\mathcal{A}$. The state does not reveal any information about the exponent of Y^* (B_{sID^*}) as $\mathcal{A}$ has not made a query to G on (esk_{r^*}, a_{r^*}). In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_1^* = \mathsf{DH}(A_{i^*}, B_{\mathrm{sID}^*})$.

ATTACK (13.). The test session is of type "In" and there is no matching session. $\mathcal{A}$ has queried the initiator's long-term secret keys a_{i^*}. Message X^* is chosen by the reduction $\mathcal{B}$ as B_{sID^*}, whereby Y^* is chosen by $\mathcal{A}$. In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_2^* = \mathsf{DH}(A_{r^*}, X^*) = \mathsf{DH}(A_{r^*}, B_{\mathrm{sID}^*})$.

ATTACK (16.). The test session is of type "Re" and there is no matching session. $\mathcal{A}$ has queried the responder's long-term secret keys a_{r^*}. Message Y^* is chosen by the reduction $\mathcal{B}$ as B_{sID^*}, whereby X^* is chosen by $\mathcal{A}$. In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_1^* = \mathsf{DH}(A_{i^*}, Y^*) = \mathsf{DH}(A_{i^*}, B_{\mathrm{sID}^*})$.

The number of queries to the DDH oracle is upper bounded by $3 \cdot Q_{\mathsf{H}}$. Thus,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right| \leq \mathrm{Adv}^{\mathsf{CorrGapCDH}}_{N+S,\,3Q_{\mathsf{H}}}(\mathcal{B}) \ .$$

Finally, the output of the TEST oracle in G_2 is independent of the bit b, so we have

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2} \ .$$

Collecting the probabilities yields the bound stated in Theorem 4.

□

6 Protocol HMQV

The HMQV protocol was first presented in [Kra05]. Compared to the original protocol, we include the context into the hash of the session key (see Figure 1 in the introduction). The protocol is defined relative to fixed parameters $(p, g, \mathbb{G})$ that describe a group $\mathbb{G}$ of prime order $p = |\mathbb{G}|$ and a generator g of $\mathbb{G}$. G and H are hash functions with $\mathsf{G} : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$ and $\mathsf{H} : \mathbb{G}^5 \to \{0,1\}^\lambda$, where $\lambda \geq \log(p)$.

One reason to include the context into the hash is the definition of matching sessions. The original proof is in the CK model which defines matching sessions solely based on the involved parties and transcripts. The eCK model additionally includes the session's type (initiator or responder). Now consider an active adversary that initiates two sessions of the same type. In the first query, it starts a session between parties A and B and receives message X. In the second query, it starts a session between B and A and receives message Y. Now it completes both sessions with the other message

respectively. Both sessions will compute the same key, but will not be matching sessions (as they are both of type "In"), thus the adversary can trivially win. This issue also affects other role-symmetric protocols, as already noted by Cremers in [Cre09]. We can avoid it by including the context inside the hash, as done in the analysis of [BCLS15] and also in various variants of the protocol, e. g., [Ust08, YZ13, ZZ15].[2]

We give a tight reduction under CorrCRGapCDH. However, we cannot show security against reflection attacks in general, which is why we require $i^* \neq r^*$ for all test sessions, indicated by the asterisk in IND-wFS-St*. Note that the original proof of HMQV needs the KEA assumption for the case that $i^* = r^*$ and $X \neq Y$ and the squared CDH assumption[3] for $i^* = r^*$ and $X = Y$, which is implied by the standard CDH assumption non-tightly.[4]

Theorem 5 ($(N+S, Q_\mathsf{G} + 2Q_\mathsf{H} + 1)$-CorrCRGapCDH $\xrightarrow{\text{tight, ROM}}$ HMQV IND-wFS-St)**.** *For any* IND-wFS-St* *adversary* $\mathcal{A}$ *against* HMQV *with* N *parties that establishes at most* S *sessions and issues at most* T *queries to the* Test *oracle and* Q_G *queries to random oracle* G *and* Q_H *queries to random oracle* H*, there exists an adversary* $\mathcal{B}$ *against* $(N+S, Q_\mathsf{G} + 2Q_\mathsf{H} + 1)$-CorrCRGapCDH *with running time* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *such that*

$$\mathrm{Adv}^{\text{IND-wFS-St}^*}_{\text{HMQV}}(\mathcal{A}) \leq \mathrm{Adv}^{\text{CorrCRGapCDH}}_{N+S,\, Q_\mathsf{G}+2Q_\mathsf{H}+1,\, Q_\mathsf{H}}(\mathcal{B}) + \frac{(N+S)^2}{p} \quad .$$

Proof. Let $\mathcal{A}$ be an adversary against IND-wFS-St* security of HMQV, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is the maximum number of test sessions. Consider the sequence of games in Figure 13.

Game G_0. This is the original IND-wFS-St* game. Similar to Equation (4), we implicitly assume that all long-term keys and all messages output by $\textsc{Session}_\mathrm{I}$ and $\textsc{Session}_\mathsf{R}$ are different. If such a collision happens, the game will abort. Using the birthday paradox, the probability for that can be upper bounded by $(N+S)^2/(2p)$ as there are N long-term key pairs and at most S messages, where exponents are chosen uniformly at random from $\mathbb{Z}_p$. This rules out attack (0a.), as there will be no two sessions having the same transcript. We get

$$\Pr[\text{IND-wFS-St}^{*\mathcal{A}} \Rightarrow 1] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1] + \frac{(N+S)^2}{2p} \quad .$$

[2] Even when dropping the session's type from the definition of matching sessions (similar to the original CK model), giving a tight proof for the original version of HMQV seems non-trivial since patching the random oracle H requires more care. In particular, it is always necessary to check if the input corresponds to any session for which the adversary can potentially compute the key, but the reduction itself cannot. In order to handle these queries in a naive way, the reduction needs to query the Ddh oracle once for each session, leading to $O(Q_\mathsf{H} \cdot S)$ queries.

[3] On input g^x, the squared CDH problem requires to compute g^{x^2}.

[4] We could also show security of HMQV including reflection attacks under a variant of CorrCRGapCDH that does not restrict the winning condition on $i \neq j$ and which can be reduced non-tightly to squared GapCDH.

GAMES G_0, $\boxed{G_1}$

00 $\text{cnt}_\text{P} \coloneqq N$
01 **for** $n \in [N]$
02 $\quad a_n \xleftarrow{\$} \mathbb{Z}_p$; $A_n \coloneqq g^{a_n}$
03 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) \coloneqq (A_n, a_n)$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
06 **for** $\text{sID}^* \in \mathcal{S}$
07 $\quad$ **if** $\text{Fresh}(\text{sID}^*) = \textbf{false}$
08 $\quad\quad$ **return** b
09 $\quad$ **if** $\text{Valid}(\text{sID}^*) = \textbf{false}$
10 $\quad\quad$ **return** b
11 **return** $[\![b = b']\!]$

$\underline{\text{Session}_\text{R}((i,r) \in [\text{cnt}_\text{P}] \times [N])}$
12 cnt_S **++**
13 $\text{sID} \coloneqq \text{cnt}_\text{S}$
14 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) \coloneqq (i,r)$
15 $\text{type}[\text{sID}] \coloneqq$ "Re"
16 $y \xleftarrow{\$} \mathbb{Z}_p$; $Y \coloneqq g^y$
17 $(R[\text{sID}], \text{state}[\text{sID}]) \coloneqq (Y, y)$
18 **return** (sID, Y)

$\underline{\text{Der}_\text{R}(\text{sID} \in [\text{cnt}_\text{S}], X)}$
19 **if** $\text{sKey}[\text{sID}] \neq \bot$ **or** $\text{type}[\text{sID}] \neq$ "Re"
20 $\quad$ **return** $\bot$
21 $(i,r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
22 $(Y,y) \coloneqq (R[\text{sID}], \text{state}[\text{sID}])$
23 $d \coloneqq \mathsf{G}(X, ID_r)$
24 $e \coloneqq \mathsf{G}(Y, ID_i)$
25 $\sigma \coloneqq (XA_i^d)^{y+ea_r}$
26 $K \coloneqq \mathsf{H}((A_i, A_r, X, Y), \sigma)$
27 $(I[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (X, K)$
28 **return** ε

$\underline{\mathsf{G}(Z, ID)}$
29 **if** $\mathsf{G}[Z, ID] = h$ **return** h
30 $h \xleftarrow{\$} \mathbb{Z}_p$
31 $\mathsf{G}[Z, ID] \coloneqq h$
32 **return** h

$\underline{\text{Session}_\text{I}((i,r) \in [N] \times [\text{cnt}_\text{P}])}$
33 cnt_S **++**
34 $\text{sID} \coloneqq \text{cnt}_\text{S}$
35 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) \coloneqq (i,r)$
36 $\text{type}[\text{sID}] \coloneqq$ "In"
37 $x \xleftarrow{\$} \mathbb{Z}_p$; $X \coloneqq g^x$
38 $(I[\text{sID}], \text{state}[\text{sID}]) \coloneqq (X, x)$
39 **return** (sID, X)

$\underline{\text{Der}_\text{I}(\text{sID} \in [\text{cnt}_\text{S}], Y)}$
40 **if** $\text{sKey}[\text{sID}] \neq \bot$ **or** $\text{type}[\text{sID}] \neq$ "In"
41 $\quad$ **return** $\bot$
42 $(i,r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
43 $(X,x) \coloneqq (I[\text{sID}], \text{state}[\text{sID}])$
44 $d \coloneqq \mathsf{G}(X, ID_r)$
45 $e \coloneqq \mathsf{G}(Y, ID_i)$
46 $\sigma \coloneqq (YA_r^e)^{x+da_i}$
47 $K \coloneqq \mathsf{H}((A_i, A_r, X, Y), \sigma)$
48 $(R[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (Y, K)$
49 **return** ε

$\underline{\text{Test}(\text{sID})}$
50 **if** $\text{sID} \in \mathcal{S}$ **return** $\bot$
51 **if** $\text{sKey}[\text{sID}] = \bot$ **return** $\bot$
52 $\mathcal{S} \coloneqq \mathcal{S} \cup \{\text{sID}\}$
53 $K_0^* \coloneqq \text{sKey}[\text{sID}]$
$\boxed{54 \quad K_0^* \xleftarrow{\$} \mathcal{K}}$
55 $K_1^* \xleftarrow{\$} \mathcal{K}$
56 **return** K_b^*

$\underline{\mathsf{H}(A_i, A_r, X, Y, \sigma)}$
57 **if** $\mathsf{H}[A_i, A_r, X, Y, \sigma] = K$
58 $\quad$ **return** K
59 **else**
60 $\quad K \xleftarrow{\$} \mathcal{K}$
61 $\quad \mathsf{H}[A_i, A_r, X, Y, \sigma] \coloneqq K$
62 $\quad$ **return** K

Figure 13: Games G_0-G_1 for the proof of Theorem 5. $\mathcal{A}$ has access to oracles $\text{O} \coloneqq \{\text{Session}_\text{I}, \text{Session}_\text{R}, \text{Der}_\text{I}, \text{Der}_\text{R}, \text{Rev-State}, \text{Reveal}, \text{Corrupt}, \text{RegisterLTK}, \text{Test}, \mathsf{G}, \mathsf{H}\}$, where RegisterLTK, Corrupt, Rev-State and Reveal are defined as in the original $\mathsf{IND\text{-}wFS\text{-}St}^*$ game (see Figure 8). G_0 implicitly assumes that no long-term keys or messages generated by the experiment collide.

Game G_1. In game G_1, the challenge oracle Test always outputs a uniformly random key, independent from the bit b (line 54). To show the difference between G_1 and G_0, we can use that

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1]\right| \quad = \quad \frac{1}{2}\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right| \quad ,$$

as $\Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 1] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1 \mid b = 1]$.

Due to the exclusion of collisions, a particular (test) session cannot be recreated, i.e., the adversary cannot create two sessions sID, sID′ of the same type that compute the same session key. Thus, the only way to distinguish G_1 from G_0 is to query the random oracle on the correct input. We construct adversary $\mathcal{B}$ against $(N + S, Q_{\mathsf{G}} + 2Q_{\mathsf{H}} + 1)$-CorrCRGapCDH in Figure 14 to interpolate between the two games. We now describe adversary $\mathcal{B}$ in detail.

$\mathcal{B}$ gets as input $(N + S)$ group elements and has access to oracles Corr, Ch and Ddh. The first N group elements $(A_1, ..., A_N)$ are used as public keys for the parties $\mathsf{P}_1, ..., \mathsf{P}_N$ (line 02). The remaining group elements $(B_1, ..., B_S)$ will be used as messages output by $\textsc{Session}_{\mathsf{I}}$ and $\textsc{Session}_{\mathsf{R}}$. This means that whenever $\mathcal{A}$ initiates a session sID, $\mathcal{B}$ increments the session counter and outputs the group element B_{sID} (lines 13, 43). To identify queries to the random oracle with σ, $\mathcal{B}$ uses a flag f which is added as additional entry in the list of queries to H. This helps to reduce the number of Ddh queries in oracles $\textsc{Der}_{\mathsf{I}}$ or $\textsc{Der}_{\mathsf{R}}$. Thus, whenever $\mathcal{A}$ calls one of the two oracles, $\mathcal{B}$ first checks the list of random oracle queries if there has already been a query on the correct σ indicated by $f = 1$ (lines 21, 51) and outputs the corresponding session key. If this is not the case, it checks if there is an entry with unknown σ (lines 23, 53) to keep session keys of matching sessions consistent. If there is no such entry, $\mathcal{B}$ chooses a session key uniformly at random (lines 26, 56) and adds an entry with unknown σ to the list. If $\mathcal{A}$ issues a random oracle query later, $\mathcal{B}$ checks if σ is correct using the Ddh oracle (line 66). In this case, it sets the flag f to 1. Furthermore, if there is an entry with unknown σ (line 68), it updates the entry with the correct value and outputs the corresponding key. Otherwise, f is set to 0. $\mathcal{B}$ chooses a key uniformly at random (line 73), adds an entry with f to the list and outputs the key.

Whenever $\mathcal{A}$ calls Rev-State on sID, $\mathcal{B}$ queries the Corr oracle to reveal the exponent of B_{sID} which is the message output by this session (line 36) and returns the corresponding exponent. Similarly, whenever $\mathcal{A}$ corrupts a party P_n, $\mathcal{B}$ queries the Corr oracle on n (line 32).

Whenever $\mathcal{A}$ queries the random oracle G on a new pair (Z, ID), $\mathcal{B}$ queries its Ch on Z (line 62) and returns the output.

We now show that if $\mathcal{A}$ queries H on the correct input for at least one test session, $\mathcal{B}$ is able to output a solution $C \in \mathsf{Win}$ to the CorrCRGapCDH problem.

Let $\text{sID}^* \in \mathcal{S}$ be any test session and $(A_{i^*}, A_{r^*}, X^*, Y^*)$ be the context of this test session. Then, $\mathcal{A}$ must have queried $\sigma^* = \mathsf{DH}(A_{i^*}^d X^*, A_{r^*}^e Y^*)$ to H, where $d = \mathsf{G}(X^*, ID_{r^*})$ and $e = \mathsf{G}(Y^*, ID_{i^*})$. Let d and e be the k_1-th and k_2-th output of G, which means that $(h_{k_1}, R_{k_1}) = (d, X^*)$ and $(h_{k_2}, R_{k_2}) = (e, Y^*)$. This means that there is an (updated) entry $(A_{i^*}, A_{r^*}, X^*, Y^*, \sigma^*, 1)$ in the list of queries to H. $\mathcal{B}$ has to find this query in the list and depending on which reveal queries $\mathcal{A}$ has made (i.e., which attack was performed), $\mathcal{B}$ outputs a solution to CorrCRGapCDH as described below. Therefore, we will now argue that for each possible attack listed in Table 1, there will be a valid solution.

For the following cases, there is a matching session sID′ and we assume (w.l.o.g.) that the test session is of type "Re". Then, messages X^* and Y^* are chosen by the

$\mathcal{B}^{\text{Corr,Ch,Ddh}}(A_1, ..., A_N, B_1, ..., B_S)$
00 $\text{cnt}_\text{P} := N$
01 **for** $n \in [N]$
02 $(\mathsf{pk}_n, \mathsf{sk}_n) := (A_n, \perp)$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
05 **for** $\text{sID}^* \in \mathcal{S}$
06 **if** $\text{Fresh}(\text{sID}^*) = \textbf{false}$ **return** 0
07 **if** $\text{Valid}(\text{sID}^*) = \textbf{false}$ **return** 0
08 **return** $C \in \mathsf{Win}$ (see text)

$\text{Session}_\text{R}((i,r) \in [\text{cnt}_\text{P}] \times [N])$
09 cnt_S ++
10 $\text{sID} := \text{cnt}_\text{S}$
11 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i,r)$
12 $\text{type}[\text{sID}] := \text{“Re”}$
13 $Y := B_{\text{sID}}$
14 $(R[\text{sID}], \text{state}[\text{sID}]) := (Y, \perp)$
15 **return** (sID, Y)

$\text{Der}_\text{R}(\text{sID} \in [\text{cnt}_\text{S}], X)$
16 **if** $\text{sKey}[\text{sID}] \neq \perp$ **or** $\text{type}[\text{sID}] \neq \text{“Re”}$
17 **return** $\perp$
18 $(i,r) := (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
19 $Y := R[\text{sID}]$
20 $\text{ctxt} := (A_i, A_r, X, Y)$
21 **if** $\exists \sigma$ s.t. $\mathsf{H}[\text{ctxt}, \sigma, 1] = K$
22 $\text{sKey}[\text{sID}] = K$
23 **else if** $\mathsf{H}[\text{ctxt}, \perp, \perp] = K$
24 $\text{sKey}[\text{sID}] = K$
25 **else**
26 $K \xleftarrow{\$} \mathcal{K}$
27 $\mathsf{H}[\text{ctxt}, \perp, \perp] = K$
28 $\text{sKey}[\text{sID}] = K$
29 $(I[\text{sID}], \text{sKey}[\text{sID}]) := (X, K)$
30 **return** ε

$\text{Corrupt}(n \in [N])$
31 $\text{corrupted}[n] := \textbf{true}$
32 $a_n \leftarrow \text{Corr}(n)$
33 $\mathsf{sk}_n := a_n$
34 **return** sk_n

Rev-State(sID)
35 $\text{revState}[\text{sID}] := \textbf{true}$
36 $b_{\text{sID}} \leftarrow \text{Corr}(N + \text{sID})$
37 $\text{state}[\text{sID}] := b_{\text{sID}}$
38 **return** $\text{state}[\text{sID}]$

$\text{Session}_\text{I}((i,r) \in [N] \times [\text{cnt}_\text{P}])$
39 cnt_S ++
40 $\text{sID} := \text{cnt}_\text{S}$
41 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i,r)$
42 $\text{type}[\text{sID}] := \text{“In”}$
43 $X := B_{\text{sID}}$
44 $(I[\text{sID}], \text{state}[\text{sID}]) := (X, \perp)$
45 **return** (sID, X)

$\text{Der}_\text{I}(\text{sID} \in [\text{cnt}_\text{S}], Y)$
46 **if** $\text{sKey}[\text{sID}] \neq \perp$ **or** $\text{type}[\text{sID}] \neq \text{“In”}$
47 **return** $\perp$
48 $(i,r) := (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
49 $(X,x) := (I[\text{sID}], \text{state}[\text{sID}])$
50 $\text{ctxt} := (A_i, A_r, X, Y)$
51 **if** $\exists \sigma$ s.t. $\mathsf{H}[\text{ctxt}, \sigma, 1] = K$
52 $\text{sKey}[\text{sID}] = K$
53 **else if** $\mathsf{H}[\text{ctxt}, \perp, \perp] = K$
54 $\text{sKey}[\text{sID}] = K$
55 **else**
56 $K \xleftarrow{\$} \mathcal{K}$
57 $\mathsf{H}[\text{ctxt}, \perp, \perp] = K$
58 $\text{sKey}[\text{sID}] = K$
59 $(R[\text{sID}], \text{sKey}[\text{sID}]) := (Y, K)$
60 **return** ε

$\mathsf{G}(Z, ID)$
61 **if** $\mathsf{G}[Z, ID] = h$ **return** h
62 $h \leftarrow \text{Ch}(Z)$
63 $\mathsf{G}[Z, ID] := h$
64 **return** h

$\mathsf{H}(A_i, A_r, X, Y, \sigma)$
65 **if** $\mathsf{H}[A_i, A_r, X, Y, \sigma, \cdot\,] = K$ **return** K
66 **if** $(A_i, A_r) \in \{A_1, ..., A_N\}$
and $\text{Ddh}(A_i^{\mathsf{G}(X, ID_r)} X, A_r^{\mathsf{G}(Y, ID_i)} Y, \sigma) = 1$
67 $f := 1$
68 **if** $\mathsf{H}[A_i, A_r, X, Y, \perp, \perp] = K$
69 replace $(\perp, \perp)$ with (σ, f)
70 **return** K
71 **else**
72 $f := 0$
73 $K \xleftarrow{\$} \mathcal{K}$
74 $\mathsf{H}[A_i, A_r, X, Y, \sigma, f] := K$
75 **return** K

Figure 14: Adversary $\mathcal{B}$ against $(N + S, Q_\mathsf{G} + 2Q_\mathsf{H} + 1)$-CorrCRGapCDH for the proof of Theorem 5. $\mathcal{A}$ has access to oracles $\text{O} := \{\text{Session}_\text{I}, \text{Session}_\text{R}, \text{Der}_\text{I}, \text{Der}_\text{R}, \text{Rev-State}, \text{Reveal}, \text{Corrupt}, \text{RegisterLTK}, \text{Test}, \mathsf{G}, \mathsf{H}\}$, where RegisterLTK and Reveal are defined as in Figure 13. Lines written in blue color highlight how $\mathcal{B}$ simulates G_0 and G_1, respectively.

reduction $\mathcal{B}$ as $B_{\mathrm{sID}'}$ and B_{sID^*}. In order to distinguish the session key, $\mathcal{A}$ has to compute $\sigma^* = \mathsf{DH}(A_{i^*}^d B_{\mathrm{sID}'}, A_{r^*}^e B_{\mathrm{sID}^*})$.

Attack (1.)+(2.). $\mathcal{A}$ has queried both long-term secret keys a_{i^*} and a_{r^*}. $\mathcal{A}$ is not allowed to query the state of those sessions. When $\mathcal{B}$ recognizes a query on σ^*, it first computes

$$\sigma^* \cdot (A_{r^*}^e Y^*)^{-da_{i^*}} \cdot (X^*)^{-ea_{r^*}} \quad = \quad \mathsf{DH}(X^*, Y^*)$$

and then, in order to get a valid forgery in the $\mathsf{CorrCRGapCDH}$, it has to make another query to the Ch oracle. Note that this is the only case where we need this. Let it be the k^*-th query. $\mathcal{B}$ can choose $r \in \mathbb{Z}_p$ arbitrary and query Ch on $R_{k^*} := g^r$. It will receive h_{k^*} and then computes

$$\begin{aligned}(\mathsf{DH}(X^*, Y^*))^{h_{k^*}} \cdot (Y^*)^r &= \mathsf{DH}((X^*)^{h_{k^*}} R_{k^*}, Y^*) \\ &= \mathsf{DH}((B_{\mathrm{sID}'})^{h_{k^*}} R_{k^*}, B_{\mathrm{sID}^*})\end{aligned}$$

Attack (3.)+(4.). $\mathcal{A}$ has queried both states b_{sID^*} and $b_{\mathrm{sID}'}$, but is not allowed to query the long-term secret keys of both parties. When $\mathcal{B}$ recognizes a query on σ^*, it computes

$$\begin{aligned}\left(\sigma^* \cdot (A_{i^*}^d X^*)^{-b_{\mathrm{sID}^*}}\right)^{1/e} &= \mathsf{DH}(A_{i^*}^d X^*, A_{r^*}) \\ &= \mathsf{DH}(A_{i^*}^{h_{k_1}} R_{k_1}, A_{r^*})\end{aligned} \tag{6}$$

Attack (5.)+(6.). $\mathcal{A}$ has queried the initiator's long-term secret key a_{i^*} and the responder's state b_{sID^*}, but neither the responder's long-term secret key a_{r^*} nor the initiator's state $b_{\mathrm{sID}'}$. When $\mathcal{B}$ recognizes a query on σ^*, it computes

$$\begin{aligned}\sigma^* \cdot (A_{r^*}^e Y^*)^{-da_{i^*}} &= \mathsf{DH}(A_{r^*}^e Y^*, X^*) \\ &= \mathsf{DH}(A_{r^*}^{h_{k_2}} R_{k_2}, B_{\mathrm{sID}'})\end{aligned} \tag{7}$$

Attack (7.)+(8.). This is the same as the case before, only that the adversary queried the other party's long-term key or state. $\mathcal{B}$ computes

$$\begin{aligned}\sigma^* \cdot (A_{i^*}^d X^*)^{-ea_{r^*}} &= \mathsf{DH}(A_{i^*}^d X^*, Y^*) \\ &= \mathsf{DH}(A_{r^*}^{h_{k_1}} R_{k_1}, B_{\mathrm{sID}^*})\end{aligned} \tag{8}$$

For the remaining cases, there is no matching session.

Attack (11.). The test session is of type "In" and X^* is chosen as B_{sID^*}, whereby Y^* is chosen by $\mathcal{A}$. $\mathcal{A}$ has queried the initiator's state b_{sID^*}. When $\mathcal{B}$ recognizes a query on σ^*, it computes

$$\begin{aligned}\left(\sigma^* \cdot (A_{r^*}^e Y^*)^{-b_{\mathrm{sID}^*}}\right)^{1/d} &= \mathsf{DH}(A_{r^*}^e Y, A_{i^*}) \\ &= \mathsf{DH}(A_{r^*}^{h_{k_2}} R_{k_2}, A_{i^*})\end{aligned} \tag{9}$$

ATTACK (12.). The test session is of type "Re" and Y^* is chosen as B_{sID^*}, whereby X^* is chosen by $\mathcal{A}$. $\mathcal{A}$ has queried the responder's state b_{sID^*}. When $\mathcal{B}$ recognizes a query on σ^*, it makes the same computation as in Equation (6).

ATTACK (13.). The test session is of type "In" and X^* is chosen as B_{sID^*}, whereby Y^* is chosen by $\mathcal{A}$. $\mathcal{A}$ has queried the initiator's long-term secret keys a_{i^*}. When $\mathcal{B}$ recognizes a query on σ^*, it makes the same computation as in Equation (7).

ATTACK (16.). The test session is of type "Re" and Y^* is chosen as B_{sID^*}, whereby X^* is chosen by $\mathcal{A}$. $\mathcal{A}$ has queried the responder's long-term secret keys a_{r^*}. When $\mathcal{B}$ recognizes a query on σ^*, it makes the same computation as in Equation (8).

We showed that in each of these cases, the adversary outputs a correct solution for $\mathsf{CorrCRGapCDH}$. Note that the requirement that $i^* \neq r^*$ is needed for attacks (3.)+(4.), (11.) and (12.). The number of queries to the CH oracle is upper bounded by $Q_{\mathsf{G}}+2Q_{\mathsf{H}}+1$ and the number of queries to the DDH oracle by Q_{H}. Thus,

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b=0] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1 \mid b=0]\right| \leq \mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{N+S,\, Q_{\mathsf{G}}+2Q_{\mathsf{H}}+1,\, Q_{\mathsf{H}}}(\mathcal{B}) \ .$$

Finally, the output of the TEST oracle in G_1 is independent of the bit b, so we have

$$\Pr[G_1^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2} \ .$$

Collecting the probabilities yields the bound stated in Theorem 5.

□

7 Concrete Bounds in the Generic Group Model

7.1 Generic Hardness of NAXOS

When analyzing NAXOS and $\mathsf{X3DH}^-$, we obtain the following generic bound.

Corollary 2 (Generic Hardness of NAXOS and $\mathsf{X3DH}^-$)**.** *For any adversary $\mathcal{A}$ ($\mathcal{B}$) against* NAXOS *($\mathsf{X3DH}^-$) in the generic group and the random oracle model running in time $\mathbf{T}(\mathcal{A})$ ($\mathbf{T}(\mathcal{B})$), we have*

$$\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{NAXOS},\mathrm{GGM}}(\mathcal{A}) = \mathrm{Adv}^{\mathsf{IND\text{-}wFS}}_{\mathsf{X3DH}^-,\mathrm{GGM}}(\mathcal{B}) = \Theta\left(\frac{\mathbf{T}(\mathcal{A})^2}{p}\right) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against NAXOS with N parties that establishes at most S sessions and issues at most T queries to the TEST oracle, at most Q_{G} queries to random oracle G, at most Q_{H} queries to random oracle H, and at most Q_{OP} queries to the group oracle. Then $\mathbf{T}(\mathcal{A}) = Q_{\mathrm{OP}} + N + S + T + Q_{\mathrm{RO}}$ is the running time of adversary $\mathcal{A}$. Let $\lambda \geq \log(p)$ be the output length of G. Combining Corollary 1 with Theorem 4 we obtain

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{NAXOS},\mathrm{GGM}}(\mathcal{A}) &\leq \frac{(Q_{\mathrm{OP}}+N+S+1)^2}{2p} + \frac{6Q_{\mathsf{H}}}{p} + \frac{3(N+S)^2}{p} + \frac{S^2}{p} + \frac{2Q_{\mathsf{G}}S}{p} \\
&= O\left(\frac{\mathbf{T}(\mathcal{A})^2}{p}\right) ,
\end{aligned}$$

where we bounded the term $\frac{(N+S-n')^2}{2p} + \frac{N+S-n'}{p} + \frac{(N+S)^2}{p} \leq \frac{3(N+S)^2}{p}$.

The lower bound $\Omega(\frac{\mathbf{T}(\mathcal{A})^2}{p})$ follows by a simple discrete logarithm attack on NAXOS. The same analysis applies to X3DH$^-$ since CorrGapCDH$(N+S)$ tightly implies $(N+S, S)$-CorrAGapCDH. □

The corollary with matching upper and lower bounds shows that the generic bounds on NAXOS and X3DH$^-$ are optimal.

7.2 Generic Hardness of HMQV

For HMQV, we split the running time of $\mathcal{A}$ into its offline running time by $\mathbf{T}_{\text{OFF}}(\mathcal{A}) = Q_{\text{OP}} + Q_{\text{RO}}$ and its online running time by $\mathbf{T}_{\text{ON}}(\mathcal{A}) = N + S + T$. It is reasonable to assume that $\mathbf{T}_{\text{OFF}}(\mathcal{A}) \gg \mathbf{T}_{\text{ON}}(\mathcal{A})$, i.e., the adversary spends much more time on offline queries than on online queries.

Corollary 3 (Generic Hardness of HMQV)**.** *For any adversary $\mathcal{A}$ against* HMQV *in the generic group and the random oracle model running in online time $\mathbf{T}_{\text{ON}}(\mathcal{A})$ and offline time $\mathbf{T}_{\text{OFF}}(\mathcal{A})$, we have*

$$\text{Adv}^{\text{IND-wFS-St}^*}_{\text{HMQV,GGM}}(\mathcal{A}) = O\left(\frac{\mathbf{T}_{\text{OFF}}(\mathcal{A})^2 + \mathbf{T}_{\text{OFF}}(\mathcal{A}) \cdot \mathbf{T}_{\text{ON}}(\mathcal{A})^2}{p}\right) .$$

Proof. Let $\mathcal{A}$ be an adversary against HMQV with N parties that establishes at most S sessions and issues at most T queries to the TEST oracle, at most $Q_{\text{RO}} := Q_{\mathsf{G}} + Q_{\mathsf{H}}$ queries to random oracles G and H, and at most Q_{OP} queries to the group oracle. Then $\mathbf{T}_{\text{OFF}}(\mathcal{A}) = Q_{\text{OP}} + Q_{\text{RO}}$ and $\mathbf{T}_{\text{ON}}(\mathcal{A}) = N + S + T$ are the offline resp. online running times of adversary $\mathcal{A}$. Combining Theorem 2 with Theorem 5 and assuming $Q_{\text{OP}} \geq (N+S)$, we obtain

$$\begin{aligned}\text{Adv}^{\text{IND-wFS-St}^*}_{\text{HMQV,GGM}}(\mathcal{A}) &\leq \frac{(Q_{\text{OP}} + N + S + 1)^2}{2p} + \frac{2Q_{\text{RO}}}{p} + \frac{3(N+S)^2(2Q_{\text{RO}}+1)}{p} \\ &= O\left(\frac{\mathbf{T}_{\text{OFF}}(\mathcal{A})^2}{p} + \frac{\mathbf{T}_{\text{OFF}}(\mathcal{A}) \cdot \mathbf{T}_{\text{ON}}(\mathcal{A})^2}{p}\right) ,\end{aligned}$$

where we bounded the term

$$\frac{(N+S-n')^2(2Q_{\text{RO}}+1)}{2p} + \frac{(N+S-n')(2Q_{\text{RO}}+1)}{p} + \frac{(N+S)^2}{p} \leq \frac{3(N+S)^2(2Q_{\text{RO}}+1)}{p} . \qquad \square$$

For HMQV we have an additive term in addition to the optimal bound $\Omega(\frac{\mathbf{T}_{\text{OFF}}(\mathcal{A})^2}{p})$. We claim that as long as $\mathbf{T}_{\text{ON}}(\mathcal{A})$ is not too large, there is no need to increase the size of group $\mathbb{G}$.

We fix a group $\mathbb{G}$ where the DL problem has 128-bit security, meaning $p \approx 2^{256}$. Assuming $\mathbf{T}_{\text{ON}}(\mathcal{A}) \leq 2^{64}$ and $\mathbf{T}_{\text{OFF}}(\mathcal{A}) \leq 2^{128}$, we obtain by the corollary

$$\frac{\text{Adv}^{\text{IND-wFS-St}^*}_{\text{HMQV,GGM}}(\mathcal{A})}{\mathbf{T}(\mathcal{A})} = \frac{\text{Adv}^{\text{IND-wFS-St}^*}_{\text{HMQV,GGM}}(\mathcal{A})}{\mathbf{T}_{\text{ON}}(\mathcal{A}) + \mathbf{T}_{\text{OFF}}(\mathcal{A})} \lesssim \frac{\mathbf{T}_{\text{OFF}}(\mathcal{A}) + \mathbf{T}_{\text{ON}}(\mathcal{A})^2}{p} \lesssim 2^{-128} .$$

That is, HMQV has 128-bit security.

Acknowledgments. We would like to thank the anonymous reviewers for their thoughtful and constructive comments. Eike Kiltz was supported by the Deutsche Forschungsgemeinschaft (DFG, German research Foundation) as part of the Excellence Strategy of the German Federal and State Governments – EXC 2092 CASA - 390781972, and by the European Union (ERC AdG REWORC - 101054911). Jiaxin Pan was supported by the Research Council of Norway under Project No. 324235. Doreen Riepel was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001. 71, 72, 77

[BCLS15] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 689–718. Springer, Heidelberg, April 2015. 70, 73, 94

[BD20] Mihir Bellare and Wei Dai. The multi-base discrete logarithm problem: Tight reductions and non-rewinding proofs for Schnorr identification and signatures. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 529–552. Springer, Heidelberg, December 2020. 75

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658. Springer, Heidelberg, March 2015. 76

[BP02] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 162–177. Springer, Heidelberg, August 2002. 104

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 76

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994. 70

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. 76

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797. Springer, Heidelberg, August 2019. 69, 70, 71, 75, 76, 87

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001. 70

[CKMS16] Sanjit Chatterjee, Neal Koblitz, Alfred Menezes, and Palash Sarkar. Another look at tightness II: Practical issues in cryptography. Cryptology ePrint Archive, Report 2016/360, 2016. https://eprint.iacr.org/2016/360. 74, 77

[Cre09] Cas J.F. Cremers. Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange. Cryptology ePrint Archive, Report 2009/253, 2009. https://eprint.iacr.org/2009/253. 94

[DRS20] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 341–373. Springer, Heidelberg, May 2020. 76

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 2020. 75

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146. Springer, Heidelberg, October 2021. 70, 76, 82

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 33–61. Springer, Heidelberg, August 2016. 73, 75, 76, 79, 104

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005. 69, 70, 71, 73, 93

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007. 69, 70, 71, 73, 74, 82

[LMQ+03] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Des. Codes Cryptography*, 28(2):119–134, March 2003. 70

[LS17] Yong Li and Sven Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1343–1360. ACM Press, October / November 2017. 84

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005. 71, 76, 79

[MP16] Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol. 2016. 69

[MQV95] Alfred Menezes, Minghua Qu, and Scott A. Vanstone. Some new key agreement protocols providing mutual implicit authentication. 1995. 70

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997. 77

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 104–118. Springer, Heidelberg, February 2001. 71, 72, 77

[P1300] IEEE Standard Specifications for Public-Key Cryptography. *IEEE Std 1363-2000*, pages 1–228, 2000. 70

[Per17] Trevor Perrin. The noise protocol framework. http://noiseprotocol.org/noise.html, 2017. 76

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. 71, 73

[PW11] Jiaxin Pan and Libin Wang. TMQV: A strongly eCK-secure Diffie-Hellman protocol without gap assumption. In Xavier Boyen and Xiaofeng Chen, editors, *ProvSec 2011*, volume 6980 of *LNCS*, pages 380–388. Springer, Heidelberg, October 2011. 71, 76

[SE16] Augustin P. Sarr and Philippe Elbaz-Vincent. On the security of the (F)HMQV protocol. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 207–224. Springer, Heidelberg, April 2016. 71

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 71, 73, 76, 79

[Ust08] Berkant Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Des. Codes Cryptogr.*, 46(3):329–342, 2008. 71, 73, 76, 94

[Ust09] Berkant Ustaoglu. Comparing sessionstatereveal and ephemeralkeyreveal for Diffie-Hellman protocols. In Josef Pieprzyk and Fangguo Zhang, editors, *ProvSec 2009*, volume 5848 of *LNCS*, pages 183–197. Springer, Heidelberg, November 2009. 76

[YZ13] Andrew Chi-Chih Yao and Yunlei Zhao. OAKE: a new family of implicitly authenticated Diffie-Hellman protocols. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1113–1128. ACM Press, November 2013. 76, 94

[ZZ15] Shijun Zhao and Qianying Zhang. sHMQV: An efficient key exchange protocol for power-limited devices. Cryptology ePrint Archive, Report 2015/110, 2015. https://eprint.iacr.org/2015/110. 94

A Omitted Proofs from Section 3

A.1 Proof of Theorem 1

Theorem 1 is proved by Lemmata 6 to 8. Proofs of these lemmas are very similar to those in Section 3.2 of [KMP16], and are fairly straightforward. Hence, we only sketch them here, instead of defining them with pseudo-codes. We first recall the reset lemma from [BP02].

Lemma 5 (Reset Lemma [BP02])**.** *Let H be a non-empty set. Let $\mathcal{C}$ be a randomized algorithm that on input (I, h) returns a pair (b, σ), where b is a bit and σ is called the side output. The accepting probability of $\mathcal{C}$ is defined as*

$$\mathsf{acc} := \Pr[b = 1 \mid h \xleftarrow{\$} H; (b, \sigma) \xleftarrow{\$} \mathcal{C}(I, h)]$$

The reset algorithm $\mathcal{R}_{\mathcal{C}}$ associated to $\mathcal{C}$ is the randomized algorithm that takes input I and proceeds as follows.

Algorithm $\mathcal{R}_{\mathcal{C}}(I)$*:*
00 *Pick random coins* ρ
01 $h \xleftarrow{\$} H$
02 $(b, s) \xleftarrow{\$} \mathcal{C}(I, h; \rho)$
03 ***if*** $b = 0$ ***return*** $(\perp, \perp)$ *// Abort in Phase 1*
04 $h' \xleftarrow{\$} H$
05 $(b', s') \xleftarrow{\$} \mathcal{C}(I, h'; \rho)$
06 ***if*** $h \neq h'$ ***and*** $b' = 1$ ***return*** (s, s')
07 ***else return*** $(\perp, \perp)$

Let

$$\mathsf{frk} := \Pr[(s, s') \neq (\perp, \perp) \mid (s, s') \xleftarrow{\$} \mathcal{R}_{\mathcal{C}}(I)].$$

Then

$$\mathsf{frk} \geq \left(\mathsf{acc} - \frac{1}{|H|}\right)^2.$$

Lemma 6 ($\mathsf{GapCDH} \xrightarrow{\text{rewind.}} (2,1)\text{-}\mathsf{CorrCRGapCDH}$)**.** *For any adversary* $\mathcal{A}$ *against* $(2,1)$-$\mathsf{CorrCRGapCDH}$*, there exists an adversary* $\mathcal{B}$ *against* GapCDH *with* $\mathbf{T}(\mathcal{B}) \approx 2\mathbf{T}(\mathcal{A})$ *and*

$$\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{2,1,Q_{\mathrm{DDH}}}(\mathcal{A}) \leq \sqrt{\mathrm{Adv}^{\mathsf{GapCDH}}_{Q_{\mathrm{DDH}}}(\mathcal{B})} + \frac{1}{p}\,.$$

Proof. We use the reset lemma (Lemma 5) with $H := \mathbb{Z}_p$ and $I := (A_1, A_2) := (g^{a_1}, g^{a_2})$. We first define an algorithm $\mathcal{C}((A_1, A_2), h; \rho)$ with oracle access to DDH that calls $\mathcal{A}((A_1, A_2); \rho)$. It answers $\mathcal{A}$'s single CH(R) query with h and $\mathcal{A}$'s DDH queries with DDH of GapCDH in a straightforward way. In $(2,1)$-$\mathsf{CorrCRGapCDH}$, $\mathcal{A}$ cannot ask any queries to CORR, as then $\mathcal{A}$ cannot win any more. Upon receiving $\mathcal{A}$'s forgery C, $\mathcal{C}$ checks if $C = (A_i^h R)^{a_j}$ with its DDH oracle, where $1 \leq i \neq j \leq 2$. If it holds, $\mathcal{C}$ returns $(1, s := (R, h, C))$; otherwise, $\mathcal{C}$ returns $(0, \perp)$. Thus, $\mathcal{C}$ returns $b = 1$ as long as $\mathcal{A}$ wins:

$$\mathsf{acc} := \mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{2,1,Q_{\mathrm{DDH}}}(\mathcal{A}).$$

Now the reduction $\mathcal{B}$ that solves the GapCDH problem is constructed as follows. $\mathcal{B}$ gets the problem instance (A_1, A_2) and has oracle access to DDH from the GapCDH problem. Next, it runs $(s, s') \xleftarrow{\$} \mathcal{R}_{\mathcal{C}}(I := (A_1, A_2))$ as described in Lemma 5, with $\mathcal{C}$ defined above. If $(s, s') \neq (\perp, \perp)$, then both transcripts $s = (R, h, C)$ and $s' = (R, h', C')$ are $(2,1)$-$\mathsf{CorrCRGapCDH}$ forgeries and $h \neq h'$. Wlog. we assume $C = (A_1^h R)^{a_2}$ and $C' = (A_1^{h'} R)^{a_2}$. $\mathcal{B}$ can compute $A_1^{a_2}$ as

$$A_1^{a_2} := \left(\frac{C}{C'}\right)^{(h-h')^{-1}}.$$

By construction, $\mathcal{B}$ is successful iff $\mathcal{R}_{\mathcal{C}}$ is successful. By Lemma 5, $\mathcal{B}$'s success probability is bounded by

$$\mathrm{Adv}^{\mathsf{GapCDH}}_{Q_{\mathrm{DDH}}}(\mathcal{B}) = \mathsf{frk} \geq \left(\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{2,1,Q_{\mathrm{DDH}}}(\mathcal{A}) - \frac{1}{|\mathbb{Z}_p|}\right)^2.$$

During the simulation, $\mathcal{B}$ queries DDH at most $2(Q_{\text{DDH}}+1)$ times ($Q_{\text{DDH}}+1$ times for each execution of $\mathcal{C}$). The running time of $\mathcal{B}$, $\mathbf{T}(\mathcal{B})$, is that of $\mathcal{R}_{\mathcal{C}}$, namely, $2\mathbf{T}(\mathcal{A})$, plus the minor overhead of asking additional DDH and computing the final result $A_1^{a_2}$ from C and C'. We write $\mathbf{T}(\mathcal{B}) \approx 2\mathbf{T}(\mathcal{A})$ to indicate that this is the dominating running time of $\mathcal{B}$. □

Lemma 7 ($(2,1)$-CorrCRGapCDH $\xrightarrow{n^2}$ $(n,1)$-CorrCRGapCDH)**.** *For any adversary $\mathcal{A}$ against $(n,1)$-CorrCRGapCDH, there exists an adversary $\mathcal{B}$ against $(2,1)$-CorrCRGapCDH with*

$$\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,1,Q_{\text{DDH}}}(\mathcal{A}) \leq n^2 \cdot \mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{2,1,Q_{\text{DDH}}}(\mathcal{B}). \tag{10}$$

Proof. The simulator receives B_1, B_2 from the challenger, and chooses a pair of indexes $i^*, j^* \in [n]$ at random. For $i \in [n] \setminus \{i^*, j^*\}$, it chooses $a_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $A_i := g^{a_i}$. Finally, it sets $(A_{i^*}, A_{j^*}) := (B_1, B_2)$, and calls $\mathcal{A}(A_1, \ldots, A_n)$. Any query to DDH is forwarded to the challengers oracle. The same is done for the single challenge call $\text{CH}(R_1)$, which returns h_1. Calls to $\text{CORR}(A_i)$ for $i \neq i^*, j^*$ are answered with the corresponding discrete logarithm a_i. The queries $\text{CORR}(A_{i^*})$ and $\text{CORR}(A_{j^*})$ are forwarded to the challenger (note that such a query will cause us to lose the game). This simulator perfectly simulates a $\mathsf{CorrCRGapCDH}_{n,1,Q_{\text{DDH}}}$ game from $\mathcal{A}$'s viewpoint. At some point, $\mathcal{A}$ will output a forgery

$$C \in \{(A_i^{h_1} \cdot R_1)^{a_j} \mid (i,j) \in ([n] \setminus \mathcal{L}_A)^2 \wedge (i \neq j)\}.$$

For the simulator to win, we require that either $(i^*, j^*) = (i, j)$ or $(i^*, j^*) = (j, i)$, meaning that we chose the correct pair of indexes from a set of $\binom{n}{2}$ pairs. This means that the probability of choosing correctly is $\frac{2}{n(n-1)} \geq \frac{1}{n^2}$, which gives us the result. □

Lemma 8 ($(n,1)$-CorrCRGapCDH $\xrightarrow{Q_{\text{CH}}}$ (n, Q_{CH})-CorrCRGapCDH)**.** *For any adversary $\mathcal{A}$ against (n, Q_{CH})-CorrCRGapCDH, there exists an adversary $\mathcal{B}$ against $(n,1)$-CorrCRGapCDH with*

$$\mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,Q_{\text{CH}},Q_{\text{DDH}}}(\mathcal{A}) \leq Q_{\text{CH}} \cdot \mathrm{Adv}^{\mathsf{CorrCRGapCDH}}_{n,1,Q_{\text{DDH}}}(\mathcal{B}). \tag{11}$$

Proof. The simulation starts with $\mathcal{B}$ picking a random integer $r \xleftarrow{\$} [Q_{\text{CH}}]$. Then, any query from $\mathcal{A}$ to either DDH or CORR is forwarded to the appropriate oracle by. When responding to calls to CH, $\mathcal{B}$ keeps track of how many such queries $\mathcal{A}$ has submitted. For every query $\text{CH}(R_i)$ with $i \in [Q_{\text{CH}}] \setminus \{r\}$, $\mathcal{B}$ returns $h_i \xleftarrow{\$} \mathbb{Z}_p$. On the r'th query, $\mathcal{B}$ submits a query to the challengers CH oracle, and forwards the response to $\mathcal{A}$. At some point $\mathcal{A}$ will submit a forgery

$$C \in \{(A_i^{h_k} \cdot R_k)^{a_j} \mid (i,j,k) \in ([n] \setminus \mathcal{L}_A)^2 \times [Q_{\text{CH}}] \wedge (i \neq j)\}.$$

Then we have that $\Pr[k = r] \leq \frac{1}{Q_{\text{CH}}}$, in which case $\mathcal{B}$ forwards C and wins the game. □

A.2 Proofs of Other Useful Lemmas

Lemmata 1 to 3 and Lemma 9 are about the relation among GapCDH, CorrGapCDH and CorrAGapCDH. These lemmas complete Figure 5 of Section 3. We give their proofs here.

Proof (of Lemma 1). $\mathcal{B}$ receives a tuple $(A_1, \ldots, A_n)$ from the $\mathsf{CorrCRGapCDH}_{n,1,Q_{\mathrm{DDH}}}$ challenger, and forwards it to adversary $\mathcal{A}$. When $\mathcal{A}$ submits a query to CORR or DDH, $\mathcal{B}$ forwards it to the appropriate $\mathsf{CorrCRGapCDH}_{n,1,Q_{\mathrm{DDH}}}$ oracle, and returns the response. At some point, $\mathcal{A}$ will output a forgery C. We assume that the forgery is valid, meaning that $C = A_i^{a_j}$ for $i \neq j$ and $i, j \notin \mathcal{L}_A$. The reduction queries $\mathrm{CH}(R = 1)$, and receives a challenge h. Finally, $\mathcal{B}$ submits a forgery $C^h = (A_i^{a_j})^h = (A_i^h \cdot 1)^{a_j}$, which is clearly a valid forgery in the $\mathsf{CorrCRGapCDH}_{n,1,Q_{\mathrm{DDH}}}$ game. □

Proof (of Lemma 2). Let $\mathcal{A}$ be an adversary that solves the n-$\mathsf{CorrGapCDH}$ problem. Given a GapCDH instance $(B_1, B_2) := (g^{b_1}, g^{b_2}) \in \mathbb{G}^2$ and the oracle DDH, $\mathcal{B}$ guesses two distinct i^*, j^* from $[n]$ and define $(A_{i^*}, A_{j^*}) := (B_1, B_2)$ and, for $i \in [n] \setminus \{i^*, j^*\}$, $\mathcal{B}$ chooses $a_i \xleftarrow{\$} \mathbb{Z}_p$ and compute $A_i := g^{a_i}$. Then $\mathcal{B}$ calls $\mathcal{A}(A_1, \ldots, A_n)$ and answer $\mathcal{A}$'s oracle queries in a straightforward way:

- Upon $\mathrm{CORR}(i \in [n])$, if $i = i^*$ or $i = j^*$, $\mathcal{B}$ aborts; otherwise, $\mathcal{B}$ returns a_i.
- Upon $\mathrm{DDH}(X, Y, Z)$, $\mathcal{B}$ answers with the Boolean value $[\![Z = Y^{\mathsf{DL}_g(X)}]\!]$.

When $\mathcal{A}$ outputs it forgery C and terminates, if $\mathcal{B}$ guessed (i^*, j^*) correctly and C is valid, namely, $C = A_{i^*}^{a_{j^*}} = B_1^{b_2}$, then $\mathcal{B}$ submits C to break the GapCDH problem; otherwise, $\mathcal{B}$ aborts. Thus, $\mathcal{B}$ wins if it guesses i^*, j^* correctly and $\mathcal{A}$ wins. Moreover, since two distinct i^*, j^* are chosen randomly from $[n]$, the probability that $\mathcal{B}$ correctly guesses these indices is bounded by $1/n^2$. Hence, we arrive at $\mathrm{Adv}^{\mathsf{GapCDH}}_{Q_{\mathrm{DDH}}}(\mathcal{B}) \geq \frac{1}{n^2}\mathrm{Adv}^{\mathsf{CorrGapCDH}}_{n,Q_{\mathrm{DDH}}}(\mathcal{A})$. □

Lemma 9 ($\mathsf{GapCDH} \to n$-GapCDH)**.** *For any adversary $\mathcal{A}$ against n-GapCDH ($n > 2$), there exists an adversary $\mathcal{B}$ against GapCDH with*

$$\mathrm{Adv}^{\mathsf{GapCDH}}_{n,Q_{\mathrm{DDH}}}(\mathcal{A}) \leq 2\mathrm{Adv}^{\mathsf{GapCDH}}_{2,Q_{\mathrm{DDH}}}(\mathcal{B}).$$

Proof. We prove this tight implication by using a re-randomization argument. $\mathcal{B}$ is constructed in the following way to break the GapCDH assumption: Upon receiving (B_1, B_2) from the GapCDH challenger, for $i \in [n]$, $\mathcal{B}$ chooses $r_i \xleftarrow{\$} \mathbb{Z}_p$, flips a random coin $\delta_i \in \{0, 1\}$, and computes $A_i := B_1 \cdot g^{r_i}$ if $\delta_i = 0$ or $A_i := B_2 \cdot g^{r_i}$ if $\delta_i = 1$. Then $\mathcal{B}$ calls the adversary $\mathcal{A}(A_1, \ldots, A_n)$. Queries to $\mathrm{DDH}(X, Y, Z)$ are forwarded to the challengers DDH oracle.

Eventually $\mathcal{A}$ outputs its forgery C. If $C = A_i^{a_j}$ and $\delta_i \neq \delta_j$, then $\mathcal{B}$ breaks the GapCDH assumption with $C' := C/(B_1^{r_j} B_2^{r_i} g^{r_i r_j})$, assuming $\delta_i = 0$ and $\delta_j = 1$ w.l.o.g.. We note that δ_i and δ_j are two independent random coins and thus $\Pr[\delta_i \neq \delta_j] = \Pr[(\delta_i, \delta_j) = (0, 1)] + \Pr[(\delta_i, \delta_j) = (1, 0)] = 1/2$. This concludes the lemma. □

Proof (of Lemma 3). We construct a reduction $\mathcal{B}$ as follow: $\mathcal{B}$ receives a GapCDH instance $(B_1, B_2) := (g^{b_1}, g^{b_2})$ and guesses an index $i^* \xleftarrow{\$} [n_1]$ on which $\mathcal{A}$ will output a forgery. $\mathcal{B}$ defines $A_{i^*} := B_1$ and for $i \in [n_1] \setminus \{i^*\}$ $\mathcal{B}$ chooses $a_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $A_i := g^{a_i}$. For $i \in [n_1 + 1, n]$ $\mathcal{B}$ re-randomizes B_2 to get A_i as in Lemma 9, namely, it chooses $r_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $A_i := B_2 \cdot g^{r_i}$. Then $\mathcal{B}$ calls $\mathcal{A}$ with $(A_1, \ldots, A_n)$ and answers $\mathcal{A}$'s oracle queries as follows:

- Upon $\mathrm{DDH}(X, Y, Z)$, $\mathcal{B}$ forwards it to the corresponding GapCDH oracle.
- Upon $\mathrm{CORR}_{n_1}(i \in n_1)$, if $i = i^*$ then $\mathcal{B}$ aborts; else, $\mathcal{B}$ stores i in $\mathcal{L}_A$ and returns a_i.

Eventually, $\mathcal{A}$ outputs its forgery C and terminates. If C is a valid forgery and $\mathcal{B}$ guesses i^* correctly, then $C = A_{i^*}^{b_2+r_{j^*}} = B_1^{b_2+r_{j^*}}$ and $\mathcal{B}$ returns $C' := C/B_1^{r_{j^*}}$ as its forgery to GapCDH. We note that the probability that i^* is a correct guess is bounded by $1/n_1$, which concludes the proof. □

B Proof of Theorem 3

Proof. Let $\mathcal{A}$ be an adversary against IND-wFS security of X3DH$^-$, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is the maximum number of test sessions. Consider the sequence of games in Figure 15.

GAME G_0. This is the original IND-wFS game. As in Equation (4), we implicitly assume that all long-term keys and all messages output by SESSION$_\mathsf{I}$ and SESSION$_\mathsf{R}$ are different. If such a collision happens, the game will abort. Using the birthday paradox, the probability for that can be upper bounded by $(N+S)^2/(2p)$ as there are N long-term key pairs and at most S messages, where exponents are chosen uniformly at random from $\mathbb{Z}_p$. This rules out attack (0a.), as there will be no two sessions having the same transcript. We get

$$\Pr[\mathsf{IND\text{-}wFS}^{\mathcal{A}} \Rightarrow 1] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1] + \frac{(N+S)^2}{2p} \ .$$

GAME G_1. In game G_1, the challenge oracle TEST outputs a uniformly random key for test sessions which will not have a matching session. Therefore, we initialize a set $\mathcal{P}$ and each time DER$_\mathsf{I}$ or DER$_\mathsf{R}$ is called, we check if there is a potential matching session (lines 23, 40), i.e., the input message was output by SESSION$_\mathsf{I}$ or SESSION$_\mathsf{R}$. If this is the case, we add the session ID to the set (lines 24, 41). A TEST query on such an sID will behave exactly as in G_0, whereas for the other sessions, it outputs a random key independently of bit b (line 53). Similar to Equation (5), it holds that

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1]\right| \quad = \quad \frac{1}{2}\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b=0] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1 \mid b=0]\right| \ .$$

We construct adversary $\mathcal{B}$ against $(N+S,N)$-CorrAGapCDH in Figure 16. $\mathcal{B}$ gets as input $(N+S)$ group elements and has access to oracles CORR and DDH. The first N group elements $(A_1,...,A_N)$ are used as public keys for the parties $\mathsf{P}_1,...,\mathsf{P}_N$ (line 02). The remaining group elements $(B_1,...,B_S)$ will be used as messages output by SESSION$_\mathsf{I}$ and SESSION$_\mathsf{R}$. This means that whenever $\mathcal{A}$ initiates a session sID, $\mathcal{B}$ increments the session counter and outputs the next group element B_{sID} (lines 15, 22). Whenever $\mathcal{A}$ calls DER$_\mathsf{I}$ or DER$_\mathsf{R}$, $\mathcal{B}$ behaves exactly as adversary $\mathcal{B}$ in Figure 11. In particular, it keeps the session keys consistent with the random oracle H using its DDH oracle. Whenever $\mathcal{A}$ corrupts a party P_n, $\mathcal{B}$ queries its own oracle CORR on n (line 26) and outputs the corresponding exponent.

The TEST oracle outputs the real session key for all queries if $b=0$ (line 32). If $b=1$, it outputs a random key for all those test session that will not have a matching session (line 36), and the real session key otherwise (line 34). Due to the exclusion of

Figure 15: Games G_0-G_2 for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{Session}_I, \text{Session}_R, \text{Der}_I, \text{Der}_R, \text{Reveal}, \text{Corrupt}, \text{RegisterLTK}, \text{Test}, \mathsf{H}\}$, where RegisterLTK, Corrupt and Reveal are defined as in the original IND-wFS game (Fig. 8). G_0 implicitly assumes that no long-term keys or messages generated by the experiment collide.

collisions, a particular (test) session cannot be recreated, i.e., the adversary cannot create two sessions sID, sID′ of the same type that compute the same session key. Thus, the only way to distinguish the two games is to query the random oracle on the correct

$\mathcal{B}^{\text{Corr,Ddh}}(A_1, ..., A_N, B_1, ..., B_S)$
00 $\text{cnt}_\text{P} := N$
01 **for** $n \in [N]$
02 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) := (A_n, \bot)$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
05 **for** $\text{sID}^* \in \mathcal{S}$
06 $\quad$ **if** $\textsc{Fresh}(\text{sID}^*) = \textbf{false}$
07 $\quad\quad$ **return** b
08 $\quad$ **if** $\textsc{Valid}(\text{sID}^*) = \textbf{false}$
09 $\quad\quad$ **return** b
10 **return** $C \in \mathsf{Win}$ (see text)

$\textsc{Session}_\text{R}((i,r) \in [\text{cnt}_\text{P}] \times [N])$
11 cnt_S ++
12 $\text{sID} := \text{cnt}_\text{S}$
13 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i, r)$
14 $\text{type}[\text{sID}] := \text{“Re”}$
15 $Y := B_{\text{sID}}$
16 $(R[\text{sID}], \text{state}[\text{sID}]) := (Y, \bot)$
17 **return** (sID, Y)

$\textsc{Session}_\text{I}((i,r) \in [N] \times [\text{cnt}_\text{P}])$
18 cnt_S ++
19 $\text{sID} := \text{cnt}_\text{S}$
20 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i, r)$
21 $\text{type}[\text{sID}] := \text{“In”}$
22 $X := B_{\text{sID}}$
23 $(I[\text{sID}], \text{state}[\text{sID}]) := (X, \bot)$
24 **return** (sID, X)

$\textsc{Corrupt}(n \in [N])$
25 $\text{corrupted}[n] := \textbf{true}$
26 $a_n \leftarrow \textsc{Corr}(n)$
27 $\mathsf{sk}_n := a_n$
28 **return** sk_n

$\textsc{Test}(\text{sID})$
29 **if** $\text{sID} \in \mathcal{S}$ **return** $\bot$
30 **if** $\text{sKey}[\text{sID}] = \bot$ **return** $\bot$
31 $\mathcal{S} := \mathcal{S} \cup \{\text{sID}\}$
32 $K_0^* := \text{sKey}[\text{sID}]$
33 **if** $\text{sID} \in \mathcal{P}$
34 $\quad K_1^* := \text{sKey}[\text{sID}]$
35 **else**
36 $\quad K_1^* \xleftarrow{\$} \mathcal{K}$
37 **return** K_b^*

Figure 16: Adversary $\mathcal{B}$ against $(N+S, N)$-CorrAGapCDH for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\text{O} := \{\textsc{Session}_\text{I}, \textsc{Session}_\text{R}, \textsc{Der}_\text{I}, \textsc{Der}_\text{R}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{RegisterLTK}, \textsc{Test}, \mathsf{H}\}$, where RegisterLTK and Reveal are defined as in game G_1. Oracles $\textsc{Der}_\text{I}$, $\textsc{Der}_\text{R}$ and H are defined as for adversary $\mathcal{B}$ in Figure 11. Lines written in blue color highlight how $\mathcal{B}$ simulates G_0 and G_1, respectively.

input for a test session which does not have a matching session. Next we show that if this happens, $\mathcal{B}$ is able to output a solution $C \in \mathsf{Win}$ to the CorrAGapCDH problem.

Let $\text{sID}^* \in \mathcal{S}$ be such a test session and $\mathsf{H}[A_{i^*}, A_{r^*}, X^*, Y^*, Z_1^*, Z_2^*, Z_3^*, 1] = \text{sKey}[\text{sID}^*]$ be the corresponding entry in the list of hash queries. $\mathcal{B}$ has to find this query in the list and depending on which reveal queries $\mathcal{A}$ has made (i.e., which attack was performed), $\mathcal{B}$ returns either Z_1^*, Z_2^* or Z_3^* as described below. Therefore, we will now argue that for those attacks in Table 2 which consider non-matching sessions, there will be a valid solution for CorrAGapCDH.

Attack (13.). The test session is of type “In” and $\mathcal{A}$ has queried the initiator's long-term secret keys a_{i^*}. Furthermore, message X^* is chosen by the reduction $\mathcal{B}$ as B_{sID^*}, whereby Y^* is chosen by $\mathcal{A}$. In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_2^* = \mathsf{DH}(A_{r^*}, X^*) = \mathsf{DH}(A_{r^*}, B_{\text{sID}^*})$ which is a correct solution for CorrAGapCDH.

Attack (16.). The test session is of type “Re” and $\mathcal{A}$ has queried the responder's long-term secret keys a_{r^*}. Furthermore, message Y^* is chosen by the reduction $\mathcal{B}$ as B_{sID^*}, whereby X^* is chosen by $\mathcal{A}$. In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_1^* = \mathsf{DH}(A_{i^*}, Y^*) = \mathsf{DH}(A_{i^*}, B_{\text{sID}^*})$ which is a correct solution for CorrAGapCDH.

The number of queries to the DDH oracle is upper bounded by $3 \cdot Q_{\mathsf{H}}$. Thus,

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_0^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right| \leq \mathrm{Adv}^{\mathsf{CorrAGapCDH}}_{N+S,\, N,\, 3Q_{\mathsf{H}}}(\mathcal{B}) \ .$$

GAME G_2. In game G_2, the challenge oracle TEST always outputs a uniformly random key, independent from the bit b (Fig. 15, line 54). As $\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 1] = \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 1]$, it holds that

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| = \frac{1}{2}\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right| .$$

We construct adversary $\mathcal{C}$ against S-GapCDH in Figure 17. $\mathcal{C}$ gets as input S group elements $(B_1, ..., B_S)$ and has access to oracle DDH. The long-term key pairs are chosen by $\mathcal{C}$ (line 02). The input elements will be used as messages output by $\text{SESSION}_{\mathsf{I}}$ and $\text{SESSION}_{\mathsf{R}}$ as in adversary $\mathcal{B}$ described before. Again, we use a flag f to identify correct queries and thus reduce the number of queries to DDH. Whenever $\mathcal{A}$ calls DER_{I} or DER_{R}, $\mathcal{C}$ first computes the two Diffie-Hellman tuples which use the long-term secret keys (lines 19, 37). For the third one, it checks the list of random oracle queries for an entry with $f = 1$ (lines 20, 38) and outputs the corresponding session key. If this is not the case, it checks if there is an entry with an unknown Diffie-Hellman tuple (lines 22, 40) or chooses a session key uniformly at random (lines 25, 43) and adds such an entry to the list. If $\mathcal{A}$ issues a random oracle query which has not been asked before, $\mathcal{C}$ checks if the third Diffie-Hellman tuple is correct using the DDH oracle (line 59). In this case, it sets the flag f to 1. Furthermore, if there is an entry with an unknown value, it updates the entry with the correct value (line 61) and outputs the corresponding key. Otherwise, f is set to 0. $\mathcal{C}$ chooses a key uniformly at random (line 66), adds an entry with f to the list and outputs the key.

The TEST oracle outputs a random key for all queries if $b = 1$ (line 55). If $b = 0$, it outputs a random key for all those test session that will not have a matching session (line 54), and the real session key otherwise (line 52). As a particular (test) session cannot be recreated, the only way to distinguish the two games is to query the random oracle on the correct input for a test session which is in the set $\mathcal{P}$ of potential matching sessions. Let $\text{sID}^* \in \mathcal{P}$ be such a test session and $\mathsf{H}[A_{i^*}, A_{r^*}, X^*, Y^*, Z_1^*, Z_2^*, Z_3^*, 1] = \text{sKey}[\text{sID}^*]$ be the corresponding entry for such a test session in the list of hash queries. At this point, it does not matter whether the adversary completes the potential matching session or not. What matters is that both messages X^* and Y^* are chosen by the reduction $\mathcal{B}$ as B_{sID^*} and $B_{\text{sID}'}$. We assume that the adversary may query both long-term keys thus covering attacks (1.)+(2.) of Table 2. In order to distinguish the session key, $\mathcal{A}$ has to compute $Z_3^* = \mathsf{DH}(X^*, Y^*) = \mathsf{DH}(B_{\text{sID}^*}, B_{\text{sID}'})$ which is a correct solution $C \in \mathsf{Win}$ to the S-GapCDH problem.

The number of queries to the DDH oracle is upper bounded by Q_{H}. Thus,

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right| \leq \mathrm{Adv}^{\mathsf{GapCDH}}_{S,\, Q_{\mathsf{H}}}(\mathcal{C}) \ .$$

Finally, the output of the TEST oracle in G_2 is independent of the bit b, so we have

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2} \ .$$

Collecting the probabilities yields the bound stated in Theorem 3. □

$\mathcal{C}^{\text{DDH}}(B_1, ..., B_S)$
00 $\text{cnt}_\text{P} \coloneqq N$
01 **for** $n \in [N]$
02 $\quad a_n \xleftarrow{\$} \mathbb{Z}_p$; $A_n \coloneqq g^{a_n}$
03 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) \coloneqq (A_n, a_n)$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
06 **for** $\text{sID}^* \in \mathcal{S}$
07 $\quad$ **if** $\text{FRESH}(\text{sID}^*) = \textbf{false}$
08 $\quad\quad$ **return** b
09 $\quad$ **if** $\text{VALID}(\text{sID}^*) = \textbf{false}$
10 $\quad\quad$ **return** b
11 **return** $C \in \mathsf{Win}$ (see text)

$\text{DER}_\mathsf{R}(\text{sID} \in [\text{cnt}_\text{S}], X)$
12 **if** $\text{sKey}[\text{sID}] \neq \bot$ **or** $\text{type}[\text{sID}] \neq$ "Re"
13 $\quad$ **return** $\bot$
14 $(i, r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
15 $Y \coloneqq R[\text{sID}]$
16 $\text{ctxt} \coloneqq (A_i, A_r, X, Y)$
17 **if** $\exists \text{sID}'$ s. t. $(\text{type}[\text{sID}'], I[\text{sID}']) = (\text{"In"}, X)$
18 $\quad \mathcal{P} \coloneqq \mathcal{P} \cup \{\text{sID}\}$
19 $(Z_1, Z_2) \coloneqq (Y^{a_i}, X^{a_r})$
20 **if** $\exists Z_3$ s. t. $\mathsf{H}[\text{ctxt}, Z_1, Z_2, Z_3, 1] = K$
21 $\quad \text{sKey}[\text{sID}] \coloneqq K$
22 **else if** $\mathsf{H}[\text{ctxt}, Z_1, Z_2, \bot, \bot] = K$
23 $\quad \text{sKey}[\text{sID}] \coloneqq K$
24 **else**
25 $\quad K \xleftarrow{\$} \mathcal{K}$
26 $\quad \mathsf{H}[\text{ctxt}, Z_1, Z_2, \bot, \bot] \coloneqq K$
27 $\quad \text{sKey}[\text{sID}] \coloneqq K$
28 $(I[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (X, K)$
29 **return** ε

$\text{DER}_\text{I}(\text{sID} \in [\text{cnt}_\text{S}], Y)$
30 **if** $\text{sKey}[\text{sID}] \neq \bot$ **or** $\text{type}[\text{sID}] \neq$ "In"
31 $\quad$ **return** $\bot$
32 $(i, r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
33 $X \coloneqq I[\text{sID}]$
34 $\text{ctxt} \coloneqq (A_i, A_r, X, Y)$
35 **if** $\exists \text{sID}'$ s. t. $(\text{type}[\text{sID}'], R[\text{sID}']) = (\text{"Re"}, Y)$
36 $\quad \mathcal{P} \coloneqq \mathcal{P} \cup \{\text{sID}\}$
37 $(Z_1, Z_2) \coloneqq (Y^{a_i}, X^{a_r})$
38 **if** $\exists Z_3$ s. t. $\mathsf{H}[\text{ctxt}, Z_1, Z_2, Z_3, 1] = K$
39 $\quad \text{sKey}[\text{sID}] \coloneqq K$
40 **else if** $\mathsf{H}[\text{ctxt}, Z_1, Z_2, \bot, \bot] = K$
41 $\quad \text{sKey}[\text{sID}] \coloneqq K$
42 **else**
43 $\quad K \xleftarrow{\$} \mathcal{K}$
44 $\quad \mathsf{H}[\text{ctxt}, Z_1, Z_2, \bot, \bot] \coloneqq K$
45 $\quad \text{sKey}[\text{sID}] \coloneqq K$
46 $(R[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (Y, K)$
47 **return** ε

$\text{TEST}(\text{sID})$
48 **if** $\text{sID} \in \mathcal{S}$ **return** $\bot$
49 **if** $\text{sKey}[\text{sID}] = \bot$ **return** $\bot$
50 $\mathcal{S} \coloneqq \mathcal{S} \cup \{\text{sID}\}$
51 **if** $\text{sID} \in \mathcal{P}$
52 $\quad K_0^* \coloneqq \text{sKey}[\text{sID}]$
53 **else**
54 $\quad K_0^* \xleftarrow{\$} \mathcal{K}$
55 $K_1^* \xleftarrow{\$} \mathcal{K}$
56 **return** K_b^*

$\mathsf{H}(A_i, A_r, X, Y, Z_1, Z_2, Z_3)$
57 **if** $\mathsf{H}[A_i, A_r, X, Y, Z_1, Z_2, Z_3, \cdot\,] = K$
58 $\quad$ **return** K
59 **if** $\text{DDH}(X, Y, Z_3) = 1$
60 $\quad f \coloneqq 1$
61 $\quad$ **if** $\mathsf{H}[A_i, A_r, X, Y, Z_1, Z_2, \bot, \bot] = K$
62 $\quad\quad$ replace $(\bot, \bot)$ with (Z_3, f)
63 $\quad\quad$ **return** K
64 **else**
65 $\quad f \coloneqq 0$
66 $K \xleftarrow{\$} \mathcal{K}$
67 $\mathsf{H}[A_i, A_r, X, Y, Z_1, Z_2, Z_3, f] \coloneqq K$
68 **return** K

Figure 17: Adversary $\mathcal{C}$ against S-GapCDH for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\text{O} \coloneqq \{\text{SESSION}_\text{I}, \text{SESSION}_\mathsf{R}, \text{DER}_\text{I}, \text{DER}_\mathsf{R}, \text{REVEAL}, \text{CORRUPT}, \text{REGISTERLTK}, \text{TEST}, \mathsf{H}\}$, where SESSION_I and $\text{SESSION}_\mathsf{R}$ are defined as for adversary $\mathcal{B}$ in Figure 16. REGISTERLTK, REVEAL and CORRUPT are defined as in game G_1. Lines written in blue color highlight how $\mathcal{C}$ simulates G_1 and G_2, respectively.

Appendix B

Tightly-Secure Authenticated Key Exchange, Revisited

An extended abstract of this article appears in the proceedings of EUROCRYPT 2021. The following version is the full version of this article which is also available in the IACR ePrint archive, ia.cr/2020/1279.

Original Publication

T. Jager, E. Kiltz, D. Riepel, S. Schäge. Tightly-Secure Authenticated Key Exchange, Revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146. Springer, Heidelberg, October 2021. https://doi.org/10.1007/978-3-030-77870-5_5

Tightly-Secure Authenticated Key Exchange, Revisited

Tibor Jager[1], Eike Kiltz[2], Doreen Riepel[2], and Sven Schäge[2]

[1] Bergische Universität Wuppertal, Wuppertal, Germany
tibor.jager@uni-wuppertal.de
[2] Ruhr-Universität Bochum, Bochum, Germany
{eike.kiltz,doreen.riepel,sven.schaege}@rub.de

Abstract. We introduce new tightly-secure authenticated key exchange (AKE) protocols that are extremely efficient, yet have only a *constant* security loss and can be instantiated in the random oracle model both from the standard DDH assumption and a subgroup assumption over RSA groups. These protocols can be deployed with optimal parameters, independent of the number of users or sessions, without the need to compensate a security loss with increased parameters and thus decreased computational efficiency.

We use the standard "Single-Bit-Guess" AKE security (with forward secrecy and state corruption) requiring all challenge keys to be simultaneously pseudo-random. In contrast, most previous papers on tightly secure AKE protocols (Bader et al., TCC 2015; Gjøsteen and Jager, CRYPTO 2018; Liu et al., ASIACRYPT 2020) concentrated on a non-standard "Multi-Bit-Guess" AKE security which is known not to compose tightly with symmetric primitives to build a secure communication channel.

Our key technical contribution is a new generic approach to construct tightly-secure AKE protocols based on non-committing key encapsulation mechanisms. The resulting DDH-based protocols are considerably more efficient than all previous constructions.

Keywords: Authenticated key exchange, tightness, non-committing encryption, forward security

1 Introduction

Authenticated Key Exchange (AKE) is a fundamental cryptographic primitive with immense practical importance. The goal is to securely establish a session key between two parties in a network where an adversary can read, send, modify or delete messages and may also corrupt selected parties and sessions.

Tightness of AKE. When proving a cryptographic scheme secure, one commonly describes a security reduction which transforms an adversary $\mathcal{A}$ that breaks the cryptographic scheme into an adversary $\mathcal{B}$ that solves some underlying complexity assumption. For instance, if $\mathcal{A}$ has advantage ϵ in breaking the scheme and $\mathcal{B}$ solves the problem with advantage $\epsilon' = \epsilon/L$, then L is called the reduction's security loss. If L is constant

(and in particular independent of the number of $\mathcal{A}$'s oracle queries) and additionally the running times of $\mathcal{A}$ and $\mathcal{B}$ are roughly identical, then we say the reduction is *tight*. Especially when choosing protocol-specific system parameters, the tightness of a security proof plays an important role. In the security model for AKE the attacker can actively control all messages sent between the involved parties and is additionally allowed to reveal secret information such as a long-term secret key (by corrupting a party), or a session key. The adversary breaks security if it is able to distinguish non-revealed session keys from random.

Multi-Challenge Security definitions. The standard and well established security notion in the context of multiple challenges [BBM00, FHKP13, GHKW16, CCG+19] is "Single-Bit Guess" (SBG) security. The blueprint of a SBG security experiment is as follows. First, the experiment picks a secret random bit $b \in \{0,1\}$. Next, the adversary is allowed to make multiple (up to, say, T) challenge queries. On each challenge query, the experiment returns a "real key" if $b = 0$, and an independent "random key" if $b = 1$. The adversary wins if it can guess the challenge bit b with a probability better than $1/2$.

In AKE protocols, challenge queries are usually called test queries and non-revealed session keys can be accessed by making multiple calls to a Test oracle. If $b = 0$, a query to Test returns the real challenge key; if $b = 1$, a query to Test returns an independent random challenge key. This notation of multi-challenge SBG security for AKE was first formalized in 2019 by Cohn-Gordon et al. [CCG+19]. By conditioning on bit b, SBG security is known to be tightly equivalent to (single-bit) "Real-Or-Random" (ROR) security, where the adversary has to distinguish a real game (where all challenge keys output by Test are real) from a random game (where all challenge keys are random). Using the above equivalence, SBG security precisely captures the intuition that *all challenge keys* are simultaneously pseudo-random.

Surprisingly, in the first publication on tightly secure AKE protocols in 2015, Bader et al. [BHJ+15] defined a different and non-standard "Multi-Bit-Guess" (MBG) AKE security notion. In MBG security, the experiment picks multiple independent challenge bits $b_1, \ldots, b_T$ and, on the i-th Test query, it returns a real challenge key if $b_i = 0$ and a random challenge key if $b_i = 1$. That is, each of the T challenge keys depends on an independent challenge bit b_i. The adversary wins if it can guess correctly one of the T challenge bits b_{i^*} with a probability better than $1/2$. We are not aware of any meaningful multi-bit ROR security game that is tightly equivalent to MBG security.[1] This makes it difficult to provide a good intuition of what MBG security tries to model.

Choosing a Meaningful Security Model for AKE. SBG and MBG security are asymptotically equivalent but only imply each other with a security loss of T, the total number of Test queries. Hence, when considering tightness, one has to carefully choose a meaningful security model.

[1] If one tries to apply a similar conditioning argument as in the single-bit case, MBG can be shown equivalent to a ROR-type security experiment where in the real game ($b_{i^*} = 0$) the i^*-th challenge key output by Test is real and in the random game ($b_{i^*} = 1$) it is random. However, the remaining $T - 1$ keys still depends on the random bits b_i ($i \neq i^*$): the i-th challenge key is real if $b_i = 0$ and it is random if $b_i = 1$. Hence, about one half of the challenge keys is expected to be real (the ones with $b_i = 0$) whereas the other half is random, and the adversary does not have any information on them.

First off, as already pointed out, SBG security is the standard and well established security notion in the context of multiple challenges [BBM00, FHKP13, GHKW16, CCG+19]. Cohn-Gordon et al. [CCG+19, Section 3] already pointed out that, in the AKE setting, SBG security tightly composes with symmetric primitives, whereas MBG security doesn't. Let us elaborate. AKE is not intended to be used as a stand-alone primitive. Rather, it is naturally composed with symmetric primitives to establish a secure channel [BFWW11, JKSS12], for example to encrypt (e.g., using AES) a message with the session key. Since SBG security is tightly equivalent to ROR security, it offers precisely the right security interface to switch *all challenge keys at once* from real to random. This step allows to infer the privacy of the encrypted messages from the security properties of the symmetric primitive. MBG security, on the other hand, does not have a meaningful ROR-style security, which makes it difficult to argue about the privacy of the encrypted messages without relying on a hybrid argument. In summary, in the context of tightness of AKE protocols, SBG security is a meaningful notion whereas MBG isn't.

Previous Results. Previous work on tight AKE protocols by Gjøsteen and Jager [GJ18] and Liu at al. [LLGW20] exclusively concentrated on the MBG model by Bader et al. [BHJ+15]. We now give a brief overview of existing AKE protocols in the context of tight SBG security.

- At CRYPTO 2019, Cohn-Gordon et al. [CCG+19] presented highly efficient two message AKE protocols with implicit authentication, in the style of HMQV [Kra05] and similar protocols. Their schemes achieve a loss of $O(N)$ in the SBG security model with weak forward secrecy, where N is the number of users. They also extend the impossibility results from [BJLS16] to show that a loss of $O(N)$ is unavoidable for many natural protocols (including HMQV [Kra05], NAXOS [LLM07], Kudla-Paterson [KP05], KEA+ [LM06]) with respect to typical cryptographic security proofs (so-called simple reductions). Furthermore, since their protocol does not feature explicit authentication, a well-known impossibility result applies [Kra05, BG11, Sch15] and their protocol cannot achieve full forward security.
- Diemert and Jager [DJ20] and independently Davis and Günther [DG20] considered the three message TLS 1.3 handshake AKE protocol with explicit authentication. Its design follows the standard "1×KEM+2×SIG" (aka. signed Diffie-Hellman) AKE approach [CK01, CF12, GJ18, DJ20, DG20, LLGW20]. TLS 1.3, when instantiated with standardized signatures (e.g., RSA-PSS, RSA-PKCS #1 v1.5, ECDSA, or EdDSA), has rather non-tight SBG security with full forward security. But when instantiated with tightly secure signatures in the multi-user setting with adaptive corruptions [BHJ+15], then SBG security of TLS 1.3 actually becomes tight. Since the TLS 1.3 protocol contains two signatures, the inefficiency of currently known tightly secure signature schemes [BHJ+15, GJ18] makes the resulting TLS instantiation very impractical.

1.1 The Difficulty of Constructing Tightly Secure AKE

Security models for authenticated key exchange are extremely complex, as they consider very strong adversaries that may modify, drop, or inject messages. Furthermore, usually

an adversary may adaptively corrupt users' long-term secrets via Corrupt-queries, session keys via Reveal-queries, and sometimes even ephemeral states of sessions via Rev-State-queries. Security is formalized with multiple Test queries, where the adversary specifies a session, receives back a real key or a random key, and has to distinguish these. This complexity makes achieving tight security challenging, particularly because all the following difficulties must be tackled simultaneously.

The "commitment problem". As explained in more detail in [GJ18], this problem is the reason why nearly all security proofs of classical key exchange protocols have a quadratic security loss. Essentially, the problem is that most AKE protocols have security proofs where a reduction can only extract a solution to a computationally hard problem if an instance of the problem is embedded into the protocol messages of the Tested sessions, but at the same time the reduction is not able to answer Reveal queries for such sessions. The standard way to resolve this is to let the reduction guess the Tested session, and to embed an instance of a computationally hard problem only there. However, this incurs a significant security loss. A tight reduction has to be able to respond to *both* Test and Reveal queries for *every* session.

The problem of long-term key reveals. A Corrupt query in typical AKE security models enables the adversary to obtain the long-term key of certain users. If we want to avoid a security loss that results from guessing corrupted and non-corrupted parties, then we must be able to construct a reduction that "knows" valid-looking long-term keys for all users throughout the security experiment. However, this is a major difficulty, for instance, in protocols where the long-term keys are key pairs for a digital signature scheme. The difficulty is that in the security proof we would have to describe a reduction that is able to extract a solution to a computationally hard problem from a forged signature, even though it "knows" the signing key and thus is able to compute a valid signature itself. Hence, in order to obtain a tightly-secure AKE protocol, one needs to devise a way such that a reduction always knows all secret keys, yet is able to argue that an adversary is, e.g., not able to forge signatures.

In order to resolve this issue, previous works [BHJ+15, GJ18] constructed signature schemes based on non-interactive OR-proof systems, which enable a reduction to "know" one out of two signing keys. It is argued that the adversary will forge a signature with respect to the other, unknown key with sufficiently high probability. However, these signature schemes are much less efficient than classical ones, and thus impose a performance penalty on the protocols.

The problem of ephemeral state reveals. Yet another difficulty arises when the security model allows ephemeral state reveals. Previous works on tightly-secure AKE did not consider this very strong security notion at all, therefore we face (and solve) this problem for the first time. From a high-level perspective, the issue is similar to the long-term key reveal problem, except that ephemeral states are considered. In order to achieve tightness, the reduction must be able to output valid-looking states for all sessions. Note that this includes even Tested sessions, where ephemeral states may be revealed when parties are not corrupted.

1.2 Main Contributions

Summarizing the previous paragraphs, we can formulate the following natural questions related to tightly secure AKE:

Q1: Do there exist implicitly authenticated two-message AKEs with tight SBG security, state reveals, and weak forward security?

Q2: Do there exist explicitly authenticated two-message AKEs with tight SBG security, state reveals, and full forward security, with *one single* signature?

In this work, we answer the two questions to the positive. Following [BR94, CCG+19], we consider SBG security, allowing adaptive corruptions of long-term secrets, adaptive reveals of session keys, and multiple adaptive Test queries. Our model also captures (weak and full) forward security (FS), and prevents key-compromise impersonation and reflection attacks. In comparison to prior work on tightly-secure key exchange [BHJ+15, GJ18, CCG+19, DJ20, DG20], we consider a model which additionally allows to reveal some internal state information.

Our DDH-Based AKE Protocols. Our two protocols instantiated from DDH are given in Figure 1. $\mathsf{AKE}_{\mathsf{wFS},\mathsf{DDH}}$ is an implicitly-authenticated two-message protocol $\mathsf{AKE}_{\mathsf{wFS},\mathsf{DDH}}$ in the sense of [Kra05]. It requires the exchange of only five group elements in total, and thus is the first efficient implicitly-authenticated protocol with weak FS that achieves full tightness.

Our second protocol $\mathsf{AKE}_{\mathsf{FS},\mathsf{DDH}}$ achieves full FS. Instead of using the standard "1×KEM+2×SIG" approach, it replaces one of the signatures with a more efficient MAC and an additional KEM ciphertext, which yields a "2×KEM+1×SIG+1×MAC" construction. When instantiated at "128-bit security" with the most efficient tightly-secure signatures of [GJ18],[2] the communication complexity is 448 bytes, again with ephemeral state reveals. In comparison, the previously most efficient tightly and fully forward-secure protocol with SBG security TLS^* (which is TLS 1.3 instantiated with the tightly-secure signature of [GJ18]) requires three messages, the transmission of 704 bytes and does not allow state reveals. See Figure 2 for a comparison of our protocols with previous works. Note that the communication bottleneck in all full FS protocols is the number of signatures. For completeness the figure also list previous protocols with tight MBG security [GJ18, LLGW20].

Generic constructions of AKE from NCKE. Our main technical tool is a new approach to achieve a tight reduction for authenticated key exchange protocols. Our starting point is an extension of (receiver) non-committing encryption (NCE) [CFGN96, Nie02] to *non-committing key encapsulation (NCKE) in the multi-user setting with corruptions*. We construct an NCKE scheme in the random oracle model from any smooth projective hash proof system (HPS) [CS02]. If the HPS' subset membership problem (SMP) is hard in the multi-instance setting, then the NCKE scheme is also tightly secure in our multi-user setting. We provide two such HPS, one from the DDH

[2] The signatures of [GJ18] consist of 2 group elements, 4 elements in $\mathbb{Z}_p$ and 2 hashes in $\{0,1\}^\kappa$. At "128-bit security" this corresponds to 256 bytes per signature.

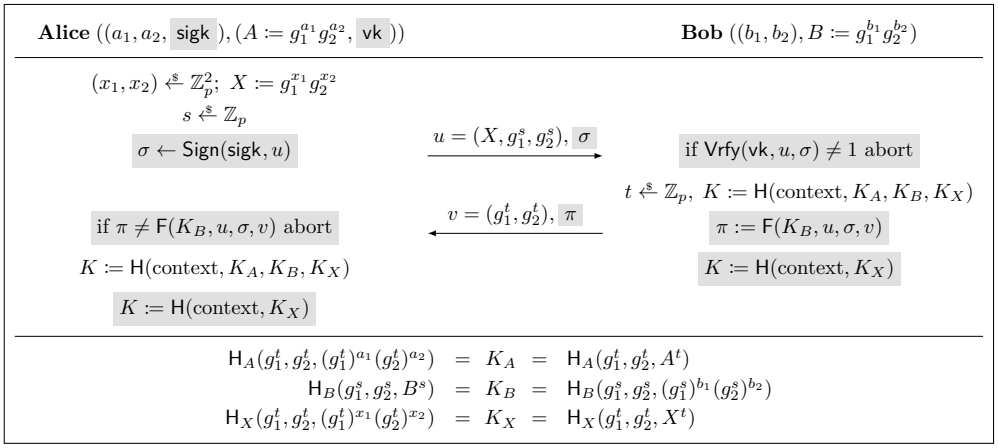

Figure 1: The two message protocols $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ (without the gray boxes) and $\mathsf{AKE}_{\mathsf{FS,DDH}}$ (including the gray boxes), where K is the resulting session key. We define context $\coloneqq (A, B, X, \mathsf{vk}, g_1^s, g_2^s, g_1^t, g_2^t, \sigma, \pi)$. $\mathsf{H}, \mathsf{H}_A, \mathsf{H}_B, \mathsf{H}_X$ and F are hash functions.

assumption, and another one from a subgroup assumption over groups of unknown order. The construction allows us to address the commitment problem described above.

We give a generic construction of an implicitly authenticated two-message AKE protocol $\mathsf{AKE}_{\mathsf{wFS}}$ with weak forward security from any NCKE scheme, whose security is tightly based on the multi-user security of the underlying NCKE scheme. Furthermore, we give a generic construction of an explicitly authenticated two-message AKE protocol $\mathsf{AKE}_{\mathsf{FS}}$ with perfect forward security by adding a tightly-secure signature scheme and a message authentication code (MAC) to our first construction, see Figure 3. Thus, we require only a single signature which is particularly useful for tightly-secure key exchange, because known constructions of suitable tightly-secure signature schemes [BHJ+15, GJ18] have relatively large signatures and replacing one signature with a MAC significantly improves the computational efficiency and communication complexity of the protocol.[3]

All these generic constructions leverage NCKE in order to resolve the technical difficulties in constructing tightly-secure AKE protocols described before.

Handling Ephemeral State Reveals. Our protocols are secure against ephemeral state reveals. We construct the first tightly-secure protocols to achieve this. Note that this requires us to deal with the situation that the reduction must "know" valid ephemeral states for *all* sessions, even tested sessions. To this end, we encrypt the state information with a symmetric long-term key. An adversary now needs to query both long-term secret key and ephemeral state to reveal the secret state information, similarly to the approach used in the NAXOS protocol [LLM07]. While the idea of achieving security against ephemeral state reveals by relying on the security of long-term keys was used before [LLM07, BJS15, Yon12, FSXY12], the approach to simply encrypt the state is new. It avoids the expensive re-computation of protocol messages required in

[3] [LSY+14] showed how to generically avoid signatures in forward-secure AKE protocols, but at the cost of additional messages.

Protocol	Comm. $(\mathbb{G}, \{0,1\}^\kappa, \mathrm{Sig})$	Bytes	#Msg.	Assump.	Model	State Reveal	Sec. Loss
Protocols with full forward security and explicit authentication							
TLS* [DJ21, DG21]	$(2,4,2)$	704	3	Strong-DH + DDH	SBG	no	$O(1)$
GJ [GJ18]	$(2,1,2)$	608	3	DDH	MBG	no	$O(1)$
LLGW [LLGW20]	$(3,0,2)$	608	2	DDH	MBG	no	$O(1)$
$\mathsf{AKE}_{\mathsf{FS,DDH}}$ (Fig. 1)	$(5,1,1)$	448	2	DDH	SBG	yes	$O(1)$
Protocols with weak forward security and implicit authentication							
HMQV [Kra05]	$(2,0,0)$	64	2	CDH	SBG	yes	$O(TN^2\ell^2)$
CCGJJ [CCG+19]	$(2,0,0)$	64	2	Strong-DH	SBG	no	$O(N)$
$\mathsf{CCGJJ}_{\mathrm{Twin}}$ [CCG+19]	$(3,0,0)$	96	2	CDH	SBG	no	$O(N)$
$\mathsf{AKE}_{\mathsf{wFS,DDH}}$ (Fig. 1)	$(5,0,0)$	160	2	DDH	SBG	yes	$O(1)$

Figure 2: Comparison of AKE protocols over a group $\mathbb{G}$, where N refers to the number of parties, ℓ to the number of sessions per party and T is the number of test queries. TLS* refers to the TLS 1.3 handshake, instantiated with the tightly-secure signatures of [GJ18]. The column **Comm.** counts the communication complexity of the protocols in terms of the number of group elements, hashes, and signatures. The column **Model** lists the AKE security model and distinguishes between multi-bit guessing (MBG) and the single-bit-guessing (SBG) security.

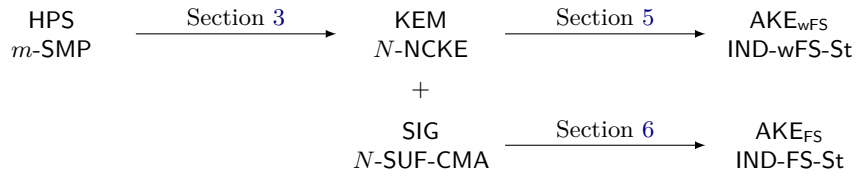

Figure 3: Overview of our transformations, where N is the maximum number of users in the NCKE security game and in the SUF-CMA security game. The subset membership problem of HPS is m-fold for $m = N \cdot q$, where q is the maximum number of challenge queries in the NCKE security game.

prior generic approaches, which makes it particularly efficient. Also, previous work did not focus on tightness and it is unclear if a tight proof can be achieved in an even stronger security model which requires to reveal the randomness.

Our approach does not work generically, e.g., it cannot be applied to the protocols in [GJ18, CCG+19], so we have to design our protocols such that they are compatible. This is due to the fact that in both works, the state is a secret DH exponent which is implicitly determined by rerandomizing the CDH (or DDH) challenge and then is embedded in multiple sessions. Thus, the reduction is able to extract the solution independently of which session is the test session, but it also does not know any of the secret exponents, which the adversary could reveal for non-test sessions.

1.3 Related Work and Open Problems

Concurrent and independent work of Liu et al. [LLGW20] also proposed a tightly secure 2-message AKE with full forward security. Compared to our protocols, they do not consider state reveal attacks and their proofs only hold in the MBG security model. Their AKE construction LLGW follows the well known 1×KEM+2×SIG approach, meaning that even neglecting the issues with the MBG security model, it is still considerably less efficient than ours (cf. Fig. 2). The main novelty of [LLGW20] is the new KEM security notion of (multi-bit) "IND-mCPA with adaptive reveals" that gives them the handle to prove tight MBG security. It is a natural question whether this KEM security notion can be adapted to a single-bit notion such that the resulting AKE protocol achieves tight SBG (rather than MBG) security. This is in particular interesting since IND-mCPA KEMs with adaptive reveals can be instantiated in the standard model, whereas our NCKE notion seem to inherently rely on random oracles. More concretely this raises the question whether (variants of) [LLGW20] can also be proved in the SBG model, without relying on random oracles.

2 Preliminaries

For an integer n, $[n]$ denotes the set $\{1, ..., n\}$. For a set S, $s \xleftarrow{\$} S$ denotes that s is sampled uniformly and independently at random from S. $y \leftarrow \mathcal{A}(x_1, x_2, ...)$ denotes that on input $x_1, x_2, ...$ the probabilistic algorithm $\mathcal{A}$ returns y. $\mathcal{A}^{\mathrm{O}}$ denotes that algorithm $\mathcal{A}$ has access to oracle O. We will use code-based games as introduced in [Sho04]. An adversary is a probabilistic algorithm. $\Pr[G^{\mathcal{A}} \Rightarrow 1]$ denotes the probability that the final output $G^{\mathcal{A}}$ of game G running adversary $\mathcal{A}$ is 1.

3 Multi-Receiver Non-Committing Key Encapsulation

In this section, we introduce Multi-Receiver Non-Committing Key Encapsulation (NCKE). We will use this concept to resolve the "commitment problem" described in the introduction, which often makes proofs for multi-party protocols with adaptive corruptions non-tight, as for example AKE protocols.

Syntax. A key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ consists of three algorithms. The key generation algorithm Gen outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk is the public key and sk the secret key. The encapsulation algorithm inputs a public key pk and outputs a ciphertext c and a key K from the key space $\mathcal{K}$, where c is called an encapsulation of K. The deterministic decapsulation algorithm inputs the secret key sk and a ciphertext c and outputs K.

By μ we denote the *collision probability* of the key generation algorithm. In particular,

$$\Pr[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}, (\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Gen} : \mathsf{pk} = \mathsf{pk}'] \leq 2^{-\mu} \ .$$

We denote the *min-entropy* of the encapsulation algorithm Encaps by

$$\gamma(\mathsf{pk}) := -\log \max_{c \in \mathcal{C}} \Pr[c = \mathsf{Encaps}(\mathsf{pk})] \ .$$

We say KEM is γ-spread if for all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen} : \gamma(\mathsf{pk}) \geq \gamma$. This implies that for all $c \in \mathcal{C}$ we have $\Pr[c = \mathsf{Encaps}(\mathsf{pk})] \leq 2^{-\gamma}$.

Security. Following [Nie02], we introduce a security definition of Multi-Receiver Non-Committing Key Encapsulation (NCKE) for a key encapsulation mechanism KEM in the random oracle model, i. e., the KEM algorithms have access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$, indicated by $\mathsf{Encaps}^{\mathsf{H}}$. Our definition is relative to a simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$. The simulated key generation algorithm SimGen generates a key pair $(\mathsf{pk}, \mathsf{sk})$. The simulated encapsulation algorithm $\mathsf{SimEncaps}$ takes both the public and private key and outputs a ciphertext c. The simulated hash algorithm $\mathsf{SimHash}$ inputs the key pair as well as three sets (used for bookkeeping) and deterministically computes a simulated hash value.

We define the two games $\mathsf{NCKE}_{\mathsf{real}}$ and $\mathsf{NCKE}_{\mathsf{sim}}$ in Figure 4 where we consider N receivers each holding a key pair $(\mathsf{pk}_n, \mathsf{sk}_n)$. In the $\mathsf{NCKE}_{\mathsf{real}}$ game, the original Encaps algorithm is used. We give each user an individual hash function H_n such that keys are computed independently. (In general, this can be implemented by using the user's public key and identity as input to the hash function as well, where collisions have to be considered.) In the $\mathsf{NCKE}_{\mathsf{sim}}$ game, the $\mathsf{SimEncaps}$ algorithm is used to compute the ciphertexts. Keys are chosen uniformly at random. The adversary may also adaptively corrupt some receivers. We require that ciphertexts of corrupted receivers always decapsulate to the key output by Encaps, which is modeled by the $\mathsf{SimHash}$ algorithm. Therefore, if the receiver is corrupted, the algorithm takes sets $\mathcal{CK}$, $\mathcal{D}$ and $\mathcal{H}$, where the first one stores all challenge ciphertexts and keys output to the adversary, the second one stores all decapsulation queries and the third one stores all hash queries which have been issued so far. Thus, the $\mathsf{SimHash}$ algorithm can answer future queries based on everything that is known to the adversary. If the receiver is not corrupted, set $\mathcal{C}$ is used instead of $\mathcal{CK}$. This set stores only challenge ciphertexts and thus a hash value is computed independently of previous challenge keys.

The goal of an adversary $\mathcal{A}$ is to distinguish between the real KEM algorithms used in game $\mathsf{NCKE}_{\mathsf{real}}$ and the simulated algorithms used in game $\mathsf{NCKE}_{\mathsf{sim}}$. This is captured in Definition 1. Note that the non-committing property is due to the $\mathsf{SimHash}$ algorithm. In particular, the $\mathsf{SimHash}$ algorithm ensures that a (uniformly random) challenge key can be explained by the corresponding ciphertext generated by $\mathsf{SimEncaps}$ as soon as the receiver is corrupted.

Definition 1 (N-Receiver Non-Committing Key Encapsulation)**.** *We define games* $\mathsf{NCKE}_{\mathsf{real}}$ *and* $\mathsf{NCKE}_{\mathsf{sim}}$ *as in Figure 4, where N is the number of users. The simulator* $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$ *is defined relative to* KEM *and is used in* $\mathsf{NCKE}_{\mathsf{sim}}$*. The advantage of an adversary $\mathcal{A}$ against* KEM *and* Sim *is defined as*

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{A}) := \left| \Pr[\mathsf{NCKE}^{\mathcal{A}}_{\mathsf{real}} \Rightarrow 1] - \Pr[\mathsf{NCKE}^{\mathcal{A}}_{\mathsf{sim}} \Rightarrow 1] \right| \ .$$

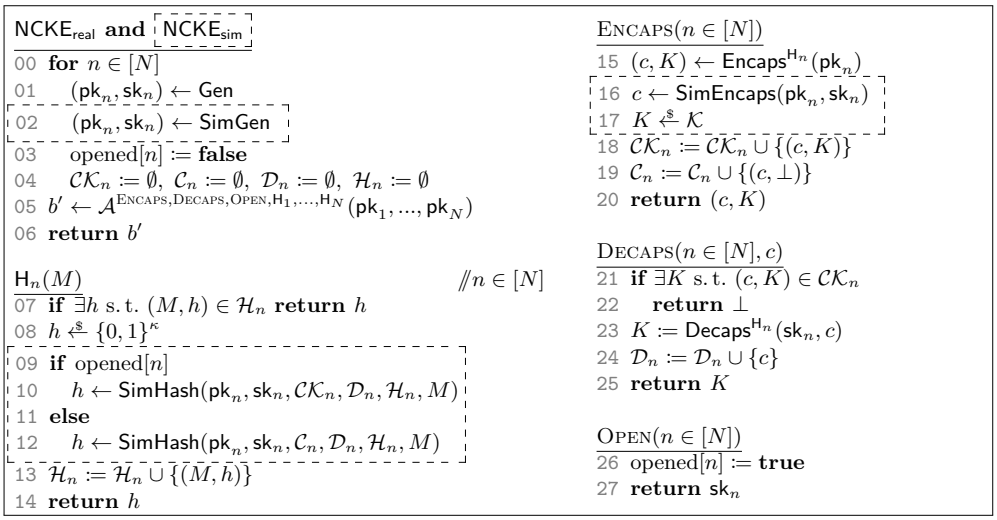

Figure 4: Real and simulated game for N-receiver non-committing key encapsulation in the random oracle model.

When we write NCKE, we mean NCKE-CCA, where the adversary is allowed to access a decapsulation oracle. Sometimes we will explicitly write NCKE-CCA to differentiate from NCKE-CPA, where the adversary cannot issue decapsulation queries.

We stress that compared to the standard definition of non-committing encryption in the random oracle model (e.g., [Nie02]), Definition 1 is for KEMs (rather than encryption), only considers receiver corruptions (rather than sender and receiver corruptions), and considers multiple receivers (rather than one single receiver).

Instantiations from Hash Proof Systems. We recall the definition of hash proof systems by Cramer and Shoup [CS02] and properties defined in [KPSY09].

Smooth Projective Hashing. Let $\mathcal{Y}$ and $\mathcal{Z}$ be sets and $\mathcal{X} \subset \mathcal{Y}$ a language. Let $\Lambda_{\mathsf{sk}} : \mathcal{Y} \to \mathcal{Z}$ be a hash function indexed with $\mathsf{sk} \in \mathcal{SK}$, where $\mathcal{SK}$ is a set. A hash function Λ_{sk} is projective if there exists a projection $\mu : \mathcal{SK} \to \mathcal{PK}$ such that $\mu(\mathsf{sk}) \in \mathcal{PK}$ defines the action of Λ_{sk} over $\mathcal{X}$. In particular, for every $c \in \mathcal{X}$, $Z = \Lambda_{\mathsf{sk}}(c)$ is uniquely determined by $\mu(\mathsf{sk})$ and c. However, there is no guarantee for $c \in \mathcal{Y} \setminus \mathcal{X}$ and it may not be possible to compute $\Lambda_{\mathsf{sk}}(c)$ from $\mu(\mathsf{sk})$ and C. A projective hash function is k-entropic if for all $c \in \mathcal{Y} \setminus \mathcal{X}$ it holds that $H_\infty(\Lambda_{\mathsf{sk}}(c) \mid \mathsf{pk}) \geq k$, where $\mathsf{pk} = \mu(\mathsf{sk})$ for $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$.

Hash Proof System. A hash proof system $\mathsf{HPS} = (\mathsf{Par}, \mathsf{Priv}, \mathsf{Pub})$ consists of three algorithms. The randomized algorithm Par generates parametrized instances of $\mathsf{par} = (group, \mathcal{Z}, \mathcal{Y}, \mathcal{X}, \mathcal{PK}, \mathcal{SK}, \Lambda_{(\cdot)} : \mathcal{Y} \to \mathcal{Z}, \mu : \mathcal{SK} \to \mathcal{PK})$, where $group$ may contain additional structural parameters. The deterministic public evaluation algorithm Pub inputs the projection key $\mathsf{pk} = \mu(\mathsf{sk})$, $c \in \mathcal{X}$ and a witness r of the fact that $c \in \mathcal{X}$ and returns $Z = \Lambda_{\mathsf{sk}}(c)$. The deterministic private evaluation algorithm Priv takes $\mathsf{sk} \in \mathcal{SK}$ and returns $\Lambda_{\mathsf{sk}}(c)$ without knowing a witness. Furthermore, we assume that μ is efficiently computable and that there are efficient algorithms for sampling $c \in \mathcal{X}$

$\underline{\mathsf{Gen}(\mathsf{par})}$	$\underline{\mathsf{Encaps}^{\mathsf{H}}(\mathsf{pk})}$	$\underline{\mathsf{Decaps}^{\mathsf{H}}(\mathsf{sk}, c)}$
00 $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$	03 $c \xleftarrow{\$} \mathcal{X}$ with witness r	06 $K := \mathsf{H}(c, \mathsf{Priv}(\mathsf{sk}, c))$
01 $\mathsf{pk} := \mu(\mathsf{sk})$	04 $K := \mathsf{H}(c, \mathsf{Pub}(\mathsf{pk}, c, r))$	07 **return** K
02 **return** $(\mathsf{pk}, \mathsf{sk})$	05 **return** (c, K)	

Figure 5: Key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$.

$\underline{\mathsf{SimEncaps}(\mathsf{pk}, \mathsf{sk})}$	$\underline{\mathsf{SimHash}(\mathsf{pk}, \mathsf{sk}, \mathcal{E}, \mathcal{D}, \mathcal{H}, M)}$
00 $c \xleftarrow{\$} \mathcal{Y} \setminus \mathcal{X}$	02 $(c, Z) := M$
01 **return** c	03 **if** $\exists K$ s. t. $(c, K) \in \mathcal{E}$ **and** $\mathsf{Priv}(\mathsf{sk}, c) = Z$
	04 $\quad h := K$
	05 **else**
	06 $\quad h \xleftarrow{\$} \{0,1\}^{\kappa}$
	07 **return** h

Figure 6: Simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$ for KEM, where $\mathsf{SimGen} = \mathsf{Gen}$. List $\mathcal{E}$ is either $\mathcal{CK}$ or $\mathcal{C}$.

uniformly together with a witness r, sampling $c \in \mathcal{Y}$ uniformly and checking membership in $\mathcal{Y}$.

(m-FOLD) SUBSET MEMBERSHIP PROBLEM. We define the m-fold subset membership problem for HPS which requires to distinguish m ciphertexts uniformly drawn from $\mathcal{X}$ from m ciphertexts uniformly drawn from $\mathcal{Y} \setminus \mathcal{X}$. The advantage of an adversary $\mathcal{A}$ is defined as

$$\mathrm{Adv}^{m-\mathsf{SM}}_{\mathsf{HPS}}(\mathcal{A}) := |\Pr[\mathcal{A}(\mathcal{Y}, \mathcal{X}, c_1, ..., c_m) \Rightarrow 1] - \Pr[\mathcal{A}(\mathcal{Y}, \mathcal{X}, c'_1, ..., c'_m) \Rightarrow 1]| \quad ,$$

where $c_1, ..., c_m \xleftarrow{\$} \mathcal{X}$ and $c'_1, ..., c'_m \xleftarrow{\$} \mathcal{Y} \setminus \mathcal{X}$.

N-RECEIVER NCKE FROM HPS. We use a k-entropic hash proof system $\mathsf{HPS} = (\mathsf{Par}, \mathsf{Pub}, \mathsf{Priv})$ with m-fold subset membership problem and a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\kappa}$ in order to construct a key encapsulation algorithm KEM and a simulator Sim as shown in Figures 5 and 6. The encapsulation algorithm Encaps samples an element c from $\mathcal{X}$ and a witness r. It runs the public evaluation algorithm and computes the key K as $\mathsf{H}(c, \mathsf{Pub}(\mathsf{pk}, c, r))$. The decapsulation algorithm Decaps uses the result of the private evaluation algorithm Priv as input to H to compute K. Instead of sampling an element from $\mathcal{X}$, the SimEncaps algorithm samples an element c uniformly at random from $\mathcal{Y} \setminus \mathcal{X}$ and only returns c. The SimHash algorithm takes as input three sets $\mathcal{E}, \mathcal{D}, \mathcal{H}$, where $\mathcal{E} \in \{\mathcal{C}, \mathcal{CK}\}$, and the value $M = (c, Z)$ chosen by the adversary. If there exists a key K such that $(c, K) \in \mathcal{E}$ (note that for $\mathcal{E} = \mathcal{C}$ this will never be true) and the adversary's input to H satisfies $\mathsf{Priv}(\mathsf{sk}, c) = Z$, then the output value h is set to K.

Theorem 1 (k-entropic HPS with $(N \cdot q_E)$-fold SMP $\Rightarrow$ N-NCKE)**.** *For any N-NCKE adversary $\mathcal{A}$ against* KEM *and* Sim *that issues at most q_E queries to* ENCAPS*, q_D queries to* DECAPS *and at most q_{H} queries to each random oracle H_n for $n \in [N]$, there exists an adversary $\mathcal{B}$ against the $(N \cdot q_E)$-fold subset membership problem of* HPS *such that*

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{A}) \leq \mathrm{Adv}^{(N \cdot q_E)\text{-}\mathsf{SM}}_{\mathsf{HPS}}(\mathcal{B}) + \frac{N \cdot q_E \cdot q_{\mathsf{H}}}{2^k} + \frac{N \cdot q_E \cdot q_D}{|\mathcal{Y} \setminus \mathcal{X}|} \quad ,$$

where HPS *is* k*-entropic,* $\mathcal{Y}$ *is the set of all ciphertexts and* $\mathcal{X}$ *is the set of valid ciphertexts.*

Proof. Let $\mathcal{A}$ be an adversary against KEM and Sim in the NCKE games. Consider the sequence of games in Figure 7.

GAME G_0. This is the original $\mathsf{NCKE}_{\mathsf{real}}$ game, hence

$$\Pr[G_0^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{NCKE}_{\mathsf{real}}^{\mathcal{A}} \Rightarrow 1] \ .$$

GAME G_1. In game G_1, the ENCAPS oracle is modified in a way that it uses the private evaluation algorithm to compute K in line 22. It holds that $\mathsf{Pub}(\mathsf{pk}, c, r) = \mathsf{Priv}(\mathsf{sk}, c)$. Thus, this does not change the adversary's view and

$$\Pr[G_1^{\mathcal{A}} \Rightarrow 1] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1] \ .$$

GAME G_2. The ENCAPS oracle now chooses the ciphertext from $\mathcal{Y} \setminus \mathcal{X}$ in line 17. We claim that

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathrm{Adv}_{\mathsf{HPS}}^{(N \cdot q_E)\text{-}\mathsf{SM}}(\mathcal{B}) \ . \tag{1}$$

In Figure 8, we construct adversary $\mathcal{B}$ against the $(N \cdot q_E)$-fold subset membership problem. $\mathcal{B}$ inputs sets $\mathcal{Y}, \mathcal{X}$ and ciphertexts $c_{n,k}$, where $n \in [N], k \in [q_E]$. If $\mathcal{B}$'s input values are elements from $\mathcal{X}$, then $\mathcal{B}$ perfectly simulates G_1. Otherwise, if the input elements are from $\mathcal{Y} \setminus \mathcal{X}$, $\mathcal{B}$ simulates G_2. This yields Equation 1.

GAME G_3. In game G_3, we raise flag BAD in line 19 and abort if the ENCAPS oracle chooses a ciphertext that was issued to the DECAPS oracle before. As a challenge ciphertext is chosen uniformly at random from $\mathcal{Y} \setminus \mathcal{X}$, the probability that BAD is raised for one specific challenge ciphertext is at most $q_D / |\mathcal{Y} \setminus \mathcal{X}|$. Union bound over all challenge ciphertexts yields

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathrm{BAD}] \leq \frac{N \cdot q_E \cdot q_D}{|\mathcal{Y} \setminus \mathcal{X}|} \ .$$

GAME G_4. In game G_4, we use internal hash functions H'_n to compute K in line 23. These are not directly accessible to the adversary and independent of the random oracle as long as the secret key has not been opened. However, if the adversary opens the secret key of user n, then it can simply compute the value $Z = \mathsf{Priv}(\mathsf{sk}_n, c)$ for any challenge ciphertext c of that user. This is why we have to patch the random oracle and output $\mathsf{H}'_n(c, Z)$ whenever $\mathcal{A}$ issues such a query (lines 09 and 10).

The only possibility for $\mathcal{A}$ to notice the difference is when it queries H_n on $(c, Z = \mathsf{Priv}(\mathsf{sk}_n, c))$ before sk_n is opened, where c is a ciphertext output by ENCAPS. Here, we use the fact that HPS is k-entropic and show that

$$\left|\Pr[G_4^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{N \cdot q_E \cdot q_{\mathsf{H}}}{2^k} \ .$$

Using a hybrid argument, we modify the computation of K using function H'_n independent of the random oracle as long as the adversary does not see the corresponding sk_n.

GAMES G_0-G_5

00 **for** $n \in [N]$
01 $\quad \mathsf{sk}_n \xleftarrow{\$} \mathcal{SK}$
02 $\quad \mathsf{pk}_n := \mu(\mathsf{sk}_n)$
03 $\quad \text{opened}[n] := \textbf{false}$
04 $\quad \mathcal{CK}_n := \emptyset$
05 $b' \leftarrow \mathcal{A}^{\textsc{Encaps},\textsc{Decaps},\textsc{Open},\mathsf{H}_1,\ldots,\mathsf{H}_N}(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
06 **return** b'

$\mathsf{H}_n(c, Z)$ $\quad /\!/ n \in [N]$
07 **if** $\exists h$ s. t. $(c, Z, h) \in \mathcal{H}_n$ **return** h
08 $h \xleftarrow{\$} \{0,1\}^\kappa$ $\quad /\!/ G_0$-G_3
09 **if** opened$[n]$ **and** $\exists K$ s. t. $(c, K) \in \mathcal{CK}_n$
$\quad$ **and** $\mathsf{Priv}(\mathsf{sk}_n, c) = Z$ $\quad /\!/ G_4$-G_5
10 $\quad h := H'_n(c, Z)$ $\quad /\!/ G_4$
11 $\quad h := K$ $\quad /\!/ G_5$
12 **else** $\quad /\!/ G_4$-G_5
13 $\quad h \xleftarrow{\$} \{0,1\}^n$ $\quad /\!/ G_4$-G_5
14 $\mathcal{H}_n := \mathcal{H}_n \cup \{(c, Z, h)\}$
15 **return** h

$\textsc{Encaps}(n \in [N])$
16 $c \xleftarrow{\$} \mathcal{X}$ with witness r $\quad /\!/ G_0$-G_1
17 $c \xleftarrow{\$} \mathcal{Y} \setminus \mathcal{X}$ $\quad /\!/ G_2$-G_5
18 **if** $c \in \mathcal{D}_n$ $\quad /\!/ G_3$-G_5
19 $\quad \text{BAD} := \textbf{true}$ $\quad /\!/ G_3$-G_5
20 $\quad$ **abort** $\quad /\!/ G_3$-G_5
21 $K := \mathsf{H}_n(c, \mathsf{Pub}(\mathsf{pk}, c, r))$ $\quad /\!/ G_0$
22 $K := \mathsf{H}_n(c, \mathsf{Priv}(\mathsf{sk}, c))$ $\quad /\!/ G_1$-G_3
23 $K := \mathsf{H}'_n(c, \mathsf{Priv}(\mathsf{sk}, c))$ $\quad /\!/ G_4$
24 **if** $\exists K'$ s. t. $(c, K') \in \mathcal{CK}_n$ $\quad /\!/ G_5$
25 $\quad K := K'$ $\quad /\!/ G_5$
26 **else** $\quad /\!/ G_5$
27 $\quad K \xleftarrow{\$} \{0,1\}^\kappa$ $\quad /\!/ G_5$
28 $\mathcal{CK}_n := \mathcal{CK}_n \cup \{(c, K)\}$
29 **return** (c, K)

$\textsc{Decaps}(n \in [N], c)$
30 **if** $\exists K$ s. t. $(c, K) \in \mathcal{CK}_n$
31 $\quad$ **return** $\perp$
32 $K := \mathsf{H}_n(c, \mathsf{Priv}(\mathsf{sk}_n, c))$
33 $\mathcal{D}_n := \mathcal{D}_n \cup \{c\}$
34 **return** K

$\textsc{Open}(n \in [N])$
35 $\text{opened}[n] := \textbf{true}$
36 **return** sk_n

Figure 7: Games G_0-G_5 for the proof of Theorem 1. H'_n in line 23 is used as an internal hash function which is not directly accessible to the adversary.

Therefore, we parameterize the hybrids with j, where $j \in [N \cdot q_E]$ denotes that K is replaced in the first j challenge ciphertexts.

In the following, we consider two consecutive hybrids, where the only difference is that the computation of K in the j-th query is modified. Let pk_{n^*} be the public key of the corresponding user and (c^*, K^*) the challenge ciphertext. In hybrid H_{j-1}, the adversary observes $(c^*, K^* = \mathsf{H}_{n^*}(c^*, Z^*))$, where $c^* \in \mathcal{Y} \setminus \mathcal{X}$ and $Z^* = \mathsf{Priv}(\mathsf{sk}_{n^*}, c^*)$. In hybrid H_j, the key K is computed with H'_{n^*} independent of H_{n^*} assuming that the adversary has not opened sk_{n^*}. Thus, in order to notice the difference, the adversary must query H_{n^*} on (c^*, Z^*). As for every $c \in \mathcal{Y} \setminus \mathcal{X}$, we have that $H_\infty(\mathsf{Priv}(\mathsf{sk}_{n^*}, c^*) \mid \mathsf{pk}_{n^*}) \geq k$, we can bound the probability of this event by the number of random oracle queries:

$$\left|\Pr[H_j^{\mathcal{A}} \Rightarrow 1] - \Pr[H_{j-1}^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{q_{\mathsf{H}}}{2^k} \ .$$

Note that a query (n^*, c) to $\textsc{Decaps}$, where $c \neq c^*$, will not reveal any additional information because the output of H_{n^*} will be different anyway.

$\textsc{Game}\ G_5$. In game G_5, the $\textsc{Encaps}$ oracle chooses key K uniformly at random in line 27. If the same ciphertext as in a previous challenge is generated, the key from that challenge will be used again to maintain consistency (see lines 24 and 25). In addition to that, the random oracle has to be modified again so that it now outputs the same K as chosen before in case a secret key has already been opened and Z is computed correctly (see line 11). The adversary's view does not change as $\mathsf{H}_n(c, \mathsf{Priv}(\mathsf{sk}_n, c)) = K$

$\underline{\mathcal{B}(\mathcal{Y}, \mathcal{X}, (c_{n,k})_{n\in[N],k\in[q_E]})}$
00 **for** $n \in [N]$
01 $\quad \mathsf{sk}_n \xleftarrow{\$} \mathcal{SK}$
02 $\quad \mathsf{pk}_n \coloneqq \mu(\mathsf{sk}_n)$
03 $\quad \text{opened}[n] \coloneqq \textbf{false}$
04 $\quad \mathcal{CK}_n \coloneqq \emptyset$
05 $\quad \text{cnt}[n] \coloneqq 0$
06 $b' \leftarrow \mathcal{A}^{\textsc{Encaps},\textsc{Decaps},\textsc{Open},\mathsf{H}_1,\ldots,\mathsf{H}_N}(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
07 **return** b'

$\underline{\textsc{Encaps}(n \in [N], m)}$
08 $j \coloneqq \text{cnt}[n]\texttt{++}$
09 $c \coloneqq c_{n,j}$
10 $K \coloneqq \mathsf{H}_n(c, \mathsf{Priv}(\mathsf{sk}_n, c))$
11 **return** (c, K)

Figure 8: Adversary $\mathcal{B}$ against the $(N \cdot q_E)$-fold subset membership problem for the proof of Theorem 1, where H_n for $n \in [N]$, Decaps and Open are defined as in G_1 of Fig. 7.

for every $(c, K) \in \mathcal{CK}_n$ if sk_n is opened. If sk_n is not opened, K is independent of H_n in both games G_4 and G_5. Hence,

$$\Pr[G_5^{\mathcal{A}} \Rightarrow 1] = \Pr[G_4^{\mathcal{A}} \Rightarrow 1] \ .$$

Finally, observe that the last game G_5 is the original $\mathsf{NCKE}_{\mathsf{sim}}$ game. Hence,

$$\Pr[G_5^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{NCKE}_{\mathsf{sim}}^{\mathcal{A}} \Rightarrow 1] \ .$$

Collecting all probabilities yields the bound stated in Theorem 1. □

We will give two concrete instantiations, one based on the DDH assumption (Section 7.1) and one based on the higher residuosity assumption (Appendix A).

4 Security Model for Two-Message Authenticated Key Exchange

A two-message key exchange protocol $\mathsf{AKE} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{R}}, \mathsf{Der}_{\mathrm{I}})$ consists of four algorithms which are executed interactively by two parties as shown in Figure 9. We denote the party which initiates the session by P_i and the party which responds to the session by P_r. The key generation algorithm $\mathsf{Gen}_{\mathsf{AKE}}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ for one party. The initialization algorithm $\mathsf{Init}_{\mathrm{I}}$ inputs the initiator's long-term secret key sk_i and the responder's long-term public key pk_r and outputs a message I and a state st. The responder's derivation algorithm $\mathsf{Der}_{\mathrm{R}}$ takes as input the responder's long-term secret key sk_r, the initiator's long-term public key pk_i and a message I. It computes a message R and a session key K. The initiator's derivation algorithm $\mathsf{Der}_{\mathrm{I}}$ inputs the initiator's long-term secret key sk_i, the responder's long-term public key pk_r, a message R and a state st. It outputs a session key K.

Note that in contrast to the initiating party P_i, the responding party P_r will not be required to save any (secret) state information besides the session key K. The session key can be derived immediately after receiving the initiator's message.

Following [HKSU20], we define a game-based security model for authenticated key exchange using pseudocode. Our models for two different levels of security are given

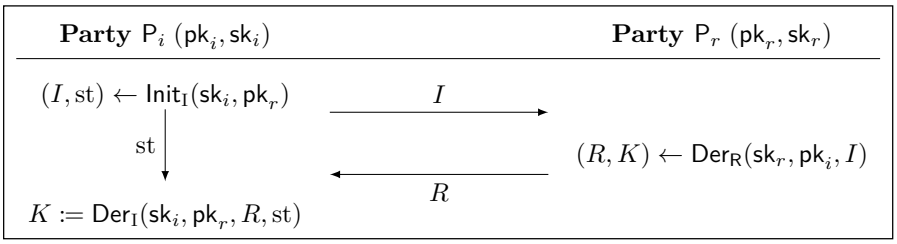

Figure 9: Running a key exchange protocol between two parties.

in Figure 10. We consider N parties $\mathsf{P}_1, ..., \mathsf{P}_N$ with long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$, $n \in [N]$. Each session between two parties has a unique identification number sID and variables which are defined relative to sID:

- init[sID] $\in [N]$ denotes the initiator of the session.
- resp[sID] $\in [N]$ denotes the responder of the session.
- type[sID] $\in$ {"In", "Re"} denotes the session's view, i. e., whether the initiator or the responder computes the session key.
- I[sID] denotes the message that was computed by the initiator.
- R[sID] denotes the message that was computed by the responder.
- state[sID] denotes the state information that is stored by the initiator.
- sKey[sID] denotes the session key.

To establish a session between two parties, the adversary is given access to oracles $\textsc{Session}_\mathrm{I}$ and $\textsc{Session}_\mathsf{R}$, where the first one starts a session of type "In" and the second one of type "Re". Following [Kra05, LLM07], these oracles also take the intended peer's identity as input. In order to complete the initiator's session, the oracle $\textsc{Der}_\mathrm{I}$ has to be queried. Furthermore, the adversary has access to oracles $\textsc{Corrupt}$, $\textsc{Reveal}$ and $\textsc{Rev-State}$ to obtain secret information. As the responder can directly compute the key in a two-message protocol, we only require the initiator to store a state. The state contains information that is needed to compute the session key when the response is received, so it will consist of public and private information. We do not require to reveal the full randomness as in the eCK model [LLM07]. A $\textsc{Rev-State}$ query may be issued at any time. We use the following boolean values to keep track of which queries the adversary made:

- corrupted[n] denotes whether the long-term secret key of party P_n was given to the adversary.
- revealed[sID] denotes whether the session key was given to the adversary.
- revState[sID] denotes whether the state information of that session was given to the adversary.
- peerCorrupted[sID] denotes whether the peer of the session was corrupted at the time the session key is computed, which is important for forward security.

The adversary can forward messages between sessions or modify them. By that, we can define the relationship between two sessions:

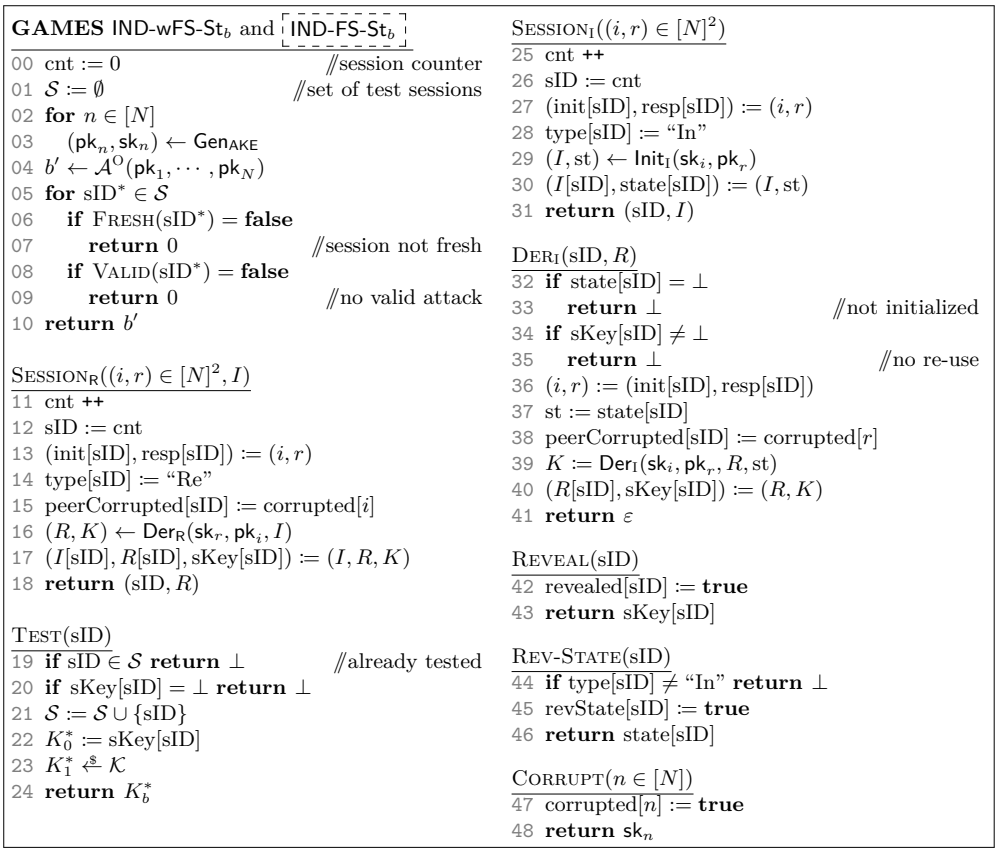

Figure 10: Games IND-wFS-St$_b$ and IND-FS-St$_b$ for AKE, where $b \in \{0, 1\}$. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}\}$. Helper procedures Fresh and Valid are defined in Figure 11. If there exists any test session which is neither fresh nor valid, the game will return 0.

- **Matching Session**: Two sessions sID, sID′ *match* if the same parties are involved (init[sID] = init[sID′] and resp[sID] = resp[sID′]), the messages sent and received are the same (I[sID] = I[sID′] and R[sID] = R[sID′]) and they are of different types (type[sID] ≠ type[sID′]).
- **Partially Matching Session**: A session sID′ of type "In" is *partially matching* to session sID of type "Re" if the initial messages are the same (I[sID] = I[sID′]).

Finally, the adversary is given access to oracle Test which will return either the session key of the specified session or a uniformly random key. In our security models, we allow multiple test queries. We store test sessions in a set $\mathcal{S}$. In general, the adversary can disclose the complete interaction between two parties by querying the long-term secret keys, the state information and the session key. However, for each test session, we require that the adversary does not issue queries such that the session key can be

```
FRESH(sID*)
00 (i*, r*) := (init[sID*], resp[sID*])
01 M(sID*) := {sID | (init[sID], resp[sID]) = (i*, r*) ∧ (I[sID], R[sID]) = (I[sID*], R[sID*])
                   ∧ type[sID] ≠ type[sID*]}                          //matching sessions
02 if revealed[sID*] or (∃sID ∈ M(sID*) : revealed[sID] = true)
03   return false                                  //A trivially learned the test session's key
04 if ∃sID ∈ M(sID*) s.t. sID ∈ S
05   return false                                  //A also tested a matching session
06 return true

VALID(sID*)
07 (i*, r*) := (init[sID*], resp[sID*])
08 M(sID*) := {sID | (init[sID], resp[sID]) = (i*, r*) ∧ (I[sID], R[sID]) = (I[sID*], R[sID*])
                   ∧ type[sID] ≠ type[sID*]}                          //matching sessions
09 P(sID*) := {sID | I[sID] = I[sID*] ∧ type[sID] = "In" ∧ type[sID] ≠ type[sID*]}
                                                                      //partially matching sessions
10 for attack ∈ Table 1 Table 2
11   if attack = true return true
12 return false
```

Figure 11: Helper procedures FRESH and VALID for games IND-wFS-St and IND-FS-St defined in Figure 10. Procedure FRESH checks if the adversary performed some trivial attack. In procedure VALID, each attack is evaluated by the set of variables shown in Table 1 (IND-wFS-St) or Table 2 (IND-FS-St) and checks if an allowed attack was performed. If the values of the variables are set as in the corresponding row, the attack was performed, i. e., attack = **true**, and thus the session is valid.

trivially computed. We define the properties of freshness and validity which all test sessions have to satisfy:

- **Freshness**: A (test) session is called *fresh* if the session key was not revealed. Furthermore, if there exists a matching session, we require that this session's key is not revealed and that this session is not also a test session.
- **Validity**: A (test) session is called *valid* if it is fresh and the adversary performed any attack which is defined in the security model. We capture this with attack tables (cf. Tables 1 and 2). A description of how to read the tables is given below.

Attack Tables. All attacks are defined using variables to indicate which queries the adversary may (not) make. We consider three dimensions covering all possible combinations of reveal queries the adversary can make:

- whether the test session is on the initiator's (type[sID*] ="In") or the responder's side (type[sID*] ="Re"),
- all combinations of long-term secret key and state reveals (corrupted and revState variables), also taking into account when a corruption happened (peerCorrupted),
- whether the adversary acted passively (matching session), partially active (partially matching session) or actively (no matching session).

This yields a full table of 24 attacks (cf. Table 3 in Appendix B), in particular capturing *key compromise impersonation* (KCI) and *maximal exposure* (MEX) attacks. An attack was performed if the variables are set to the corresponding values in the table. However,

	$\mathcal{A}$ gets (Initiator, Responder)	corrupted[i^*]	corrupted[r^*]	type[sID*]	revState[sID*]	$\exists$sID $\in \mathfrak{M}$(sID*) : revState[sID]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$	$\exists$sID $\in \mathfrak{P}$(sID*) : revState[sID]	$\lvert\mathfrak{P}(\text{sID}^*)\rvert$
(0)	**multiple partially matching sessions**	–	–	–	–	–	–	–	> 1
(1∨2)	**(long-term, long-term)**	–	–	–	**F**	**F**	1	–	–
(7∨8)	**(state, long-term)**	**F**	–	–	–	–	1	–	–
(10)	**(long-term, long-term)**	–	–	"Re"	**F**	n/a	0	**F**	1
(16)	**(state, long-term)**	**F**	–	"Re"	**F**	n/a	0	–	1
(19)	**(state, state)**	**F**	**F**	"In"	–	n/a	0	n/a	0
(21)	**(long-term, state)**	–	**F**	"In"	**F**	n/a	0	n/a	0
(24)	**(state, long-term)**	**F**	–	"Re"	**F**	n/a	0	n/a	0

Table 1: Distilled table of attacks for wFS adversaries against two-message protocols. This table is obtained from the full table of attacks by using that responders do not have a state and that we are considering weak forward security. The numbering of attacks is inherited from the full table. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. **F** means "false" and "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

when considering two-message protocols, where the responder's side does not have a state, and we only consider *weak forward security*, some of the attacks are redundant. Thus, we obtain *distilled* tables. We exclude trivial attacks, e.g., the generic attack on two-message AKE protocols with state-reveals described in [LS17]. Therefore, the adversary is not allowed to obtain the state of a partially matching session. Also note that by definition, a partially matching session for a two-message protocol can only be of type "Re". Table 1 is the distilled table used for the IND-wFS-St security game and Table 2 is used for the IND-FS-St security game. A more detailed justification on how the distilled tables are obtained by pointing out trivial attacks is given in Appendix B. Note that the numbering of attacks in the distilled tables is inherited from the full table.

However, if the protocol does not use appropriate randomness, it should not be considered secure in our model. Thus, if the adversary is able to create more than one (partially) matching session to a test session, it may also run a trivial attack. We model this in row (0) of Tables 1 and 2.

Example. If the test session is an initiating session (type[sID*] ="In"), the state was not revealed (revState[sID*] = **false**) and there is a matching session ($|\mathfrak{M}(\text{sID}^*)| = 1$), then row (1∨2) will evaluate to true. In this scenario, the adversary is allowed to query both long-term secret keys.

For all test sessions, at least one attack has to evaluate to true. Then, the adversary wins if it distinguishes the session keys from uniformly random keys which it obtains through queries to the Test oracle.

	$\mathcal{A}$ gets (Initiator, Responder)	corrupted[i^*]	corrupted[r^*]	peerCorrupted[sID*]	type[sID*]	revState[sID*]	$\exists$sID $\in \mathfrak{M}$(sID*) : revState[sID]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$	$\exists$sID $\in \mathfrak{P}$(sID*) : revState[sID]	$\lvert\mathfrak{P}(\text{sID}^*)\rvert$
(0)	**multiple partially matching sessions**	–	–	–	–	–	–	–	–	> 1
(1∨2)	**(long-term, long-term)**	–	–	–	–	**F**	**F**	1	–	–
(7∨8)	**(state, long-term)**	**F**	–	–	–	–	–	1	–	–
(10)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	**F**	1
(16)	**(state, long-term)**	**F**	–	–	"Re"	**F**	n/a	0	–	1
(17)	**(long-term, long-term)**	–	–	**F**	"In"	**F**	n/a	0	n/a	0
(18)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	n/a	0
(23)	**(state, long-term)**	**F**	–	**F**	"In"	–	n/a	0	n/a	0

Table 2: Distilled table of attacks for full FS adversaries against two-message protocols. This table is obtained from the full table of attacks by removing redundant rows and using that responders do not have a state. The numbering of attacks is inherited from the full table. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. **F** means "false" and "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

Definition 2 (Key Indistinguishability of AKE)**.** *We define games* $\mathsf{IND\text{-}wFS\text{-}St}_b$ *and* $\mathsf{IND\text{-}FS\text{-}St}_b$ *for* $b \in \{0,1\}$ *as in Figures 10 and 11. The advantage of an adversary* $\mathcal{A}$ *against* AKE *in these games is defined as*

$$\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr[\mathsf{IND\text{-}wFS\text{-}St}^{\mathcal{A}}_1 \Rightarrow 1] - \Pr[\mathsf{IND\text{-}wFS\text{-}St}^{\mathcal{A}}_0 \Rightarrow 1]\right| \quad \textit{and}$$
$$\mathrm{Adv}^{\mathsf{IND\text{-}FS\text{-}St}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_1 \Rightarrow 1] - \Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_0 \Rightarrow 1]\right| \ .$$

When proving the security of a protocol, the success probability for each attack strategy listed in the corresponding table will have to be analyzed, thus showing that independently of which queries the adversary makes, it cannot distinguish the session key from a uniformly random key.

4.1 Relation to other Definitions

In this section, we will refer to the most widely used security definitions for authenticated key exchange protocols. In the first place, these include the CK model [CK01] and the stronger definition used for the HMQV protocol (CK+) in [Kra05], the eCK model [LLM07] and the strengthened version of [CF12], the definitions given in [JKSS12] and [BHJ+15] which are both extensions of the BR model [BR94], and the definition of IND-AA security in [HKSU20]. In [Cre09, Cre11], Cremers showed that the CK, CK+ und eCK model are incomparable. Thus, we will not do a formal comparison of security

models, but only point out similarities and differences between our definition and the definitions listed above.

Party Corruption. We allow the adversary to corrupt a party which means that it will obtain that party's long-term secret key as in the eCK model and the models given in [JKSS12, BHJ+15, HKSU20]. In contrast, a corrupt query in the CK and CK+ model will reveal all information in the memory of that party, i. e., long-term secrets and session-specific information.

State-Reveals. Our model only allows state-reveal queries on initiating sessions because the initiator has to wait for the response to compute the session key. Thus, the state contains all that information that is needed to derive the session key as soon as the responder's message is received. The responder can directly compute the session key and does not have to store other information. The eCK model explicitly defines the state as the randomness that is used in the protocol. In the CK model, it is not clear which information is included in the state, but it is left to be specified by the AKE protocol itself. Other models such as [JKSS12], its extension given in [BHJ+15] and the one used in [CCG+19] do not allow state-reveals at all.

Here, we want to emphasize that in particular all previous work on tight AKE does not consider state reveals and we are the first ones to address this problem.

(Weak) Forward Security. Following Krawczyk [Kra05], we specify two levels of forward security. IND-wFS-St models weak forward security, whereas IND-FS-St models full forward security. The first one is intended for 2-message protocols with implicit authentication, as those cannot achieve full forward security [Kra05]. The second one is intended for protocols with explicit authentication. With those definitions, we capture the same properties as the most common security models given in [CK01, Kra05, LLM07, JKSS12, BHJ+15], where some of them only define either weak or full forward security depending on whether they consider implicitly or explicitly authenticated protocols.

Matching Sessions and Partnering. As most security models, ours use the concept of matching sessions to define a relation between two sessions. Following Cremer and Feltz [CF12], we additionally use the term of origin (or partially matching) sessions, which refers to a relaxation of the definition of matching sessions. The concept of origin sessions is used for full forward security, in particular we need this to handle the no-match attack described by Li and Schäge [LS17], where two sessions compute the same session key but do not have matching conversations. Recent works such as [GJ18, CCG+19] take up the approach of origin sessions and oracle partnering based on session keys as additional requirement.

On registering corrupt keys. Some security models for AKE allow the adversary also to *register* adversarially-generated keys, this holds in particular for previous works considering tightly-secure key exchange [BHJ+15, GJ18, CCG+19]. Technically this makes the security model strictly stronger, as one can easily construct contrived protocols that are insecure with adversarially-registered keys, but secure without.

However, in the actual security proofs in [BHJ+15, GJ18, CCG+19], adversarially-registered keys are treated no differently than corrupted keys. We chose to keep model, security proofs and notation as simple as possible (it is already complex enough, anyway), and thus omitted this query. However, it is straightforward to extend our model with it,

and the proofs need not to be changed. Whenever the adversary registers a new key, it would immediately be marked as "corrupted" (just like in [BHJ+15, GJ18, CCG+19]). Apart from that, no additional changes to the proofs are required, since the proofs deal with all corrupted keys in the same way, regardless of their distribution or whether they are generated by the experiment or an external entity. We also do not require a proof of knowledge of the corrensponding secret key for the registration, or a proof that the registered public key is valid in any sense.

5 AKE with Weak Forward Security

In this section, we show how to build an implicitly authenticated AKE protocol using the concept of non-committing key encapsulation.

In particular, from two key encapsulation mechanisms $\mathsf{KEM}_{\mathsf{CPA}} = (\mathsf{Gen}_{\mathsf{CPA}}, \mathsf{Encaps}_{\mathsf{CPA}}, \mathsf{Decaps}_{\mathsf{CPA}})$ and $\mathsf{KEM}_{\mathsf{CCA}} = (\mathsf{Gen}_{\mathsf{CCA}}, \mathsf{Encaps}_{\mathsf{CCA}}, \mathsf{Decaps}_{\mathsf{CCA}})$, we construct a two-message authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{wFS}} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathsf{I}}, \mathsf{Der}_{\mathsf{R}}, \mathsf{Der}_{\mathsf{I}})$ as shown in Figures 12 and 13. W.l.o.g. $\mathsf{KEM}_{\mathsf{CPA}}$, $\mathsf{KEM}_{\mathsf{CCA}}$, $\mathsf{AKE}_{\mathsf{wFS}}$ have identical key space $\mathcal{K}$. Each party holds a long-term key pair $(\mathsf{pk}, \mathsf{sk})$ for $\mathsf{KEM}_{\mathsf{CCA}}$ and a symmetric key k to encrypt the secret state information which has to be stored by the initiating party. State encryption protects against state attacks and is implemented using a symmetric encryption scheme defined as $\mathsf{E}_k(\mathrm{st}') := (IV, \mathsf{G}(k, IV) \oplus \mathrm{st}')$ for a random nonce IV. Here $\mathsf{G} : \{0,1\}^* \to \{0,1\}^d$ is a random oracle and d is an integer denoting the maximum bit length of the unencrypted state st'. The protocol uses an additional cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ to output the session key.

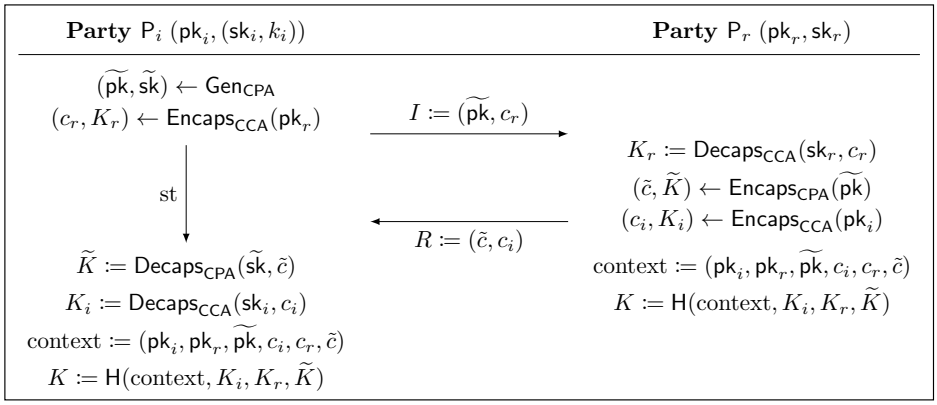

Figure 12: Visualization: Running protocol $\mathsf{AKE}_{\mathsf{wFS}}$ between two parties.

The initiating party generates an ephemeral key pair for $\mathsf{KEM}_{\mathsf{CPA}}$, then runs the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the peer's public key to output a ciphertext c_r and a key K_r and sends the ephemeral public key and c_r to the intended receiver. All values are stored temporarily and encrypted as described above, as they will later be needed to compute the session key. The responding party uses its secret key sk_r to compute key K_r from c_r. Next, it runs the $\mathsf{Encaps}_{\mathsf{CPA}}$ algorithm on the received ephemeral public key to compute

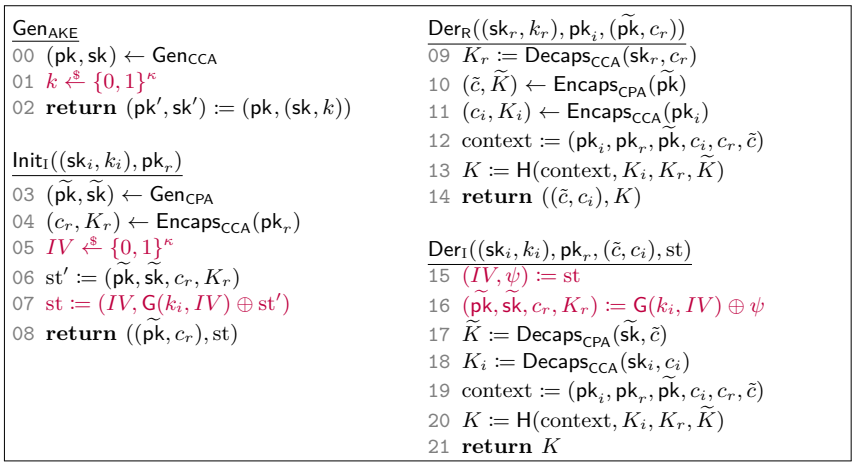

Figure 13: Authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{wFS}}$ from $\mathsf{KEM}_{\mathsf{CPA}}$ and $\mathsf{KEM}_{\mathsf{CCA}}$. Lines written in purple color are only used to encrypt the state.

a ciphertext $\tilde{c}$ and a key $\widetilde{K}$ and then the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the initiator's public key to output c_i and K_i. It sends both ciphertexts to the initiating party and computes the session key evaluating the hash function H on all public context and the three shared keys K_r, K_i and $\widetilde{K}$. The initiator retrieves the secret state information and computes K_i and $\widetilde{K}$ from c_i and $\tilde{c}$. Now, it can also establish the session key.

Theorem 2 ($\mathsf{KEM}_{\mathsf{CPA}}$ NCKE-CPA + $\mathsf{KEM}_{\mathsf{CCA}}$ NCKE-CCA $\overset{\text{ROM}}{\Rightarrow}$ $\mathsf{AKE}_{\mathsf{wFS}}$ IND-wFS-St)**.** *For any* IND-wFS-St *adversary* $\mathcal{A}$ *against* $\mathsf{AKE}_{\mathsf{wFS}}$ *with* N *parties that establishes at most* S *sessions and issues at most* T *queries to the test oracle* Test, q_{G} *queries to random oracle* G *and at most* q_{H} *queries to random oracle* H*, there exists an* N-NCKE-CCA *adversary* $\mathcal{B}$ *against* $\mathsf{KEM}_{\mathsf{CCA}}$ *and* $\mathsf{Sim}_{\mathsf{CCA}}$ *and an* S-NCKE-CPA *adversary* $\mathcal{C}$ *against* $\mathsf{KEM}_{\mathsf{CPA}}$ *and* $\mathsf{Sim}_{\mathsf{CPA}}$ *such that*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}_{\mathsf{wFS}}}(\mathcal{A}) \leq\ & 2\cdot\left(\mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}) + \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C})\right) + T\cdot\left(\frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right)\\ &+ N^2\cdot\left(\frac{1}{2^{\mu_{\mathsf{CCA}}}} + \frac{1}{2^{\kappa}}\right) + S^2\cdot\left(\frac{1}{2^{\mu_{\mathsf{CPA}}}} + \frac{1}{2^{\gamma_{\mathsf{CCA}}}} + \frac{1}{2^{\gamma_{\mathsf{CPA}}}} + \frac{1}{2^{\kappa}}\right) + 2S\cdot\frac{q_{\mathsf{G}}}{2^{2\kappa}}\ ,\end{aligned}$$

where $\mathsf{Sim}_{\mathsf{CCA}}$ *and* $\mathsf{Sim}_{\mathsf{CPA}}$ *are the simulators from the* NCKE *experiments,* μ_{CCA} *and* μ_{CPA} *are the collision probability of the key generation algorithms* $\mathsf{Gen}_{\mathsf{CCA}}$ *and* $\mathsf{Gen}_{\mathsf{CPA}}$*,* γ_{CCA} *and* γ_{CPA} *are the spreadness parameters of the encapsulation algorithms* $\mathsf{Encaps}_{\mathsf{CCA}}$ *and* $\mathsf{Encaps}_{\mathsf{CPA}}$ *and* κ *is a security parameter. The running times of* $\mathcal{B}$ *and* $\mathcal{C}$ *consist essentially of the time required to execute the security experiment with the adversary once, plus a minor number of additional operations (including bookkeeping, lookups etc.).*

Proof. Let $\mathcal{A}$ be an adversary against IND-wFS-St security of $\mathsf{AKE}_{\mathsf{wFS}}$, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is

the maximum number of test queries. Consider the sequence of games in Figures 14 and 15.

GAMES $G_{0,b}$. These are the original $\mathsf{IND\text{-}wFS\text{-}St}_b$ games. In order to exclude collisions, we implicitly assume that all key pairs, long-term keys as well as ephemeral keys generated by $\mathsf{Gen}_{\mathsf{CCA}}$ and $\mathsf{Gen}_{\mathsf{CPA}}$, and all ciphertexts output by the $\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{CCA}}$ and $\mathsf{Encaps}^{\widetilde{\mathsf{H}}}_{\mathsf{CPA}}$ algorithms are distinct. (If such a collision happens at any time in the game, we would abort. However, for sake of readability we do not explicitly write that in the code of games $G_{0,b}$.)

We consider this in our bound. Therefore, let μ_{CCA} and μ_{CPA} be the collision probabilities of the key generation algorithms $\mathsf{Gen}_{\mathsf{CCA}}$ and $\mathsf{Gen}_{\mathsf{CPA}}$ and let γ_{CCA} and γ_{CPA} be the spreadness parameters of $\mathsf{KEM}_{\mathsf{CCA}}$ and $\mathsf{KEM}_{\mathsf{CPA}}$. Then by union bound and the birthday bound, the upper bound for key collisions is $N^2/2^{\mu_{\mathsf{CCA}}} + S^2/2^{\mu_{\mathsf{CPA}}}$ and for ciphertext collisions $S^2/2^{\gamma_{\mathsf{CCA}}} + S^2/2^{\gamma_{\mathsf{CPA}}}$, as we have N parties, at most S sessions with at most one ephemeral key pair and at most two ciphertexts. We also assume that values k_n and IV are distinct, which adds the additional term $N^2/2^{\kappa} + S^2/2^{\kappa}$, where κ is the bit length of k_n and IV.

We additionally store the state of a session sID in plaintext in variable state′[sID] (line 59, Fig. 14) which is directly accessed in $\textsc{Der}_{\mathrm{I}}$, instead of decrypting the state. This is only conceptual. For bookkeeping, we introduce the two sets $\mathcal{C}$ and $\mathcal{CK}$ from the NCKE-CCA game in lines 35, 36 (Fig. 15) and 51, 52 (Fig. 14). In total, we have

$$\begin{aligned}|\Pr[\mathsf{IND\text{-}wFS\text{-}St}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{IND\text{-}wFS\text{-}St}_0^{\mathcal{A}} \Rightarrow 1]| &\leq \left|\Pr[G_{0,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{0,0}^{\mathcal{A}} \Rightarrow 1]\right| \\ &+ N^2 \cdot \left(2^{-\mu_{\mathsf{CCA}}} + 2^{-\kappa}\right) + S^2 \cdot \left(2^{-\mu_{\mathsf{CPA}}} + 2^{-\gamma_{\mathsf{CCA}}} + 2^{-\gamma_{\mathsf{CPA}}} + 2^{-\kappa}\right) . \qquad (2)\end{aligned}$$

In the following, we want to use the property of receiver non-committing key encapsulation of $\mathsf{KEM}_{\mathsf{CCA}}$. Therefore, we use the simulator $\mathsf{Sim}_{\mathsf{CCA}} = (\mathsf{SimGen}_{\mathsf{CCA}}, \mathsf{SimEncaps}_{\mathsf{CCA}}, \mathsf{SimHash}_{\mathsf{CCA}})$ which is defined relative to $\mathsf{KEM}_{\mathsf{CCA}}$.

GAMES $G_{1,b}$. In games $G_{1,b}$, we use the $\mathsf{SimGen}_{\mathsf{CCA}}$ algorithm to generate long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$ in line 04 (Fig. 14). $\textsc{Session}_{\mathrm{I}}$ uses the $\mathsf{SimEncaps}_{\mathsf{CCA}}$ algorithm to compute c_r in line 49 and draws a random key K_r in line 50. To maintain consistency, challenges are saved in line 52 and K_r is retrieved in $\textsc{Session}_{\mathsf{R}}$ (line 28, Fig. 15) when the same c_r is issued. The same is done for ciphertexts c_i and keys K_i in $\textsc{Session}_{\mathsf{R}}$: $\mathsf{SimEncaps}_{\mathsf{CCA}}$ is used to generate c_i (line 33, Fig. 15), K_i is drawn uniform at random (line 34) and both are saved and retrieved (line 36, Fig. 15 and line 72, Fig. 14). Furthermore, the $\mathsf{SimHash}_{\mathsf{CCA}}$ algorithm is used in random oracles H_n, where $n \in [N]$. In case party n is corrupted, i. e., sk_n is known to $\mathcal{A}$, we call $\mathsf{SimHash}_{\mathsf{CCA}}$ with set $\mathcal{CK}_n$, otherwise with set $\mathcal{C}_n$.

For $b \in \{0, 1\}$, we construct adversaries $\mathcal{B}_b$ against N-NCKE-CCA security of $\mathsf{KEM}_{\mathsf{CCA}}$ in Figure 16. $\mathcal{B}_b$ inputs long-term public keys $\mathsf{pk}_1, ..., \mathsf{pk}_N$ and has access to oracles $\textsc{Encaps}, \textsc{Decaps}$ and $\textsc{Open}$ as well as random oracles H'_n, where $n \in [N]$. $\mathcal{B}_b$ generates N symmetric keys k_n which are part of the long-term secret key and forwards its input public keys to $\mathcal{A}$.

If $\mathcal{A}$ queries $\textsc{Session}_{\mathrm{I}}$, $\mathcal{B}_b$ generates an ephemeral key pair $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$. Next, $\mathcal{B}_b$ calls $\textsc{Encaps}$ on party r in line 35. As described before, challenges are saved and retrieved when ciphertexts match. If they do not match which means that c_r issued to $\textsc{Session}_{\mathsf{R}}$

GAMES $G_{0,b}$-$G_{4,b}$

00 $\mathrm{cnt} \coloneqq 0$
01 $\mathcal{S} \coloneqq \emptyset$
02 **for** $n \in [N]$
03 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$ //G_0
04 $\quad (\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{SimGen}_{\mathsf{CCA}}$ //$G_{1\text{-}4}$
05 $\quad k_n \xleftarrow{\$} \{0,1\}^\kappa$
06 $\quad (\mathsf{pk}'_n, \mathsf{sk}'_n) \coloneqq (\mathsf{pk}_n, (\mathsf{sk}_n, k_n))$
07 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}'_1, ..., \mathsf{pk}'_N)$
08 **for** $\mathrm{sID}^* \in \mathcal{S}$
09 $\quad$ **if** $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$ **return** 0
10 $\quad$ **if** $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$ **return** 0
11 **return** b'

$\textsc{Session}_{\mathsf{R}}((i,r) \in [N]^2, I)$

12 cnt ++
13 sID $\coloneqq$ cnt
14 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) \coloneqq (i,r)$
15 $\mathrm{type}[\mathrm{sID}] \coloneqq$ "Re"
16 $(\widetilde{\mathsf{pk}}, c_r) \coloneqq I$
17 $(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$ //$G_{0\text{-}2}$
// simulate $(\tilde{c}, \widetilde{K})$ when $\widetilde{\mathsf{pk}}$ comes from $\textsc{Session}_{\mathrm{I}}$:
18 **if** $\exists \mathrm{sID}'$ s. t. $\mathrm{state}'[\mathrm{sID}'] = (\widetilde{\mathsf{pk}}, \cdot, \cdot, \cdot)$ //$G_{3\text{-}4}$
19 $\quad (\cdot, \widetilde{\mathsf{sk}}, \cdot, \cdot) \coloneqq \mathrm{state}'[\mathrm{sID}']$ //$G_{3\text{-}4}$
20 $\quad \tilde{c} \leftarrow \mathsf{SimEncaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ //$G_{3\text{-}4}$
21 $\quad \widetilde{K} \xleftarrow{\$} \mathcal{K}$ //$G_{3\text{-}4}$
22 $\quad \widetilde{\mathcal{C}}_{\mathrm{sID}'} \coloneqq \widetilde{\mathcal{C}}_{\mathrm{sID}'} \cup \{(\tilde{c}, \bot)\}$ //$G_{3\text{-}4}$
23 $\quad \widetilde{\mathcal{CK}}_{\mathrm{sID}'} \coloneqq \widetilde{\mathcal{CK}}_{\mathrm{sID}'} \cup \{(\tilde{c}, \widetilde{K})\}$ //$G_{3\text{-}4}$
24 **else** //$G_{3\text{-}4}$
25 $\quad (\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$ //$G_{3\text{-}4}$
26 $K_r \coloneqq \mathsf{Decaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$ //G_0
// retrieve K_r when c_r used before:
27 **if** $\exists K'_r$ s. t. $(c_r, K'_r) \in \mathcal{CK}_r$ //$G_{1\text{-}4}$
28 $\quad K_r \coloneqq K'_r$ //$G_{1\text{-}4}$
29 **else** //$G_{1\text{-}4}$
30 $\quad K_r \coloneqq \mathsf{Decaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$ //$G_{1\text{-}4}$
31 $\quad \mathcal{D}_r \coloneqq \mathcal{D}_r \cup \{c_r\}$ //$G_{1\text{-}4}$
32 $(c_i, K_i) \leftarrow \mathsf{Encaps}^{\mathsf{H}_i}_{\mathsf{CCA}}(\mathsf{pk}_i)$ //G_0
// simulate (c_i, K_i):
33 $c_i \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_i, \mathsf{sk}_i)$ //$G_{1\text{-}4}$
34 $K_i \xleftarrow{\$} \mathcal{K}$ //$G_{1\text{-}4}$
35 $\mathcal{C}_i \coloneqq \mathcal{C}_i \cup \{(c_i, \bot)\}$
36 $\mathcal{CK}_i \coloneqq \mathcal{CK}_i \cup \{(c_i, K_i)\}$
37 $\mathrm{context} \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
38 $K \coloneqq \mathsf{H}(\mathrm{context}, K_i, K_r, \widetilde{K})$
39 $R \coloneqq (\tilde{c}, c_i)$
40 $(I[\mathrm{sID}], R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) \coloneqq (I, R, K)$
41 **return** (sID, R)

$\textsc{Session}_{\mathrm{I}}((i,r) \in [N]^2)$

42 cnt ++
43 sID $\coloneqq$ cnt
44 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) \coloneqq (i,r)$
45 $\mathrm{type}[\mathrm{sID}] \coloneqq$ "In"
46 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$ //$G_{0\text{-}2}$
47 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{SimGen}_{\mathsf{CPA}}$ //$G_{3\text{-}4}$
48 $(c_r, K_r) \leftarrow \mathsf{Encaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{pk}_r)$ //G_0
// simulate (c_r, K_r):
49 $c_r \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_r, \mathsf{sk}_r)$ //$G_{1\text{-}4}$
50 $K_r \xleftarrow{\$} \mathcal{K}$ //$G_{1\text{-}4}$
51 $\mathcal{C}_r \coloneqq \mathcal{C}_r \cup \{(c_r, \bot)\}$
52 $\mathcal{CK}_r \coloneqq \mathcal{CK}_r \cup \{(c_r, K_r)\}$
53 $I \coloneqq (\widetilde{\mathsf{pk}}, c_r)$
54 $IV \xleftarrow{\$} \{0,1\}^\kappa$
55 $\mathrm{st}' \coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r)$
56 $\mathrm{st} \coloneqq (IV, \mathsf{G}(k_i, IV) \oplus \mathrm{st}')$ //$G_{0\text{-}1}$
57 $\mathrm{st} \coloneqq (IV, \bot)$ //$G_{2\text{-}4}$
58 $(I[\mathrm{sID}]) \coloneqq (I, \mathrm{st})$
59 $\mathrm{state}'[\mathrm{sID}] \coloneqq \mathrm{st}'$
60 **return** (sID, I)

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, R)$

61 **if** $\mathrm{state}[\mathrm{sID}] = \bot$ **or** $\mathrm{sKey}[\mathrm{sID}] \neq \bot$ **return** $\bot$
62 $(i,r) \coloneqq (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
63 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r) \coloneqq \mathrm{state}'[\mathrm{sID}]$
64 $(\tilde{c}, c_i) \coloneqq R$
65 $\widetilde{K} \coloneqq \mathsf{Decaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$ //$G_{0\text{-}2}$
// retrieve $\widetilde{K}$ when $\tilde{c}$ used before:
66 **if** $\exists \widetilde{K}'$ s. t. $(\tilde{c}, \widetilde{K}') \in \widetilde{\mathcal{CK}}_{\mathrm{sID}}$ //$G_{3\text{-}4}$
67 $\quad \widetilde{K} \coloneqq \widetilde{K}'$ //$G_{3\text{-}4}$
68 **else** //$G_{3\text{-}4}$
69 $\quad \widetilde{K} \coloneqq \mathsf{Decaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$ //$G_{3\text{-}4}$
70 $K_i \coloneqq \mathsf{Decaps}^{\mathsf{H}_i}_{\mathsf{CCA}}(\mathsf{sk}_i, c_i)$ //G_0
// retrieve K_i when c_i used before:
71 **if** $\exists K'_i$ s. t. $(c_i, K'_i) \in \mathcal{CK}_i$ //$G_{1\text{-}4}$
72 $\quad K_i \coloneqq K'_i$ //$G_{1\text{-}4}$
73 **else** //$G_{1\text{-}4}$
74 $\quad K_i \coloneqq \mathsf{Decaps}^{\mathsf{H}_i}_{\mathsf{CCA}}(\mathsf{sk}_i, c_i)$ //$G_{1\text{-}4}$
75 $\quad \mathcal{D}_i \coloneqq \mathcal{D}_i \cup \{c_i\}$ //$G_{1\text{-}4}$
76 $\mathrm{context} \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
77 $K \coloneqq \mathsf{H}(\mathrm{context}, K_i, K_r, \widetilde{K})$
78 $(R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) \coloneqq (R, K)$
79 **return** ε

$\mathsf{H}(x)$

80 **if** $\exists K$ s. t. $(x, K) \in \mathcal{H}$ **return** K
81 $K \xleftarrow{\$} \mathcal{K}$
82 $\mathcal{H} \coloneqq \mathcal{H} \cup \{(x, K)\}$
83 **return** K

Figure 14: Games $G_{0,b}$-$G_{4,b}$ for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, ..., \mathsf{H}_N, \widetilde{\mathsf{H}}_1, ..., \widetilde{\mathsf{H}}_S\}$. $\textsc{Reveal}$ and $\textsc{Corrupt}$ are defined as in the original $\mathsf{IND\text{-}wFS\text{-}St}$ game (Fig. 10). $\textsc{Rev-State}$, $\textsc{Test}$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for $\mathrm{sID} \in [S]$ and H_n for $n \in [N]$ are defined in Fig. 15.

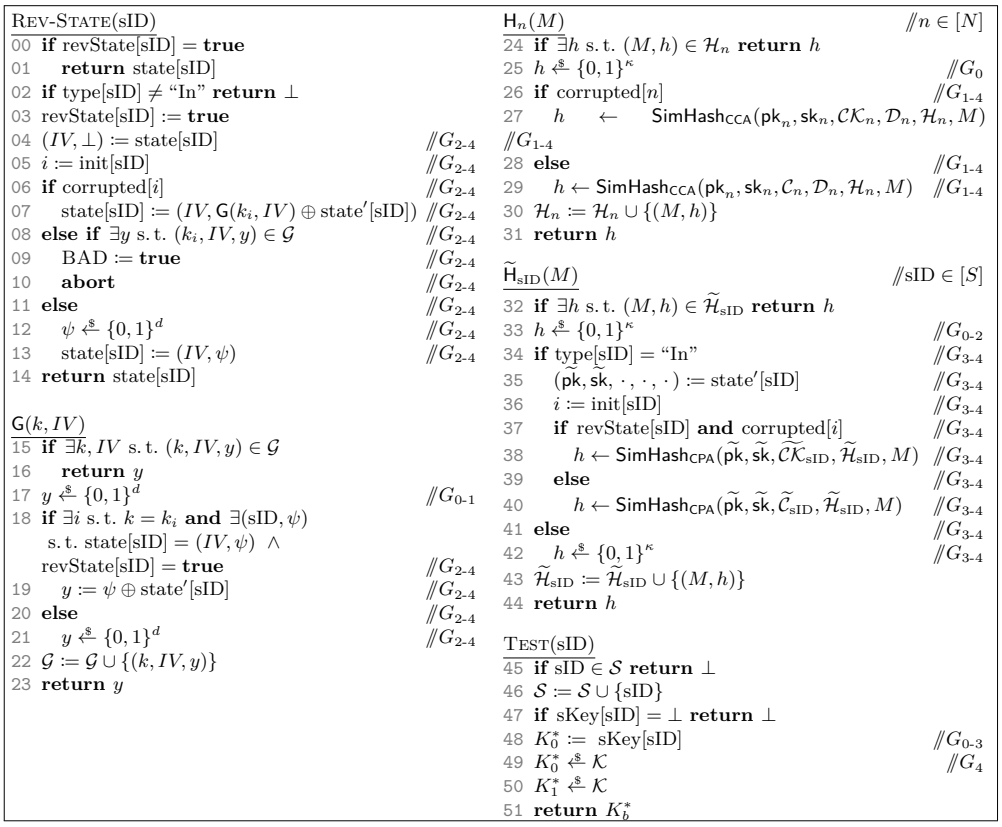

Figure 15: Oracles REV-STATE, G, H_n for $n \in [N]$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for $\mathrm{sID} \in [S]$ and TEST for games $G_{0,b}$-$G_{4,b}$ in Figure 14.

is new, DECAPS is queried to receive the corresponding key K_r in line 19. Next, $\mathcal{B}_b$ calls the ENCAPS oracle on party i (line 20) and in $\mathrm{DER_I}$ key K_i is retrieved (line 50) or DECAPS is queried (line 52).

If $\mathcal{A}$ queries CORRUPT on party n, $\mathcal{B}_b$ queries OPEN to obtain sk_n and outputs both sk_n and k_n. Queries to a random oracle H_n are forwarded to H'_n.

If $\mathcal{B}_b$ is in the $\mathsf{NCKE\text{-}CCA}_{\mathsf{real}}$ game, it perfectly simulates $G_{0,b}$. Otherwise, if $\mathcal{B}_b$ is in the $\mathsf{NCKE\text{-}CCA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{1,b}$. We have

$$\begin{aligned} \left|\Pr[G^{\mathcal{A}}_{1,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{0,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CCA}^{\mathcal{B}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CCA}^{\mathcal{B}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}_b) \end{aligned} \tag{3}$$

Instead of using the algorithms $\mathsf{Gen}_{\mathsf{CPA}}$ and $\mathsf{Encaps}_{\mathsf{CPA}}$, we now also want to use the simulator $\mathsf{Sim}_{\mathsf{CPA}} = (\mathsf{SimGen}_{\mathsf{CPA}}, \mathsf{SimEncaps}_{\mathsf{CPA}}, \mathsf{SimHash}_{\mathsf{CPA}})$ for the ephemeral keys and ciphertexts. However, we first introduce an intermediate game which moves the encryption of the state to the REV-STATE oracle. We do this to prepare the reduction to $\mathsf{NCKE\text{-}CPA}$ security, where $\widetilde{\mathsf{sk}}$ will not be available and thus the state cannot be

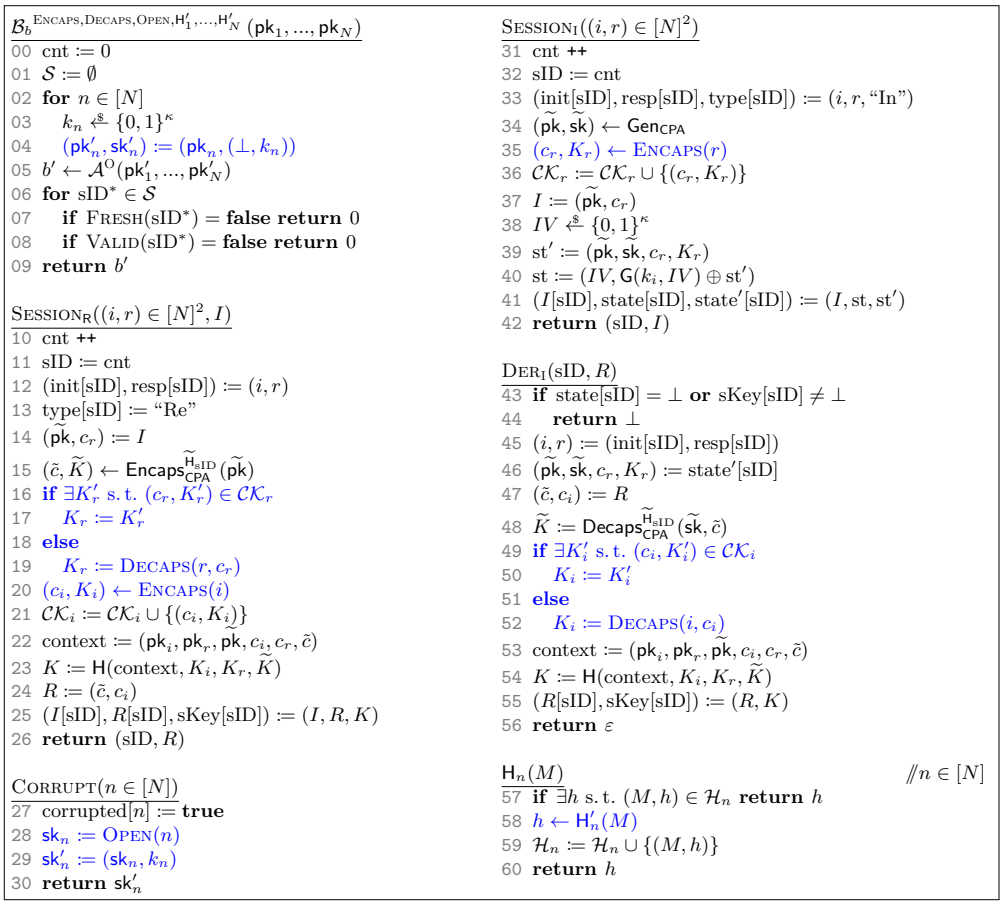

Figure 16: Adversaries $\mathcal{B}_b$ against N-NCKE-CCA for the proof of Eqn. (3). $\mathcal{A}$ has access to oracles $\text{O} := \{\text{SESSION}_\text{I}, \text{SESSION}_\text{R}, \text{DER}_\text{I}, \text{REVEAL}, \text{REV-STATE}, \text{CORRUPT}, \text{TEST}, \mathsf{G}, \mathsf{H}, \widetilde{\mathsf{H}}_1, \ldots \widetilde{\mathsf{H}}_S, \mathsf{H}_1, \ldots, \mathsf{H}_N\}$, where REVEAL, REV-STATE, TEST, H and $\widetilde{\mathsf{H}}_{\text{sID}}$ for sID $\in [S]$ are defined as in Figure 14 resp. 15. Lines written in blue color highlight how the adversary simulates $G_{0,b}$ and interpolates to $G_{1,b}$.

computed in the first place. This means we first delay the computation of the state as long as possible in games $G_{2,b}$ and then in games $G_{3,b}$ the simulator will finally be used.

GAMES $G_{2,b}$. Here, we move the encryption of the state from SESSION$_\text{I}$ (line 56, Fig. 14) to the REV-STATE oracle. When REV-STATE is queried, we check if the initiator i is corrupted in line 06 and honestly compute the state because then the adversary can simply make the same computation. Next, we check if the adversary already made a query to G, where (k_i, IV) are as in the corresponding session. If this is the case, we raise flag BAD in line 09 and abort. Otherwise, we choose a random string ψ in line 12 and return this value. Line 00 ensures that answers will be consistent when REV-STATE

is queried twice on the same sID. However, we have to patch G for the case that $\mathcal{A}$ issues a query with the correct symmetric key k_i such that the random value ψ decrypts correctly. Note that if BAD is not raised, the adversary's view is the same in games $G_{2,b}$ and $G_{1,b}$. As BAD implies that G is queried on any correct pair (k_i, IV) although $IV \in \{0,1\}^\kappa$ is unknown and $k_i \in \{0,1\}^\kappa$ is also unknown because otherwise the state would have been computed honestly in line 07, we have

$$\left|\Pr[G_{2,b} \Rightarrow 1] - \Pr[G_{1,b} \Rightarrow 1]\right| \leq \Pr[\text{BAD}] \leq S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} \ .$$

Games $G_{3,b}$. In games $G_{3,b}$, $\textsc{Session}_{\text{I}}$ uses the $\mathsf{SimGen}_{\mathsf{CPA}}$ algorithm to generate ephemeral key pairs $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ in line 47 (Fig. 14). In lines 19-23 (Fig. 15), we first recover the ephemeral secret key $\widetilde{\mathsf{sk}}$ from the state of the corresponding initiating session if the ephemeral public key specified in I was output by $\textsc{Session}_{\text{I}}$, i. e., it was not chosen by the adversary. Next, the $\mathsf{SimEncaps}_{\mathsf{CPA}}$ algorithm is used to compute $\tilde{c}$ and we draw a random key $\widetilde{K}$. To maintain consistency, we save challenges to restore $\widetilde{K}$ later in line 67 (Fig. 14) if the same $\tilde{c}$ is queried to $\textsc{Der}_{\text{I}}$. Furthermore, we use the $\mathsf{SimHash}_{\mathsf{CPA}}$ algorithm in random oracles $\widetilde{\mathsf{H}}_{\text{sID}}$, but only for those sessions that choose their own ephemeral key pair $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ in $\textsc{Session}_{\text{I}}$ as explained above. In particular, we first check if sID is a session of type "In" in line 34 (Fig. 15). In case the state of that session is revealed and the initiator is corrupted, we call $\mathsf{SimHash}_{\mathsf{CPA}}$ with set $\widetilde{\mathcal{CK}}_{\text{sID}}$, otherwise with set $\widetilde{\mathcal{C}}_{\text{sID}}$.

For $b \in \{0,1\}$, we construct adversaries $\mathcal{C}_b$ against S-NCKE-CPA security of $\mathsf{KEM}_{\mathsf{CPA}}$ in Figures 17 and 18. $\mathcal{C}_b$ inputs S ephemeral public keys $\widetilde{\mathsf{pk}}_1, ..., \widetilde{\mathsf{pk}}_S$ and has access to oracles $\textsc{Encaps}$, $\textsc{Open}$ and random oracles $\widetilde{\mathsf{H}}'_s$, where $s \in [S]$.

If $\mathcal{A}$ queries $\textsc{Session}_{\text{I}}$, $\mathcal{C}_b$ computes (c_r, K_r) and sets $\widetilde{\mathsf{pk}}$ to $\widetilde{\mathsf{pk}}_{\text{sID}}$ in line 38 (Fig. 17). Note that it cannot assign $\widetilde{\mathsf{sk}}$ here and thus cannot define st$'$ in line 45 because $\widetilde{\mathsf{sk}}_{\text{sID}}$ is unknown. Thus, it cannot compute the state, but will only draw IV. When $\mathcal{A}$ issues a query to $\textsc{Rev-State}$, $\mathcal{C}_b$ checks if the initiator is corrupted, calls $\textsc{Open}$ to obtain the corresponding ephemeral secret key $\widetilde{\mathsf{sk}}$ (line 23, Fig. 18) and will then compute and output the complete state. If the initiator is not corrupted and $\mathcal{C}_b$ does not abort, ψ is chosen uniformly at random. If the adversary queries G on (k_i, IV), we patch the random oracle by calling $\textsc{Open}$ in line 02 (Fig. 18) and computing the correct value for st$'$, thus determining output value y.

If $\mathcal{A}$ queries $\textsc{Session}_{\mathsf{R}}$, $\mathcal{C}_b$ checks whether there exists another session sID$'$ with the same $\widetilde{\mathsf{pk}}$ and if it does, $\mathcal{C}_b$ queries its $\textsc{Encaps}$ oracle on sID$'$ in line 17 (Fig. 17). At that point note why we can only embed the challenge in those sessions with a $\widetilde{\mathsf{pk}}$ output by $\textsc{Session}_{\text{I}}$. An active adversary may query $\textsc{Session}_{\mathsf{R}}$ on any $\widetilde{\mathsf{pk}}$ that it has chosen himself. Thus, $\mathcal{C}_b$ might not be able to query the challenge oracle $\textsc{Encaps}$ on that $\widetilde{\mathsf{pk}}$.

As described before, we save challenges relative to sID$'$ in line 18 to restore them later again when $\textsc{Der}_{\text{I}}$ is called in line 55. If the adversary queries $\textsc{Der}_{\text{I}}$ on a new value $\tilde{c}$, $\mathcal{C}_b$ calls $\textsc{Open}$ to obtain $\widetilde{\mathsf{sk}}$ and compute $\widetilde{K}$. Next, $\mathcal{C}_b$ computes K_i from c_i, retrieves K_r from the state and finally calculates the session key K.

Queries to a random oracle $\widetilde{\mathsf{H}}_{\text{sID}}$ are forwarded to the corresponding oracle $\widetilde{\mathsf{H}}'_{\text{sID}}$ whenever a session is of type "In".

$\mathcal{C}_b^{\text{ENCAPS,OPEN,}\widetilde{\mathsf{H}}'_1,\ldots,\widetilde{\mathsf{H}}'_S}(\widetilde{\mathsf{pk}}_1,\ldots,\widetilde{\mathsf{pk}}_S)$

00 $\text{cnt} \coloneqq 0$
01 $\mathcal{S} \coloneqq \emptyset$
02 **for** $n \in [N]$
03 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$
04 $\quad k_n \xleftarrow{\$} \{0,1\}^\kappa$
05 $\quad(\mathsf{pk}'_n, \mathsf{sk}'_n) \coloneqq (\mathsf{pk}_n, (\mathsf{sk}_n, k_n))$
06 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}'_1, \ldots, \mathsf{pk}'_N)$
07 **for** $\text{sID}^* \in \mathcal{S}$
08 $\quad$ **if** $\textsc{Fresh}(\text{sID}^*) = \textbf{false}$ **return** 0
09 $\quad$ **if** $\textsc{Valid}(\text{sID}^*) = \textbf{false}$ **return** 0
10 **return** b'

$\textsc{Session}_{\mathrm{R}}((i,r) \in [N]^2, I)$

11 cnt ++
12 $\text{sID} \coloneqq \text{cnt}$
13 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) \coloneqq (i,r)$
14 $\text{type}[\text{sID}] \coloneqq$ "Re"
15 $(\widetilde{\mathsf{pk}}, c_r) \coloneqq I$
16 **if** $\exists \text{sID}'$ s. t. $\widetilde{\mathsf{pk}} = \widetilde{\mathsf{pk}}_{\text{sID}'}$
17 $\quad(\tilde{c}, \widetilde{K}) \leftarrow \textsc{Encaps}(\text{sID}')$
18 $\quad\widetilde{\mathcal{CK}}_{\text{sID}'} \coloneqq \widetilde{\mathcal{CK}}_{\text{sID}'} \cup \{(\tilde{c}, \widetilde{K})\}$
19 **else**
20 $\quad(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}_{\mathsf{CPA}}^{\widetilde{\mathsf{H}}_{\text{sID}}}(\widetilde{\mathsf{pk}})$
21 **if** $\exists K'_r$ s. t. $(c_r, K'_r) \in \mathcal{CK}_r$
22 $\quad K_r \coloneqq K'_r$
23 **else**
24 $\quad K_r \coloneqq \mathsf{Decaps}_{\mathsf{CCA}}^{\mathsf{H}_r}(\mathsf{sk}_r, c_r)$
25 $\quad\mathcal{D}_r \coloneqq \mathcal{D}_r \cup \{c_r\}$
26 $c_i \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_i, \mathsf{sk}_i)$
27 $K_i \xleftarrow{\$} \mathcal{K}$
28 $\mathcal{C}_i \coloneqq \mathcal{C}_i \cup \{(c_i, \bot)\}$
29 $\mathcal{CK}_i \coloneqq \mathcal{CK}_i \cup \{(c_i, K_i)\}$
30 $\text{context} \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
31 $K \coloneqq \mathsf{H}(\text{context}, K_i, K_r, \widetilde{K})$
32 $R \coloneqq (\tilde{c}, c_i)$
33 $(I[\text{sID}], R[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (I, R, K)$
34 **return** (sID, R)

$\textsc{Session}_{\mathrm{I}}((i,r) \in [N]^2)$

35 cnt ++
36 $\text{sID} \coloneqq \text{cnt}$
37 $(\text{init}[\text{sID}], \text{resp}[\text{sID}], \text{type}[\text{sID}]) \coloneqq (i, r, \text{"In"})$
38 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \coloneqq (\widetilde{\mathsf{pk}}_{\text{sID}}, \bot)$ // $\widetilde{\mathsf{sk}}_{\text{sID}}$ unknown
39 $c_r \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_r, \mathsf{sk}_r)$
40 $K_r \xleftarrow{\$} \mathcal{K}$
41 $\mathcal{C}_r \coloneqq \mathcal{C}_r \cup \{(c_r, \bot)\}$
42 $\mathcal{CK}_r \coloneqq \mathcal{CK}_r \cup \{(c_r, K_r)\}$
43 $I \coloneqq (\widetilde{\mathsf{pk}}, c_r)$
44 $IV \xleftarrow{\$} \{0,1\}^\kappa$
45 $\text{st}' \coloneqq (\widetilde{\mathsf{pk}}, \bot, c_r, K_r)$
46 $\text{st} \coloneqq (IV, \bot)$
47 $(I[\text{sID}], \text{state}[\text{sID}], \text{state}'[\text{sID}]) \coloneqq (I, \text{st}, \text{st}')$
48 **return** (sID, I)

$\textsc{Der}_{\mathrm{I}}(\text{sID}, R)$

49 **if** $\text{state}[\text{sID}] = \bot$ **or** $\text{sKey}[\text{sID}] \neq \bot$
50 $\quad$ **return** $\bot$
51 $(i,r) \coloneqq (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
52 $(\widetilde{\mathsf{pk}}, \cdot, c_r, K_r) \coloneqq \text{state}'[\text{sID}]$
53 $(\tilde{c}, c_i) \coloneqq R$
54 **if** $\exists \widetilde{K}'$ s. t. $(\tilde{c}, \widetilde{K}') \in \widetilde{\mathcal{CK}}_{\text{sID}}$
55 $\quad\widetilde{K} \coloneqq \widetilde{K}'$
56 **else**
57 $\quad\widetilde{\mathsf{sk}} \coloneqq \textsc{Open}(\text{sID})$
58 $\quad\widetilde{K} \coloneqq \mathsf{Decaps}_{\mathsf{CPA}}^{\widetilde{\mathsf{H}}_{\text{sID}}}(\widetilde{\mathsf{sk}}, \tilde{c})$
59 **if** $\exists K'_i$ s. t. $(c_i, K'_i) \in \mathcal{CK}_i$
60 $\quad K_i \coloneqq K'_i$
61 **else**
62 $\quad K_i \coloneqq \mathsf{Decaps}_{\mathsf{CCA}}^{\mathsf{H}_i}(\mathsf{sk}_i, c_i)$
63 $\quad\mathcal{D}_i \coloneqq \mathcal{D}_i \cup \{c_i\}$
64 $\text{context} \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
65 $K \coloneqq \mathsf{H}(\text{context}, K_i, K_r, \widetilde{K})$
66 $(R[\text{sID}], \text{sKey}[\text{sID}]) \coloneqq (R, K)$
67 **return** ε

Figure 17: Adversaries $\mathcal{C}_b$ against S-NCKE-CPA for the proof of Eqn. (4). $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, \ldots, \mathsf{H}_N, \widetilde{\mathsf{H}}_1, \ldots, \widetilde{\mathsf{H}}_S\}$, where $\textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{H}$ and H_n for $n \in [N]$ are defined as in Figure 14 resp. 15. $\textsc{Rev-State}$, G and $\widetilde{\mathsf{H}}_{\text{sID}}$ for $\text{sID} \in [S]$ are defined in Figure 18. Lines written in blue color highlight how the adversary simulates $G_{2,b}$ and interpolates to $G_{3,b}$.

If $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{real}}$ game, it perfectly simulates $G_{2,b}$. Otherwise, if $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{3,b}$. We have

$$\begin{aligned}\left|\Pr[G^{\mathcal{A}}_{3,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{2,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{C}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{C}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}_b)\ . \end{aligned} \tag{4}$$

$\underline{\mathsf{G}(k, IV)}$
00 **if** $\exists \bar{k}, IV$ s. t. $(k, IV, y) \in \mathcal{G}$ **return** y
01 **if** $\exists i$ s. t. $k = k_i$ **and**
$\exists(\text{sID}, \psi)$ s. t. state[sID] $= (IV, \psi)$
$\wedge$ revState[sID] $=$ **true**
02 $\widetilde{\mathsf{sk}} \coloneqq \textsc{Open}(\text{sID})$
03 state$'$[sID] $\coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r)$
04 $y \coloneqq \psi \oplus$ state$'$[sID]
05 **else**
06 $y \xleftarrow{\$} \{0,1\}^d$
07 $\mathcal{G} \coloneqq \mathcal{G} \cup \{(k, IV, y)\}$
08 **return** y

$\underline{\widetilde{\mathsf{H}}_{\text{sID}}(M)}$ //sID $\in [S]$
09 **if** $\exists h$ s. t. $(M, h) \in \widetilde{\mathcal{H}}_{\text{sID}}$ **return** h
10 **if** type[sID] $=$ "In"
11 $h \leftarrow \widetilde{\mathsf{H}}'_{\text{sID}}(M)$
12 **else**
13 $h \xleftarrow{\$} \{0,1\}^\kappa$
14 $\widetilde{\mathcal{H}}_{\text{sID}} \coloneqq \widetilde{\mathcal{H}}_{\text{sID}} \cup \{(M, h)\}$
15 **return** h

$\underline{\textsc{Rev-State}(\text{sID})}$
16 **if** revState[sID] $=$ **true**
17 **return** state[sID]
18 **if** type[sID] $\neq$ "In" **return** $\bot$
19 revState[sID] $\coloneqq$ **true**
20 $(IV, \bot) \coloneqq$ state[sID]
21 $i \coloneqq$ init[sID]
22 **if** corrupted[i]
23 $\widetilde{\mathsf{sk}} \coloneqq \textsc{Open}(\text{sID})$
24 state$'$[sID] $\coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r)$
25 state[sID] $\coloneqq (IV, \mathsf{G}(IV, k_i) \oplus$ state$'$[sID]$)$
26 **else if** $\exists y$ s. t. $(k_i, IV, y) \in \mathcal{G}$
27 BAD $\coloneqq$ **true**
28 **abort**
29 **else**
30 $\psi \xleftarrow{\$} \{0,1\}^d$
31 state[sID] $\coloneqq (IV, \psi)$
32 **return** state[sID]

Figure 18: Oracles G, $\widetilde{\mathsf{H}}_{\text{sID}}$ for sID $\in [S]$ and Rev-State for adversaries $\mathcal{C}_b$ against S-NCKE-CPA in Figure 17.

Games $G_{4,b}$. In games $G_{4,b}$, we change the output for K_0^* in the Test oracle to a random key in line 49. Now games $G_{4,0}$ and $G_{4,1}$ are equal as well as games $G_{4,1}$ and $G_{3,1}$, hence

$$\begin{aligned} \left|\Pr[G_{3,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{3,0}^{\mathcal{A}} \Rightarrow 1]\right| &= \left|\Pr[G_{4,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{3,0}^{\mathcal{A}} \Rightarrow 1]\right| \\ &= \left|\Pr[G_{4,0}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{3,0}^{\mathcal{A}} \Rightarrow 1]\right| \ . \end{aligned} \tag{5}$$

It remains to bound $\left|\Pr[G_{4,0}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{3,0}^{\mathcal{A}} \Rightarrow 1]\right|$. Therefore, we will now consider the different attacks for IND-wFS-St as described in Table 1. Depending on which queries the adversary makes, each test session must belong to at least one of the attacks or the game will return 0 anyway. For the analysis, we consider the worst case scenario where the adversary queries as much information as it is allowed to obtain.

When referring to a particular test session sID*, we will denote all values used with an asterisk, i. e., context$^* = (\mathsf{pk}_{i^*}, \mathsf{pk}_{r^*}, \widetilde{\mathsf{pk}}^*, c_{i^*}, c_{r^*}, \tilde{c}^*)$ and $IV^*, k_{i^*}, K_{i^*}, K_{r^*}, \widetilde{K}^*$. As we assumed in the beginning that ciphertexts and long-term as well as ephemeral key pairs are all different, it is not possible to recreate a particular session. In particular, this means that there is no partially matching session and attack (0) of Table 1 will return false.

Now the only possibility to learn any test key K_0^* is through random oracle queries. Let QUERY be the event that $(\text{context}, K_i, K_r, \widetilde{K})$ of any test session is queried to H and QUERY* be the event that $(\text{context}^*, K_{i^*}, K_{r^*}, \widetilde{K}^*)$ of a specific test session is queried to H. We have

$$\left|\Pr[G_{4,0}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{3,0}^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\text{QUERY}] \leq T \cdot \Pr[\text{QUERY}^*] \ ,$$

where the last inequality uses union bound over the number of test sessions T.

Attack	Security relies on ...
$(1 \vee 2)$, (10)	IV^* unknown $\Rightarrow \widetilde{\mathsf{sk}}^*$ unknown $\Rightarrow \widetilde{K}^*$ unknown
$(7 \vee 8)$, (16)	sk_{i^*} unknown $\Rightarrow K_{i^*}$ unknown
(19)	sk_{r^*} and k_{i^*} unknown $\Rightarrow K_{r^*}$ unknown
(21)	sk_{r^*} and IV^* unknown $\Rightarrow K_{r^*}$ unknown
(24)	sk_{i^*} unknown $\Rightarrow K_{i^*}$ unknown

Figure 19: Overview of attacks for the proof of Theorem 2.

We will now focus on the event QUERY^* and iterate over the attacks in Table 1. An overview on the argumentation is given in Figure 19.

Attack $(1 \vee 2)$, (10). If $(1 \vee 2) \Rightarrow$ **true**, the test session has a matching session and both long-term secret keys $(\mathsf{sk}_{i^*}, k_{i^*})$ and $(\mathsf{sk}_{r^*}, k_{r^*})$ are revealed. Hence, $\mathcal{A}$ can compute K_{i^*} and K_{r^*}. However, $\mathcal{A}$ is not allowed to query the test session's state or the state of the matching session, depending on the type of the test session. Thus, $\mathcal{A}$ has no information about $\widetilde{\mathsf{sk}}^*$. As there is a matching session for this test session, $\widetilde{\mathsf{pk}}^*$ was generated by SESSION_I, which means that $\widetilde{K}^*$ is chosen uniformly at random and thus independent of $\widetilde{\mathsf{pk}}^*$ and $\tilde{c}^*$.

If (10) $\Rightarrow$ **true**, the test session has a partially matching and is of type "Re". Here, $\mathcal{A}$ is not allowed to query the state of the partially matching session. Except for that, everything remains the same as explained above. Hence,

$$\Pr[\text{QUERY}^* \mid (1 \vee 2) \Rightarrow \textbf{true}] = \Pr[\text{QUERY}^* \mid (10) \Rightarrow \textbf{true}] \leq \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

Attack $(7 \vee 8)$, (16). If $(7 \vee 8) \Rightarrow$ **true**, the test session has a matching session and the state (IV^*, ψ^*) is revealed. Furthermore, $\mathcal{A}$ can obtain $(\mathsf{sk}_{r^*}, k_{r^*})$ and thus K_{r^*}. As secret key sk_{i^*} is unknown to $\mathcal{A}$, K_{i^*} which is chosen uniformly at random from the key space of $\mathsf{KEM}_{\mathsf{CCA}}$ is also unknown to $\mathcal{A}$. $\widetilde{K}^*$ is also chosen uniformly at random and unknown as long as $\mathcal{A}$ does not obtain $\widetilde{\mathsf{sk}}^*$ through queries to G or guesses $\widetilde{K}^*$ correctly. However, to bound event QUERY^*, we will only make use of the fact that K_{i^*} is a uniformly random key.

If (16) $\Rightarrow$ **true**, the test session has a partially matching session and is of type "Re". Here, $\mathcal{A}$ can reveal the state of the partially matching session (IV, ψ). The rest remains the same. It follows that

$$\Pr[\text{QUERY}^* \mid (7 \vee 8) \Rightarrow \textbf{true}] = \Pr[\text{QUERY}^* \mid (16) \Rightarrow \textbf{true}] \leq \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

Attack (19). If (19) $\Rightarrow$ **true**, the test session has no matching session and the type of this test session is "In". $\mathcal{A}$ is allowed to obtain the initiator's state (IV^*, ψ^*) and can choose $(\tilde{c}^*, \widetilde{K}^*)$ and (c_{i^*}, K_{i^*}) itself. K_{r^*} is unknown to $\mathcal{A}$, unless it obtains it through queries to G. Therefore, $\mathcal{A}$ can either guess k_{i^*} and query G or it can query H directly. Hence,

$$\Pr[\text{QUERY}^* \mid (19) \Rightarrow \textbf{true}] \leq \frac{q_\mathsf{G}}{2^\kappa} + \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

ATTACK (21). If (21) $\Rightarrow$ **true**, the test session has no matching session and the type of this test session is "In". $\mathcal{A}$ is allowed to obtain the initiator's long-term secret key $(\mathsf{sk}_{i^*}, k_{i^*})$ and can choose $(\tilde{c}^*, \widetilde{K}^*)$ and (c_{i^*}, K_{i^*}) itself. However, K_r^* is unknown to $\mathcal{A}$ and as it is chosen uniformly at random from the key space of $\mathsf{KEM}_{\mathsf{CCA}}$, $(\text{context}^*, K_{i^*}, K_{r^*}, \widetilde{K}^*)$ is queried to H with probability at most $q_{\mathsf{H}}/|\mathcal{K}|$. This yields

$$\Pr[\text{QUERY}^* \mid (21) \Rightarrow \textbf{true}] \leq \frac{q_{\mathsf{H}}}{|\mathcal{K}|} \ .$$

ATTACK (24). If (24) $\Rightarrow$ **true**, the test session has no matching session and the type of the test session is "Re", which means that $\mathcal{A}$ can reveal the responder's long-term secret key $(\mathsf{sk}_{r^*}, k_{r^*})$. Here, $\mathcal{A}$ can choose $(\widetilde{\mathsf{pk}}^*, \widetilde{\mathsf{sk}}^*)$ and (c_{r^*}, K_{r^*}) itself and thus is able to compute $\widetilde{K}^*$. As $(\mathsf{sk}_{i^*}, k_{i^*})$ is unknown, so is K_{i^*} and we have

$$\Pr[\text{QUERY}^* \mid (24) \Rightarrow \textbf{true}] \leq \frac{q_{\mathsf{H}}}{|\mathcal{K}|} \ .$$

Taking the maximum over the conditional probabilities, it follows that

$$\left|\Pr[G_{4,0}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{3,0}^{\mathcal{A}} \Rightarrow 1]\right| \quad \leq \quad T \cdot \Pr[\text{QUERY}^*] \leq T \cdot \left(\frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right) \ . \qquad (6)$$

Finally, folding both adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$ into one adversary $\mathcal{B}$ and $\mathcal{C}_0$ and $\mathcal{C}_1$ into one adversary $\mathcal{C}$ yields

$$\begin{aligned} \mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}_0) + \mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}_1) &= 2 \cdot \mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}) \text{ and} \\ \mathrm{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}_0) + \mathrm{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}_1) &= 2 \cdot \mathrm{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}) \ . \end{aligned}$$

The proof of Theorem 2 follows by collecting the probabilities from Eqns. (2)-(6). □

Note that the non-committing property is essential to embed random KEM keys in each session and thus to achieve tightness. This way, we only need to make a case distinction at the end and can argue that for all test sessions at least one KEM key is independent of the adversary's view no matter which queries it has made (provided it did not make a trivial attack). Relying on a weaker assumption requires to make a case distinction earlier in the proof and may involve guessing as in some cases it is not clear which KEM key will be revealed (through corruption and/or reveal or state reveal) at a later point in time.

6 AKE with Full Forward Security

We show how to build an explicitly authenticated AKE protocol using the concept of non-committing key encapsulation. As we also need a signature scheme, we will first give the relevant definitions.

6.1 Digital Signatures

A digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}_{\mathsf{SIG}}, \mathsf{Sign}, \mathsf{Vrfy})$ consists of three algorithms. The key generation algorithm $\mathsf{Gen}_{\mathsf{SIG}}$ outputs a key pair $(\mathsf{vk}, \mathsf{sigk})$, where vk is the verification key and sigk the signing key. The signing algorithm Sign inputs a signing key sigk and a message m and outputs a signature σ. The deterministic verification algorithm Vrfy inputs the verification key vk, a message m and a signature σ and outputs 1 if σ is a valid signature for m, otherwise it outputs 0.

In Figure 20, we define the security game N user Strong UnForgeability under Chosen Message Attacks with corruptions (N-SUF-CMA). The definition is similar to the one given in [BHJ+15], except that we require *strong* unforgeability, i. e., the adversary may also find a new signature for a message it queried to the Sign oracle before. The advantage of an adversary $\mathcal{A}$ is defined as

$$\mathrm{Adv}^{N\text{-SUF-CMA}}_{\mathsf{SIG}}(\mathcal{A}) := \Pr[N\text{-SUF-CMA}^{\mathcal{A}} \Rightarrow 1] \ .$$

GAME N-SUF-CMA

00 $\mathcal{S}^{\mathsf{corr}} := \emptyset$
01 **for** $n \in [N]$
02 $\quad (\mathsf{vk}_n, \mathsf{sigk}_n) \leftarrow \mathsf{Gen}_{\mathsf{SIG}}$
03 $\quad \mathcal{S}_n := \emptyset$
04 $(n^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\textsc{Sign},\textsc{Corrupt}}(\mathsf{vk}_1, \cdots, \mathsf{vk}_N)$
05 **if** $\mathsf{Vrfy}(\mathsf{vk}_{n^*}, m^*, \sigma^*) = 1$ **and** $n^* \notin \mathcal{S}^{\mathsf{corr}}$ **and** $(m^*, \sigma^*) \notin \mathcal{S}_{n^*}$
06 $\quad$ **return** 1
07 **else**
08 $\quad$ **return** 0

$\textsc{Sign}(n \in [N], m)$
09 $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, m)$
10 $\mathcal{S}_n := \mathcal{S}_n \cup \{(m, \sigma)\}$
11 **return** σ

$\textsc{Corrupt}(n \in [N])$
12 $\mathcal{S}^{\mathsf{corr}} := \mathcal{S}^{\mathsf{corr}} \cup \{n\}$
13 **return** sigk_n

Figure 20: Game N-SUF-CMA for SIG.

6.2 Transformation using NCKE and a Signature Scheme

From two key encapsulation mechanisms $\mathsf{KEM}_{\mathsf{CPA}} = (\mathsf{Gen}_{\mathsf{CPA}}, \mathsf{Encaps}_{\mathsf{CPA}}, \mathsf{Decaps}_{\mathsf{CPA}})$ and $\mathsf{KEM}_{\mathsf{CCA}} = (\mathsf{Gen}_{\mathsf{CCA}}, \mathsf{Encaps}_{\mathsf{CCA}}, \mathsf{Decaps}_{\mathsf{CCA}})$ with key space $\mathcal{K}$ and a digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}_{\mathsf{SIG}}, \mathsf{Sign}, \mathsf{Vrfy})$, we construct a two-message authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{FS}} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{R}}, \mathsf{Der}_{\mathrm{I}})$ with key space $\mathcal{K}$ as shown in Figures 21 and 22. Each party has a key pair $(\mathsf{vk}, \mathsf{sigk})$ for SIG, a key pair $(\mathsf{pk}, \mathsf{sk})$ for $\mathsf{KEM}_{\mathsf{CCA}}$ and a symmetric key k to encrypt the secret state information which has to be stored by the initiating party (cf. Section 5). The protocol uses additional cryptographic hash functions $\mathsf{F} : \{0,1\}^* \to \{0,1\}^\kappa$ to compute value π and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ to output the session key.

The initiating party computes an ephemeral key pair for $\mathsf{KEM}_{\mathsf{CPA}}$, runs the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the intended receiver's public key pk_r to obtain a ciphertext c_r and a key K_r and signs both the ephemeral public key and c_r, which are sent to the receiver along with the signature. The receiver verifies the signature and then runs the $\mathsf{Encaps}_{\mathsf{CPA}}$ algorithm on the ephemeral public key to output a ciphertext $\tilde{c}$ and a key $\widetilde{K}$. It computes

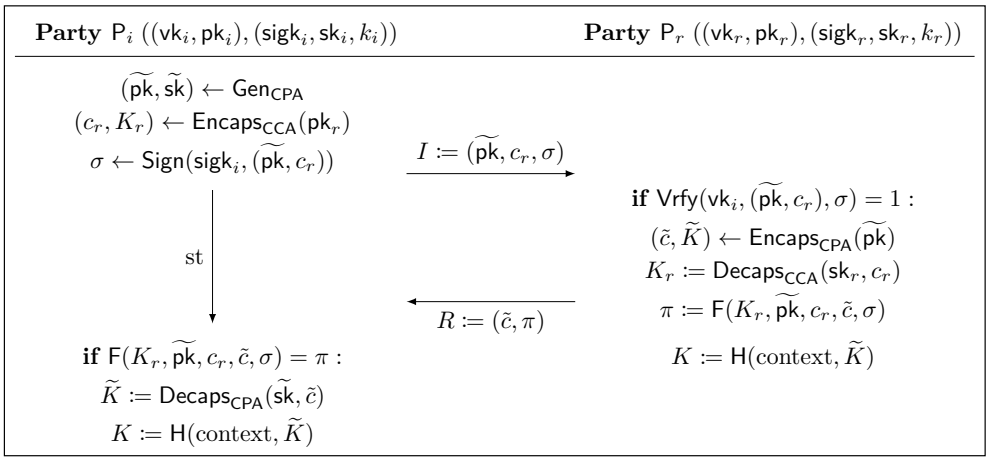

Figure 21: Visualization: Running $\mathsf{AKE}_{\mathsf{FS}}$ between two parties, where K is the resulting session key and context $:= (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$

K_r using its secret key sk_r. It then tags the received message together with $\tilde{c}$ and K_r by evaluating hash function F and sends the output together with $\tilde{c}$ to the initiator. The initiator retrieves K_r from the secret state and also evaluates F. If the output is the same, it computes $\widetilde{K}$ using the ephemeral secret key. The session key is computed evaluating hash function H on all public context and key $\widetilde{K}$.

Theorem 3 ($\mathsf{KEM}_{\mathsf{CPA}}$ NCKE-CPA+$\mathsf{KEM}_{\mathsf{CCA}}$ NCKE-CCA+SIG N-SUF-CMA $\overset{\text{ROM}}{\Rightarrow}$ $\mathsf{AKE}_{\mathsf{FS}}$ IND-FS-St)**.** *For any* IND-FS-St *adversary* $\mathcal{A}$ *against* $\mathsf{AKE}_{\mathsf{FS}}$ *with* N *parties that establishes at most* S *sessions and issues at most* T *queries to test oracle* TEST*, at most* q_{H}*,* q_{G} *and* q_{F} *queries to random oracles* H*,* G *and* F*, there exists an* N-SUF-CMA *adversary* $\mathcal{B}$ *against* SIG*, an* S-NCKE-CPA *adversary* $\mathcal{C}$ *against* $\mathsf{KEM}_{\mathsf{CPA}}$ *and* $\mathsf{Sim}_{\mathsf{CPA}}$ *and an* N-NCKE-CCA *adversary* $\mathcal{D}$ *against* $\mathsf{KEM}_{\mathsf{CCA}}$ *and* $\mathsf{Sim}_{\mathsf{CCA}}$ *such that*

$$\begin{aligned}\mathrm{Adv}^{\text{IND-FS-St}}_{\mathsf{AKE}_{\mathsf{FS}}}(\mathcal{A}) \leq 2 \cdot \Big(\mathrm{Adv}^{N\text{-SUF-CMA}}_{\mathsf{SIG}}(\mathcal{B}) + \mathrm{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}) + \mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{D})\Big) \\ + T \cdot \left(\frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right) + N^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{SIG}}}} + \frac{1}{2^{\mu_{\mathsf{CCA}}}} + \frac{1}{2^{\kappa}}\right) \\ + S^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{CPA}}}} + \frac{1}{2^{\gamma_{\mathsf{CCA}}}} + \frac{1}{2^{\gamma_{\mathsf{CPA}}}} + \frac{1}{2^{\kappa}}\right) + 2S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} \ ,\end{aligned}$$

where $\mathsf{Sim}_{\mathsf{CPA}}$ *and* $\mathsf{Sim}_{\mathsf{CCA}}$ *are the simulators from the* NCKE-CPA *and* NCKE-CCA *experiment,* μ_{SIG}*,* μ_{CPA}*,* μ_{CCA} *are collision probabilities of the key generation algorithms* $\mathsf{Gen}_{\mathsf{SIG}}$*,* $\mathsf{Gen}_{\mathsf{CPA}}$ *and* $\mathsf{Gen}_{\mathsf{CCA}}$ *and* γ_{CPA}*,* γ_{CCA} *are the spreadness parameters of the encapsulation algorithms. The running times of* $\mathcal{B}$*,* $\mathcal{C}$ *and* $\mathcal{D}$ *consist essentially of the time required to execute the security experiment with the adversary once, plus a minor number of additional operations (including bookkeeping, lookups etc.).*

Proof. Let $\mathcal{A}$ be an adversary against IND-FS-St security of $\mathsf{AKE}_{\mathsf{FS}}$, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is

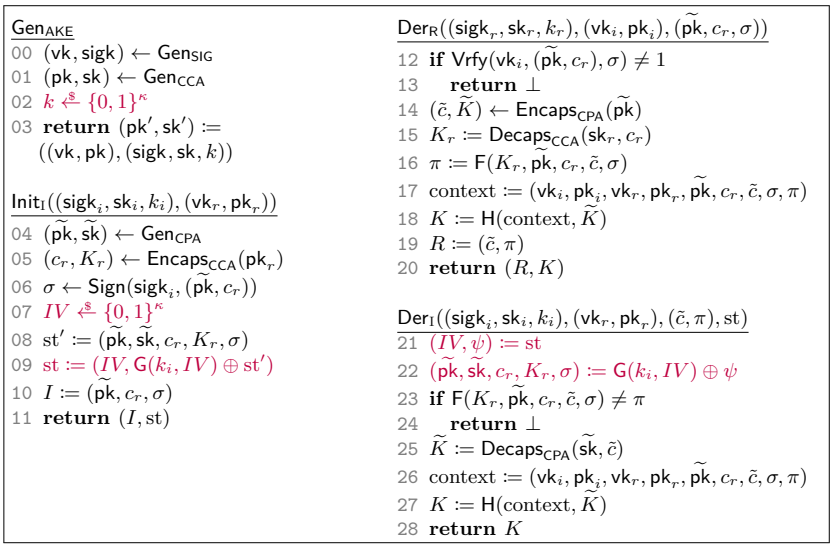

Figure 22: Authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{FS}}$ from $\mathsf{KEM}_{\mathsf{CPA}}$, $\mathsf{KEM}_{\mathsf{CCA}}$ and SIG. Lines written in purple color are only used to encrypt the state.

the maximum number of test sessions. Consider the sequence of games in Figures 23 and 24.

GAMES $G_{0,b}$. These are the original $\mathsf{IND\text{-}FS\text{-}St}_b$ games. Similar to Equation (2) in the proof of Theorem 2, we assume all key pairs, N long-term keys generated by $\mathsf{Gen}_{\mathsf{SIG}}$ and $\mathsf{Gen}_{\mathsf{CCA}}$ as well as ephemeral keys (at most S) generated by $\mathsf{Gen}_{\mathsf{CPA}}$, and all ciphertexts (at most S) output by the $\mathsf{Encaps}_{\mathsf{CPA}}$ and $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithms to be distinct. We also assume that values k_n, $n \in [N]$, and IV (at most S) are distinct. This yields

$$\begin{aligned}\left|\Pr[\mathsf{IND\text{-}FS\text{-}St}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{IND\text{-}FS\text{-}St}_0^{\mathcal{A}} \Rightarrow 1]\right| \;&\leq \left|\Pr[G_{0,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{0,0}^{\mathcal{A}} \Rightarrow 1]\right| \\ &+ N^2\left(2^{-\mu_{\mathsf{SIG}}} + 2^{-\mu_{\mathsf{CCA}}} + 2^{-\kappa}\right) \\ &+ S^2\left(2^{-\mu_{\mathsf{CPA}}} + 2^{-\gamma_{\mathsf{CCA}}} + 2^{-\gamma_{\mathsf{CPA}}} + 2^{-\kappa}\right)\;,\end{aligned} \tag{7}$$

where μ_{SIG}, μ_{CPA}, μ_{CCA} are collision probabilities of the key generation algorithms $\mathsf{Gen}_{\mathsf{SIG}}$, $\mathsf{Gen}_{\mathsf{CPA}}$ and $\mathsf{Gen}_{\mathsf{CCA}}$ and γ_{CPA}, γ_{CCA} are the spreadness parameters of $\mathsf{Encaps}_{\mathsf{CPA}}$ and $\mathsf{Encaps}_{\mathsf{CCA}}$.

GAMES $G_{1,b}$. In games $G_{1,b}$, we raise flag $\mathsf{BREAK}_{\mathsf{SIG}}$ in line 22 (Fig. 24) and abort when $\mathcal{A}$ has not queried CORRUPT to obtain sigk_i yet and has not only forwarded a message and a signature output by $\mathrm{SESSION}_{\mathrm{I}}$. Note that if the game aborts, the signature must be valid because this is checked before in line 19. Due to the difference lemma [Sho04],

$$\left|\Pr[G_{1,b}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{0,b}^{\mathcal{A}} \Rightarrow 1]\right| \;\leq\; \Pr[\mathsf{BREAK}_{\mathsf{SIG}}]\;. \tag{8}$$

To bound $\Pr[\mathsf{BREAK}_{\mathsf{SIG}}]$, we construct adversaries $\mathcal{B}_b$ for $b \in \{0,1\}$ against N-$\mathsf{SUF\text{-}CMA}$ security of SIG in Figure 25.

GAMES $G_{0,b}$-$G_{5,b}$

00 $\mathrm{cnt} \coloneqq 0$
01 $\mathcal{S} \coloneqq \emptyset$
02 **for** $n \in [N]$
03 $\quad(\mathsf{vk}_n, \mathsf{sigk}_n) \leftarrow \mathsf{Gen}_{\mathsf{SIG}}$
04 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$ // $G_{0\text{-}1}$
05 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{SimGen}_{\mathsf{CCA}}$ // $G_{2\text{-}5}$
06 $\quad k_n \xleftarrow{\$} \{0,1\}^\kappa$
07 $\quad(\mathsf{pk}'_n, \mathsf{sk}'_n) \coloneqq ((\mathsf{vk}_n, \mathsf{pk}_n), (\mathsf{sigk}_n, \mathsf{sk}_n, k_n))$
08 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}'_1, ..., \mathsf{pk}'_N)$
09 **for** $\mathrm{sID}^* \in \mathcal{S}$
10 $\quad$ **if** $\textsc{Fresh}(\mathrm{sID}^*) = \textbf{false}$ **return** 0
11 $\quad$ **if** $\textsc{Valid}(\mathrm{sID}^*) = \textbf{false}$ **return** 0
12 **return** b'

$\underline{\textsc{Session}_{\mathrm{R}}((i,r) \in [N]^2, I)}$
13 cnt ++
14 $\mathrm{sID} \coloneqq \mathrm{cnt}$
15 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) \coloneqq (i, r)$
16 $\mathrm{type}[\mathrm{sID}] \coloneqq$ "Re"
17 $\mathrm{peerCorrupted}[\mathrm{sID}] \coloneqq \mathrm{corrupted}[i]$
18 $(\widetilde{\mathsf{pk}}, c_r, \sigma) \coloneqq I$
19 **if** $\mathsf{Vrfy}(\mathsf{vk}_i, (\widetilde{\mathsf{pk}}, c_r), \sigma) \neq 1$
20 $\quad$ **return** $\bot$
21 **if** $\mathrm{peerCorrupted}[\mathrm{sID}] = \textbf{false}$ **and** $\nexists \mathrm{sID}'$ s. t. $(\mathrm{init}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], I[\mathrm{sID}']) = (i, \text{"In"}, I)$ // $G_{1\text{-}5}$
22 $\quad\mathsf{BREAK}_{\mathsf{SIG}} \coloneqq \textbf{true}$ // $G_{1\text{-}5}$
23 $\quad$ **abort** // $G_{1\text{-}5}$
24 $(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$ // $G_{0\text{-}3}$
25 **if** $\exists \mathrm{sID}'$ s. t. $\mathrm{state}'[\mathrm{sID}'] = (\widetilde{\mathsf{pk}}, \cdot, \cdot, \cdot, \cdot)$ // $G_{4\text{-}5}$
26 $\quad(\cdot, \widetilde{\mathsf{sk}}, \cdot, \cdot, \cdot) \coloneqq \mathrm{state}'[\mathrm{sID}']$ // $G_{4\text{-}5}$
27 $\quad\tilde{c} \leftarrow \mathsf{SimEncaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ // $G_{4\text{-}5}$
28 $\quad\widetilde{K} \xleftarrow{\$} \mathcal{K}$ // $G_{4\text{-}5}$
29 $\quad\widetilde{\mathcal{C}}_{\mathrm{sID}'} \coloneqq \widetilde{\mathcal{C}}_{\mathrm{sID}'} \cup \{(\tilde{c}, \bot)\}$ // $G_{4\text{-}5}$
30 $\quad\widetilde{\mathcal{CK}}_{\mathrm{sID}'} \coloneqq \widetilde{\mathcal{CK}}_{\mathrm{sID}'} \cup \{(\tilde{c}, \widetilde{K})\}$ // $G_{4\text{-}5}$
31 **else** // $G_{4\text{-}5}$
32 $\quad(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$ // $G_{4\text{-}5}$
33 $K_r \coloneqq \mathsf{Decaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$ // $G_{0\text{-}1}$
34 **if** $\exists K'_r$ s. t. $(c_r, K'_r) \in \mathcal{CK}_r$ // $G_{2\text{-}5}$
35 $\quad K_r \coloneqq K'_r$ // $G_{2\text{-}5}$
36 **else** // $G_{2\text{-}5}$
37 $\quad K_r \coloneqq \mathsf{Decaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$ // $G_{2\text{-}5}$
38 $\quad\mathcal{D}_r \coloneqq \mathcal{D}_r \cup \{c_r\}$ // $G_{2\text{-}5}$
39 $\pi \coloneqq \mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma)$
40 $\mathrm{context} \coloneqq (\mathsf{vk}_i, \mathsf{pk}_{\vec{i}}, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$
41 $K \coloneqq \mathsf{H}(\mathrm{context}, \widetilde{K})$
42 $R \coloneqq (\tilde{c}, \pi)$
43 $(I[\mathrm{sID}], R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) \coloneqq (I, R, K)$
44 **return** (sID, R)

$\underline{\textsc{Session}_{\mathrm{I}}((i,r) \in [N]^2)}$
45 cnt ++
46 $\mathrm{sID} \coloneqq \mathrm{cnt}$
47 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) \coloneqq (i, r)$
48 $\mathrm{type}[\mathrm{sID}] \coloneqq$ "In"
49 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$ // $G_{0\text{-}3}$
50 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{SimGen}_{\mathsf{CPA}}$ // $G_{4\text{-}5}$
51 $(c_r, K_r) \leftarrow \mathsf{Encaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{pk}_r)$ // $G_{0\text{-}1}$
52 $c_r \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_r, \mathsf{sk}_r)$ // $G_{2\text{-}5}$
53 $K_r \xleftarrow{\$} \mathcal{K}$ // $G_{2\text{-}5}$
54 $\mathcal{C}_r \coloneqq \mathcal{C}_r \cup \{(c_r, \bot)\}$
55 $\mathcal{CK}_r \coloneqq \mathcal{CK}_r \cup \{(c_r, K_r)\}$
56 $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_i, (\widetilde{\mathsf{pk}}, c_r))$
57 $I \coloneqq (\widetilde{\mathsf{pk}}, c_r, \sigma)$
58 $IV \xleftarrow{\$} \{0,1\}^\kappa$
59 $\mathrm{st}' \coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r, \sigma)$
60 $\mathrm{st} \coloneqq (IV, \mathsf{G}(k_i, IV) \oplus \mathrm{st}')$ // $G_{0\text{-}2}$
61 $\mathrm{st} \coloneqq (IV, \bot)$ // $G_{3\text{-}5}$
62 $(I[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) \coloneqq (I, \mathrm{st})$
63 $\mathrm{state}'[\mathrm{sID}] \coloneqq \mathrm{st}'$
64 **return** (sID, I)

$\underline{\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, R)}$
65 **if** $\mathrm{state}[\mathrm{sID}] = \bot$ **or** $\mathrm{sKey}[\mathrm{sID}] \neq \bot$
66 $\quad$ **return** $\bot$
67 $(i, r) \coloneqq (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
68 $\mathrm{peerCorrupted}[\mathrm{sID}] \coloneqq \mathrm{corrupted}[r]$
69 $(\tilde{c}, \pi) \coloneqq R$
70 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r, \sigma) \coloneqq \mathrm{state}'[\mathrm{sID}]$
71 **if** $\mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma) \neq \pi$
72 $\quad$ **return** $\bot$
73 $\widetilde{K} \coloneqq \mathsf{Decaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$ // $G_{0\text{-}3}$
74 **if** $\exists \widetilde{K}'$ s. t. $(\tilde{c}, \widetilde{K}') \in \widetilde{\mathcal{CK}}_{\mathrm{sID}}$ // $G_{4\text{-}5}$
75 $\quad\widetilde{K} \coloneqq \widetilde{K}'$ // $G_{4\text{-}5}$
76 **else** // $G_{4\text{-}5}$
77 $\quad\widetilde{K} \coloneqq \mathsf{Decaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$ // $G_{4\text{-}5}$
78 $\mathrm{context} \coloneqq (\mathsf{vk}_i, \mathsf{pk}_{\vec{i}}, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$
79 $K \coloneqq \mathsf{H}(\mathrm{context}, \widetilde{K})$
80 $(R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) \coloneqq (R, K)$
81 **return** ε

$\underline{\mathsf{H}(x)}$
82 **if** $\exists K$ s. t. $(x, K) \in \mathcal{H}$
83 $\quad$ **return** K
84 $K \xleftarrow{\$} \mathcal{K}$
85 $\mathcal{H} \coloneqq \mathcal{H} \cup \{(x, K)\}$
86 **return** K

Figure 23: Games $G_{0,b}$-$G_{5,b}$ for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, ..., \mathsf{H}_N, \widetilde{\mathsf{H}}_1, ..., \widetilde{\mathsf{H}}_S\}$, where Corrupt and Reveal are defined as in the original IND-FS-St game (Fig. 10) and oracles Rev-State, F, G, H_n for $n \in [N]$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for $\mathrm{sID} \in [S]$ and Test are defined in Figure 24.

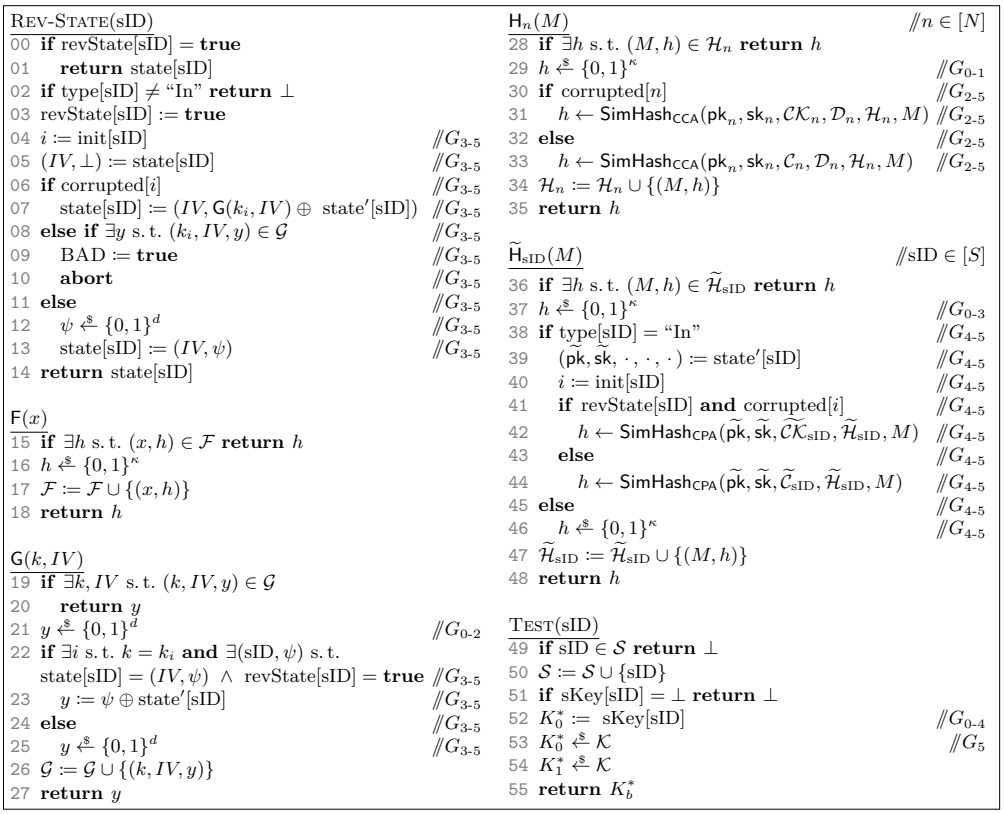

Figure 24: Oracles REV-STATE, F, G, H_n for $n \in [N]$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ and TEST for sID $\in [S]$ for games $G_{0,b}$-$G_{5,b}$ in Fig. 23.

$\mathcal{B}_b$ inputs N verification keys $(\mathsf{vk}_1, ..., \mathsf{vk}_N)$ and has access to signing oracle SIGN and corruption oracle CORRUPT′. It then generates N key pairs for $\mathsf{KEM}_{\mathsf{CCA}}$ and N symmetric keys k_n which are part of the long-term secret key. It forwards the public keys to adversary $\mathcal{A}$. Whenever $\mathcal{A}$ queries $\mathrm{SESSION}_{\mathrm{I}}$ on a pair (i, r), $\mathcal{B}_b$ queries SIGN on user i in line 37 to obtain a signature σ to message $(\widetilde{\mathsf{pk}}, c_r)$. It then outputs $(\widetilde{\mathsf{pk}}, c_r, \sigma)$ to $\mathcal{A}$.

To answer a CORRUPT query on user n, $\mathcal{B}_b$ queries its own oracle CORRUPT′ to obtain signing key sigk_n in line 46 and outputs secret keys sigk_n, sk_n and k_n to $\mathcal{A}$.

Recall that flag $\mathsf{BREAK}_{\mathsf{SIG}}$ is raised when $\mathcal{A}$ has not queried CORRUPT to obtain sigk_i, but computes a valid signature which was not output by $\mathrm{SESSION}_{\mathrm{I}}$. If $\mathcal{A}$ queries $\mathrm{SESSION}_{\mathsf{R}}$ on pair (i, r) and $I = (\widetilde{\mathsf{pk}}, c_r, \sigma)$ such that this is the case, $\mathcal{B}_b$ wins the N-SUF-CMA game by returning $(i, (\widetilde{\mathsf{pk}}, c_r), \sigma)$. It follows that

$$\Pr[\mathsf{BREAK}_{\mathsf{SIG}}] \quad = \quad \mathrm{Adv}^{N\text{-}\mathsf{SUF}\text{-}\mathsf{CMA}}_{\mathsf{SIG}}(\mathcal{B}_b) \ . \tag{9}$$

GAMES $G_{2,b}$. In games $G_{2,b}$, we use the $\mathsf{SimGen}_{\mathsf{CCA}}$ algorithm to generate long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$ in line 05 (Fig. 23). Next, $\mathrm{SESSION}_{\mathrm{I}}$ uses the $\mathsf{SimEncaps}_{\mathsf{CCA}}$ algorithm

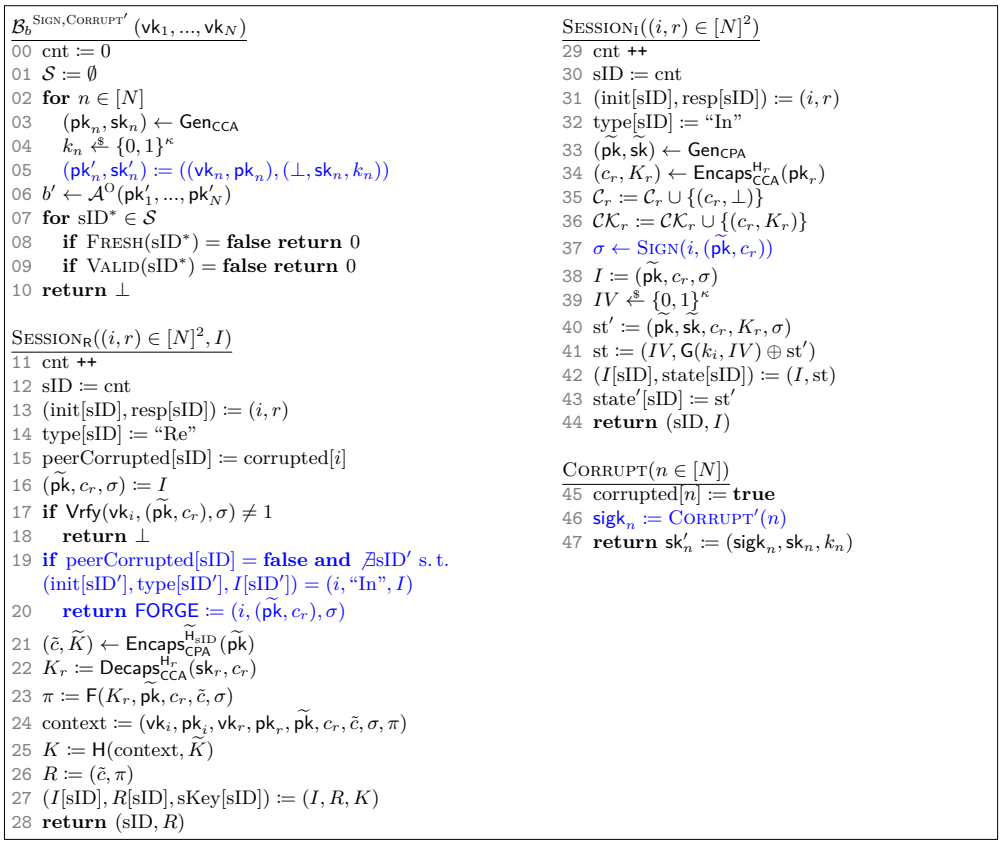

Figure 25: Adversaries $\mathcal{B}_b$ against N-SUF-CMA for the proof of Eqn. (9). $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{Session}_{\mathrm{I}}, \text{Session}_{\mathsf{R}}, \text{Der}_{\mathrm{I}}, \text{Reveal}, \text{Rev-State}, \text{Corrupt}, \text{Test}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, ..., \mathsf{H}_N, \widetilde{\mathsf{H}}_1, ..., \widetilde{\mathsf{H}}_S\}$, where Der_{I}, Reveal, Rev-State, Test, F, G, H, H_n for $n \in [N]$ and $\widetilde{\mathsf{H}}_{\text{sID}}$ for $\text{sID} \in [S]$ are defined as in games $G_{0,b}$ in Figure 23 resp. 24. Lines written in blue color highlight how the adversary simulates $G_{0,b}$ and how event $\mathsf{BREAK}_{\mathsf{SIG}}$ leads to a forgery.

to compute ciphertext c_r in line 52 and draws a random key K_r in line 50. K_r is then retrieved in line 35 (Fig. 24) when the same c_r is issued to $\text{Session}_{\mathsf{R}}$. Furthermore, the $\mathsf{SimHash}_{\mathsf{CCA}}$ algorithm is used in all random oracles H_n, where $n \in [N]$. In case party n is corrupted, i. e., sk_n is known to the adversary $\mathcal{A}$, we call $\mathsf{SimHash}_{\mathsf{CCA}}$ with set $\mathcal{CK}_n$, otherwise with set $\mathcal{C}_n$.

For $b \in \{0,1\}$, we construct adversaries $\mathcal{C}_b$ against N-NCKE-CCA security of $\mathsf{KEM}_{\mathsf{CCA}}$ in Figure 26, similar to adversaries $\mathcal{B}_b$ in Figure 16, only that here we have only ciphertexts generated by initiating sessions and that signatures are simulated as well. If $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CCA}_{\mathsf{real}}$ game, it perfectly simulates $G_{1,b}$. Otherwise, if $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CCA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{2,b}$. We have

Figure 26: Adversaries $\mathcal{C}_b$ against N-NCKE-CCA for the proof of Eqn. (10). $\mathcal{A}$ has access to oracles $\text{O} := \{\text{SESSION}_{\text{I}}, \text{SESSION}_{\text{R}}, \text{DER}_{\text{I}}, \text{REVEAL}, \text{REV-STATE}, \text{CORRUPT}, \text{TEST}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, \ldots, \mathsf{H}_N, \widetilde{\mathsf{H}}_1, \ldots, \widetilde{\mathsf{H}}_S\}$, where DER_{I}, REVEAL, REV-STATE, TEST, F, G, H and $\widetilde{\mathsf{H}}_{\text{sID}}$ for sID $\in [S]$ are defined as in games $G_{1,b}$ in Figure 23 resp. 24. Lines written in blue color highlight how the adversary simulates $G_{1,b}$ and interpolates to $G_{2,b}$.

$$\begin{aligned} \left|\Pr[G^{\mathcal{A}}_{2,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{1,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CCA}^{\mathcal{C}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CCA}^{\mathcal{C}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \text{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{C}_b) \end{aligned} \tag{10}$$

We make the same preparations as in the proof of Theorem 2, before switching to the algorithms of the simulator $\mathsf{Sim}_{\mathsf{CPA}} = (\mathsf{SimGen}_{\mathsf{CPA}}, \mathsf{SimEncaps}_{\mathsf{CPA}}, \mathsf{SimHash}_{\mathsf{CPA}})$ in games $G_{4,b}$. In particular, we introduce an intermediate game which delays the computation and encryption of the state.

GAMES $G_{3,b}$. We move the encryption of the state to the REV-STATE oracle and choose only IV when $\text{SESSION}_{\text{I}}$ is called. Then, when REV-STATE is queried and the initiator i

is corrupted, we honestly compute the state in line 07 (Fig. 23). If the adversary already made a query to G, where (k_i, IV) are as in the corresponding session, we raise flag BAD in line 09 and abort in line 10. Otherwise, we choose a random string ψ in line 12 and patch the random oracle G for the case that $\mathcal{A}$ issues a query with the correct symmetric key k_i and IV. If BAD is not raised, the adversary's view does not change. We have

$$\left|\Pr[G_{3,b} \Rightarrow 1] - \Pr[G_{2,b} \Rightarrow 1]\right| \leq \Pr[\mathrm{BAD}] \leq S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} \ .$$

Games $G_{4,b}$. Following the previous proofs, we now use the $\mathsf{SimGen}_{\mathsf{CPA}}$ algorithm to generate ephemeral key pairs $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$, the $\mathsf{SimEncaps}_{\mathsf{CPA}}$ algorithm to compute ciphertext $\tilde{c}$ and choose a uniformly random key $\widetilde{K}$ whenever the ephemeral public key $\widetilde{\mathsf{pk}}$ was output by $\textsc{Session}_{\mathrm{I}}$. Random oracles $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ will then use the $\mathsf{SimHash}_{\mathsf{CPA}}$ algorithm.

In Figure 27, we construct adversaries $\mathcal{D}_b$ for $b \in \{0,1\}$ against S-NCKE-CPA which are similar to adversaries $\mathcal{C}_b$ defined in Figure 17. Here, $\mathcal{D}_b$ generates N key pairs with $\mathsf{Gen}_{\mathsf{SIG}}$ and simulates the signatures.

If $\mathcal{D}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{real}}$ game, it perfectly simulates $G_{3,b}$. Otherwise, if $\mathcal{D}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{4,b}$. We have

$$\begin{aligned} \left|\Pr[G^{\mathcal{A}}_{4,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{D}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{D}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{D}_b) \ . \end{aligned} \tag{11}$$

Games $G_{5,b}$. In games $G_{5,b}$, we change the output for K_0^* in the $\textsc{Test}$ oracle to a random key in line 53 (Fig. 23). Now games $G_{5,0}$ and $G_{5,1}$ are equal as well as games $G_{5,1}$ and $G_{4,1}$, hence

$$\begin{aligned} \left|\Pr[G^{\mathcal{A}}_{4,1} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| &= \left|\Pr[G^{\mathcal{A}}_{5,1} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| \\ &= \left|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| \ . \end{aligned} \tag{12}$$

It remains to bound $|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]|$. Therefore, we will now consider the different attacks for $\mathsf{IND\text{-}FS\text{-}St}$ as described in Table 2. Depending on which queries the adversary makes, each test session must belong to at least one of the attacks or the game will return 0 anyway.

Again, we will assume that the adversary queries as much information as possible. Variables of a particular test session sID^* are denoted by $\mathrm{context}^* = (\mathsf{vk}_{i^*}, \mathsf{pk}_{i^*}, \mathsf{vk}_{r^*}, \mathsf{pk}_{r^*}, \widetilde{\mathsf{pk}}^*, c_{r^*}, \tilde{c}^*, \sigma^*, \pi^*)$ and $IV^*, k_{i^*}, \widetilde{K}^*$. As we assumed in the beginning that ciphertexts and long-term as well as ephemeral key pairs are all different, it is not possible to recreate a particular session. In particular, this means that there is no partially matching session and row (0) will return false.

Now the only possibility to learn any test key K_0^* is through random oracle queries. Let QUERY be the event that $(\mathrm{context}, \widetilde{K})$ of any test session is queried to H and QUERY^* be the event that $(\mathrm{context}^*, \widetilde{K}^*)$ of a specific test session is queried to H. We have

$$\left|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| \leq \Pr[\mathrm{QUERY}] \ .$$

Figure 27: Adversaries $\mathcal{D}_b$ against S-NCKE-CPA for the proof of Eqn. (11). $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, \ldots, \mathsf{H}_N, \widetilde{\mathsf{H}}_1, \ldots, \widetilde{\mathsf{H}}_S\}$, where $\textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{F}, \mathsf{H}$ and H_n for $n \in [N]$ are defined as in games $G_{3,b}$ in Figure 23 resp. 24. Oracles $\textsc{Rev-State}$, G, $\widetilde{\mathsf{H}}_{\text{sID}}$ for sID $\in [S]$ are defined in Figure 28. Lines written in blue color highlight how the adversary simulates $G_{3,b}$ and interpolates to $G_{4,b}$.

Union bound over the maximum number of test sessions T yields

$$\Pr[\text{QUERY}] \leq T \cdot \Pr[\text{QUERY}^*] \ .$$

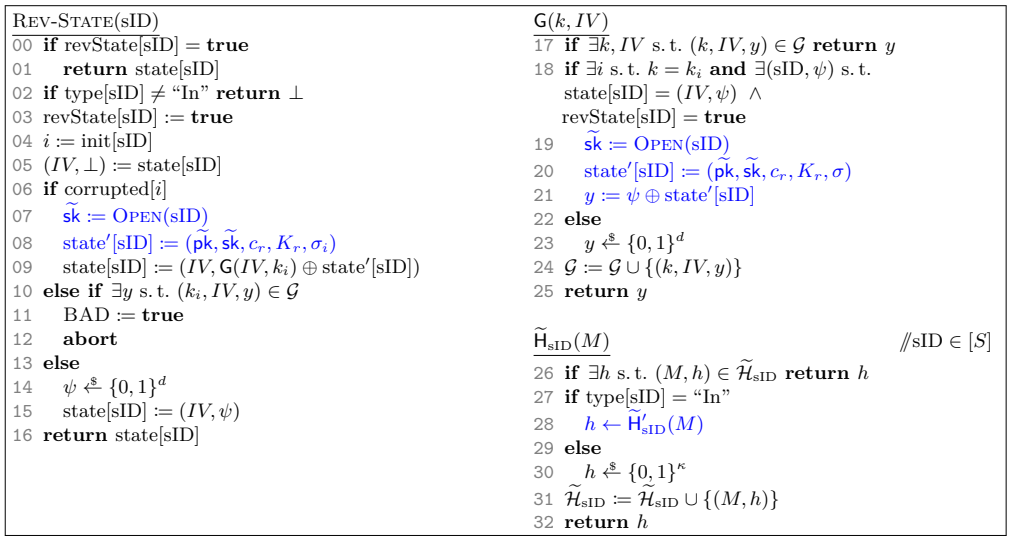

Figure 28: Oracles REV-STATE, G, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for sID $\in [S]$ for adversaries $\mathcal{D}_b$ in Fig. 27.

We will now focus on the event QUERY* and iterate over the attacks in Table 2. An overview is given in Figure 29.

ATTACK $(1 \vee 2)$, (10). If $(1 \vee 2) \Rightarrow$ **true**, the test session has a matching session and both long-term secret keys $(\mathsf{sigk}_{i^*}, \mathsf{sk}_{i^*}, k_{i^*})$ and $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$ are revealed. However, $\mathcal{A}$ is not allowed to query the test session's state or the state of the matching session, depending on the type of the test session. Thus, $\mathcal{A}$ has no information about $\widetilde{\mathsf{sk}}^*$. As there is a matching session for this test session, $\widetilde{\mathsf{pk}}^*$ was generated by SESSION$_{\mathrm{I}}$, which means that $\widetilde{K}^*$ is chosen uniformly at random and thus independent of $\widetilde{\mathsf{pk}}^*$ and $\tilde{c}^*$. Hence, the probability that $\mathcal{A}$ queries H on (context*, $\widetilde{K}^*$) is $q_{\mathsf{H}}/|\mathcal{K}|$.

If $(10) \Rightarrow$ **true**, the test session has a partially matching session and it is of type "Re". $\mathcal{A}$ is allowed to obtain both long-term secret keys as the test session is completed. $\mathcal{A}$ is not allowed to query the state of the partially matching session. This yields the same scenario as described above. It follows that

$$\Pr[\text{QUERY}^* \mid (1 \vee 2) \Rightarrow \textbf{true}] = \Pr[\text{QUERY}^* \mid (10) \Rightarrow \textbf{true}] \leq \frac{q_{\mathsf{H}}}{|\mathcal{K}|} \ .$$

ATTACK $(7 \vee 8)$, (16). If $(7 \vee 8) \Rightarrow$ **true**, the test session has a matching session and the state (IV^*, ψ^*) is revealed. Furthermore, $\mathcal{A}$ can obtain $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$. As the initiator is not corrupted, $\widetilde{\mathsf{sk}}^*$ is unknown to $\mathcal{A}$, unless it issues a query to G on a correct value k_{i^*}. The probability that this happens is upper bounded by $q_{\mathsf{G}}/2^{\kappa}$. Another way to learn the session key is to query H directly, where $\mathcal{A}$ succeeds with probability $q_{\mathsf{H}}/|\mathcal{K}|$, as $\widetilde{K}^*$ is chosen uniformly at random.

If $(16) \Rightarrow$ **true**, the test session has a partially matching session and it is of type "Re". $\mathcal{A}$ can reveal the state (IV, ψ) of the partially matching session, the rest remains

Attack	Security relies on ...
$(1 \vee 2)$, (10)	IV^* unknown $\Rightarrow \widetilde{\mathsf{sk}}^*$ unknown $\Rightarrow \widetilde{K}^*$ unknown
$(7 \vee 8)$, (16)	k_{i^*} unknown $\Rightarrow \widetilde{\mathsf{sk}}^*$ unknown $\Rightarrow \widetilde{K}^*$ unknown
(17), (23)	sk_{r^*} unknown before session completed $\Rightarrow \pi^*$ cannot be computed
(18)	sigk_{i^*} unknown before session completed $\Rightarrow \sigma^*$ cannot be forged

Figure 29: Overview of attacks for the proof of Theorem 3.

unchanged to before. Thus,

$$\Pr[\text{QUERY}^* \mid (7 \vee 8) \Rightarrow \textbf{true}] = \Pr[\text{QUERY}^* \mid (16) \Rightarrow \textbf{true}] \leq \frac{q_{\mathsf{G}}}{2^\kappa} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|} \ .$$

Attack (17), (23). If (17) $\Rightarrow$ **true**, the test session has no matching session and it is of type "In". $\mathcal{A}$ is allowed to obtain the initiator's long-term secret key $(\mathsf{sigk}_{i^*}, \mathsf{sk}_{i^*}, k_{i^*})$ and can choose $(\tilde{c}^*, \widetilde{K}^*)$ itself, but the responder's long-term secret key $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$ will only be available after the session key is established. Thus, $\mathcal{A}$ does not know K_{r^*}, which is a uniformly random key. $\mathcal{A}$ can only complete the session if it manages to compute π by querying F on K_{r^*}, where the probability for that is upper bounded by $q_{\mathsf{F}}/|\mathcal{K}|$.

If (23) $\Rightarrow$ **true**, instead of obtaining the initiator's long-term secret key, $\mathcal{A}$ is allowed to obtain the initiator's state (IV^*, ψ^*). The rest remains the same. $\mathcal{A}$ can only complete the session if it manages to forge π, for which it has to query F on the correct key K_{r^*}. Thus,

$$\Pr[\text{QUERY}^* \mid (17) \Rightarrow \textbf{true}] = \Pr[\text{QUERY}^* \mid (23) \Rightarrow \textbf{true}] \leq \frac{q_{\mathsf{F}}}{|\mathcal{K}|} \ .$$

Attack (18). If (18) $\Rightarrow$ **true**, the test session has no matching session and it is of type "Re", which means that $\mathcal{A}$ can reveal the responder's long-term secret key $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$. Here, $\mathcal{A}$ has two possibilities. First, it can choose $(\widetilde{\mathsf{pk}}^*, \widetilde{\mathsf{sk}}^*)$ and (c_{r^*}, K_{r^*}) itself. However, the initiator's long-term secret key $(\mathsf{sigk}_{i^*}, \mathsf{sk}_{i^*}, k_{i^*})$ will only be available after the session key is established and $\mathcal{A}$ has to forge σ_{i^*} to call $\textsc{Session}_{\mathsf{R}}$ in the first place. As the game aborts if that happens, the session key will never be computed.

$$\Pr[\text{QUERY}^* \mid (18) \Rightarrow \textbf{true}] = 0 \ .$$

Taking the maximum over the conditional probabilities and assuming that $q_{\mathsf{H}} \approx q_{\mathsf{F}}$, it follows that

$$\begin{aligned} \left|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| &\leq \Pr[\text{QUERY}] \leq T \cdot \Pr[\text{QUERY}^*] \\ &\leq T \cdot \left(\frac{q_{\mathsf{G}}}{2^\kappa} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right) \end{aligned} \tag{13}$$

The proof of Theorem 3 follows by collecting the probabilities from Equations (8)-(13) and by folding adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$, $\mathcal{C}_0$ and $\mathcal{C}_1$ as well as $\mathcal{D}_0$ and $\mathcal{D}_1$ into single adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. □

7 Concrete Instantiation of AKE Protocols

7.1 NCKE from the DDH Assumption

Let us first describe the hash proof system we will use. Therefore, let GGen be a group generation algorithm which takes the security parameter 1^κ as input and returns $(\mathbb{G}, p, g_1)$, where g_1 is a generator of the cyclic group $\mathbb{G}$ with prime order p. Define $group = (\mathbb{G}, p, g_1, g_2)$, where $g_2 = g_1^w$ for $w \xleftarrow{\$} \mathbb{Z}_p$. Define $\mathcal{Y} = \mathbb{Z}_p^2$ and $\mathcal{X} = \{(g_1^r, g_2^r) : r \in \mathbb{Z}_p\}$. A value r is a witness that $(c_1, c_2) \in \mathcal{X}$. Define $\mathcal{SK} = \mathbb{Z}_p^2$, $\mathcal{PK} = \mathbb{Z}_p$ and $\mathcal{Z} = \mathbb{Z}_p$. For $\mathsf{sk} = (x_1, x_2) \in \mathbb{Z}_p^2$, define $\mu(\mathsf{sk}) = X = g_1^{x_1} g_2^{x_2}$. This defines the output of the parameter generation algorithm Par.

For $(c_1, c_2) \in \mathcal{Y}$ define $\Lambda_{\mathsf{sk}}(c_1, c_2) := Z = (c_1^{x_1} c_2^{x_2})$. This defines the private evaluation algorithm $\mathsf{Priv}(\mathsf{sk}, (c_1, c_2))$. Given $\mathsf{pk} = \mu(\mathsf{sk}) = X$, $(c_1, c_2) \in \mathcal{X}$ and a witness $r \in \mathbb{Z}_p$ such that $(c_1, c_2) = (g_1^r, g_2^r)$, the public evaluation algorithm $\mathsf{Pub}(\mathsf{pk}, (c_1, c_2), r)$ computes $Z = \Lambda_{\mathsf{sk}}(c_1, c_2)$ as $Z = X^r$.

We define $\mathsf{KEM}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Encaps}_{\mathsf{DDH}}, \mathsf{Decaps}_{\mathsf{DDH}})$ with global parameters $\mathsf{par} := (\mathbb{G}, p, g_1, g_2)$ as shown in Figure 30.

$\underline{\mathsf{Gen}_{\mathsf{DDH}}(\mathsf{par})}$	$\underline{\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{DDH}}(\mathsf{pk}, m)}$	$\underline{\mathsf{Decaps}^{\mathsf{H}}_{\mathsf{DDH}}(\mathsf{sk}, (c_1, c_2))}$
00 $(x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2$	03 $r \xleftarrow{\$} \mathbb{Z}_p$	07 $K := \mathsf{H}(c_1, c_2, c_1^{x_1} c_2^{x_2})$
01 $X := g_1^{x_1} g_2^{x_2}$	04 $(c_1, c_2) := (g_1^r, g_2^r)$	08 **return** K
02 **return** $(\mathsf{pk} := X,$	05 $K := \mathsf{H}(c_1, c_2, X^r)$	
$\mathsf{sk} := (x_1, x_2))$	06 **return** $((c_1, c_2), K)$	

Figure 30: Key encapsulation mechanism $\mathsf{KEM}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Encaps}_{\mathsf{DDH}}, \mathsf{Decaps}_{\mathsf{DDH}})$.

Definition 3 (m-fold DDH Problem)**.** *Let* GGen *be a PPT algorithm that on input* 1^κ *outputs a cyclic group* $\mathbb{G}$ *of prime order* $2^{k-1} \leq p \leq 2^k$ *with generator* g_1*. Furthermore let* $g_2 = g_1^\omega$ *for* $\omega \xleftarrow{\$} \mathbb{Z}_p$*. The* m-DDH *problem requires to distinguish* m DDH *tuples from* m *uniformly random tuples:*

$$\mathrm{Adv}^{m\text{-}\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{A}) := \Big|\Pr[\mathcal{A}(\mathbb{G}, p, g_1, g_2, (g_1^{r_i}, g_2^{r_i})_{i\in[m]}) \Rightarrow 1] - \Pr[\mathcal{A}(\mathbb{G}, p, g_1, g_2, (g_1^{r_i}, g_2^{r'_i})_{i\in[m]}) \Rightarrow 1]\Big| \ ,$$

where probability is taken over $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}$*,* $r_i, r'_i \xleftarrow{\$} \mathbb{Z}_p$ *for* $i \in [m]$*, as well as the coin tosses of* $\mathcal{A}$*.*

Lemma 1 (Random self-reducibility of DDH [EHK+13])**.** *For any adversary* $\mathcal{C}$ *against the* m*-fold* DDH *problem, there exists an adversary* $\mathcal{B}$ *against the* DDH *problem with roughly the same running time such that*

$$\mathrm{Adv}^{m\text{-}\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{C}) \leq \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + \frac{1}{p-1} \ .$$

The following theorem establishes that the construction given in Figure 30 is an N-receiver non-committing encapsulation mechanism under the DDH assumption.

Theorem 4. *Under the* DDH *assumption and in the random oracle model,* $\mathsf{KEM}_{\mathsf{DDH}}$ *is an* N*-receiver non-committing key encapsulation mechanism. In particular, for any* N-NCKE-CCA *adversary* $\mathcal{A}$ *against* $\mathsf{KEM}_{\mathsf{DDH}}$ *and* $\mathsf{Sim}_{\mathsf{DDH}}$ *that issues at most* q_E *queries per user to* ENCAPS, q_D *queries to* DECAPS *and at most* q_{H} *queries to each random oracle* H_n, $n \in [N]$, *there exists an adversary* $\mathcal{B}$ *against* DDH *with roughly the same running time such that*

$$\mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{DDH}}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + \frac{N \cdot q_E \cdot (q_{\mathsf{H}} + q_D + 1)}{p} + \frac{1}{p-1} \ ,$$

where $\mathsf{Sim}_{\mathsf{DDH}}$ *is the simulator defined relative to* $\mathsf{KEM}_{\mathsf{DDH}}$.

Proof. We apply Theorem 1 and analyze the entropy of the underlying HPS. The key space $\mathcal{Z}$ is $\mathbb{Z}_p$. For $\mathsf{sk} = (x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathsf{pk} = \mu(\mathsf{sk}) = g_1^{x_1} g_2^{x_2}$ and $Z = \mathsf{Priv}(\mathsf{sk}, (c_1, c_2)) = c_1^{x_1} c_2^{x_2}$, where $(c_1, c_2) = (g_1^r, g_2^{r'})$ and $(r, r') \xleftarrow{\$} \mathbb{Z}_p^2$, we have

$$\begin{pmatrix} \log_{g_1} \mathsf{pk} \\ \log_{g_1} Z \end{pmatrix} = M \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \text{ where } M = \begin{pmatrix} 1 & w \\ r & wr' \end{pmatrix} \ .$$

If $r \neq r'$, then $\det M = w(r' - r) \neq 0$, which implies that pk and Z are random and independent group elements as long as x_1, x_2 are unknown. Thus, for all $Z' \in \mathcal{Z}$, holds that $\Pr[Z = Z'] = 1/p$. In Definition 3, all values r_i and r'_i are drawn uniformly at random from $\mathbb{Z}_p$. The probability that $r_i = r'_i$ for any $i \in [N \cdot q_E]$ is upper bounded by $N \cdot q_E/p$. Furthermore, the probability that a specific challenge ciphertext is issued to DECAPS before it is output by ENCAPS is at most q_D/p. It follows that

$$\mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{A}) \leq \mathrm{Adv}^{m\text{-DDH}}_{\mathsf{GGen}}(\mathcal{B}) + \frac{N \cdot q_E}{p} + \frac{N \cdot q_E \cdot q_{\mathsf{H}}}{p} + \frac{N \cdot q_E \cdot q_D}{p} \ .$$

Now Theorem 4 follows directly from Lemma 1. □

7.2 Concrete Instantiation of AKE Protocols

We instantiate protocols $\mathsf{AKE}_{\mathsf{wFS}}$ (Section 5) and $\mathsf{AKE}_{\mathsf{FS}}$ (Section 6.2) with $\mathsf{KEM}_{\mathsf{DDH}}$ (Section 7.1) for both $\mathsf{KEM}_{\mathsf{CPA}}$ and $\mathsf{KEM}_{\mathsf{CCA}}$. We will not give a concrete instantiation of the signature scheme used in $\mathsf{AKE}_{\mathsf{FS}}$ at this point. The resulting protocols $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ and $\mathsf{AKE}_{\mathsf{FS,DDH}}$ are shown in Figure 1 in the introduction.

Note that for $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ we can improve efficiency by sending only one ciphertext for both $\widetilde{\mathsf{pk}}$ and pk_i in the second message, as $\mathsf{KEM}_{\mathsf{DDH}}$ is a multi-recipient KEM. We establish Theorem 5 and give a proof sketch.

Theorem 5 (IND-wFS-St security of $\mathsf{AKE}_{\mathsf{wFS,DDH}}$)**.** *Under the* DDH *assumption,* $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ *is* IND-wFS-St *secure in the random oracle model. In particular, for any* IND-wFS-St *adversary* $\mathcal{A}$ *against* $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ *with* N *parties that establishes at most* S *sessions and issues at most* T *queries to the test oracle* TEST, q_{G} *queries to random oracle* G, $q_{\widetilde{\mathsf{H}}}$, q_{H_n} *queries to each random oracle* $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ *and* H_n *and at most* q_{H} *queries*

to random oracle H*, there exists an adversary* $\mathcal{B}$ *against* DDH *with roughly the same running time such that*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}_{\mathsf{wFS},\mathsf{DDH}}}(\mathcal{A}) \quad \leq \quad & 2 \cdot \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + T \cdot \frac{q_{\mathsf{G}} + q_{\mathsf{H}}}{2^{\kappa}} + (N+S)^2 \cdot \frac{1}{p} + N^2 \cdot \frac{1}{2^{\kappa}} \\ & + S^2 \cdot \left(\frac{2}{p} + \frac{1}{2^{\kappa}}\right) + 2S \cdot \left(\frac{q_{\mathsf{G}}}{2^{2\kappa}} + \frac{q_{\widetilde{\mathsf{H}}} + q_{\mathsf{H}_n} + 1}{p}\right) + \frac{2}{p-1} \ ,\end{aligned}$$

where κ *is a security parameter.*

Due to the improved construction, we cannot apply Theorem 2 directly, but we give a proof sketch from the DDH assumption and show that the same technique as in the proofs of Theorems 2 and 4 can be used.

Proof. We proceed similar and consider collisions first. We assume that all key pairs generated by $\mathsf{Gen}_{\mathsf{DDH}}$ are different. Note that we also have to consider collisions between long-term and ephemeral public keys. It holds that

$$\Pr[x_1, x_2, x_1', x_2' \xleftarrow{\$} \mathbb{Z}_p : g_1^{x_1} g_2^{x_2} = g_1^{x_1'} g_2^{x_2'}] = 1/p \ .$$

Union bound yields $(N+S)^2/p$, as we have N long-term public keys and at most S ephemeral public keys. For ciphertexts $(c_1, c_2) \in \mathcal{C}$ output by the encapsulation algorithm $\mathsf{Encaps}_{\mathsf{DDH}}$, it holds that

$$\Pr[r \xleftarrow{\$} \mathbb{Z}_p : (c_1, c_2) = (g_1^r, g_2^r)] = 1/p \ ,$$

which yields an upper bound for collisions of S^2/p, as there are at most S sessions with one ciphertext. We also assume that values IV are different in all sessions and keys k_n are different for all parties.

We use the secret keys to compute keys K_i, K_r and $\widetilde{K}$. Next, we replace all ciphertexts by uniformly random group elements at the same time, reducing to the S-fold DDH assumption and use the random self-reducibility property. In addition to that, we ensure that all ciphertexts are indeed invalid by adding S/p which is the probability that exponents are the same for any ciphertext.

Instead of the corresponding random oracles, we use internal hash functions $\widetilde{\mathsf{H}}'_{\mathrm{sID}}$ and H'_n for $\mathrm{sID} \in [S]$ and $n \in [N]$ to compute keys K_i, K_r and $\widetilde{K}$, but patch the random oracles if the secret key is known to the adversary. As there are at most S challenge keys computed with a long-term key pair and at most S challenge keys computed with an ephemeral key pair, the difference can be upper bounded by $S \cdot q_{\mathsf{H}_n}/p + S \cdot q_{\widetilde{\mathsf{H}}}/p$ using a hybrid argument. Now we can replace K_i, K_r and $\widetilde{K}$ by uniformly random keys.

The rest of the proof is equal to the proof of Theorem 2. The size of the key space of $\mathsf{KEM}_{\mathsf{DDH}}$ is 2^{κ} and the bound follows by collecting all probabilities. □ For protocol

$\mathsf{AKE}_{\mathsf{FS},\mathsf{DDH}}$, we apply Theorem 3 to show $\mathsf{IND\text{-}FS\text{-}St}$ security. The collision probabilities for $\mathsf{KEM}_{\mathsf{DDH}}$ are already shown in the previous proof. Additionally, we need a strongly unforgeable signature scheme.

Theorem 6 (IND-FS-St security of $\mathsf{AKE}_{\mathsf{FS,DDH}}$)**.** *For an N-SUF-CMA secure signature scheme* SIG *and under the* DDH *assumption,* $\mathsf{AKE}_{\mathsf{FS,DDH}}$ *is* IND-FS-St *secure in the random oracle model. In particular, for any* IND-FS-St *adversary $\mathcal{A}$ against* $\mathsf{AKE}_{\mathsf{FS,DDH}}$ *with N parties that establishes at most S sessions and issues at most T queries to the test oracle* TEST*, q_{G} queries to random oracle* G*, q_{F} queries to random oracle* F*, $q_{\widetilde{\mathsf{H}}}$, q_{H_n} queries to each random oracle $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ and H_n and at most q_{H} queries to random oracle* H*, there exists an adversary $\mathcal{B}$ against* DDH *and an adversary $\mathcal{C}$ against N-SUF-CMA such that*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}FS\text{-}St}}_{\mathsf{AKE}_{\mathsf{FS,DDH}}}(\mathcal{A}) \;\leq\; & 4\cdot \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + 2\cdot \mathrm{Adv}^{N\text{-}\mathsf{SUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{C}) + T\cdot\frac{q_{\mathsf{F}}+q_{\mathsf{G}}+q_{\mathsf{H}}}{2^{\kappa}} \\ & + N^2\cdot\left(\frac{1}{2^{\mu_{\mathsf{SIG}}}}+\frac{1}{p}+\frac{1}{2^{\kappa}}\right) + S^2\cdot\left(\frac{2q_{\widetilde{\mathsf{H}}}+6}{p}+\frac{1}{2^{\kappa}}\right) \\ & + 2NS\cdot\frac{q_{\mathsf{H}_n}+2}{p} + 2S\cdot\frac{q_{\mathsf{G}}}{2^{2\kappa}} + \frac{4}{p-1}\;,\end{aligned}$$

where μ_{SIG} is the collision probability of the key generation algorithm $\mathsf{Gen}_{\mathsf{SIG}}$ and κ is a security parameter.

The signature scheme can be instantiated with the tight scheme based on the DDH and CDH assumption proposed by Gjøsteen and Jager in [GJ18], which is also used in their authenticated key exchange protocol.

Acknowledgments

Tibor Jager was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823. Eike Kiltz was supported by the BMBF iBlockchain project, the EU H2020 PROMETHEUS project 780701, DFG SPP 1736 Big Data, and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy - EXC 2092 CASA – 390781972. Doreen Riepel was funded by the DFG under Germany´s Excellence Strategy - EXC 2092 CASA – 390781972. Sven Schäge was supported by the German Federal Ministry of Education and Research (BMBF), Project DigiSeal (16KIS0695) and Huawei Technologies Düsseldorf, Project vHSM.

References

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000. 116, 117

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011*, pages 51–62. ACM Press, October 2011. 117

[BG11] Colin Boyd and Juan Manuel González Nieto. On forward secrecy in one-round key exchange. In Liqun Chen, editor, *13th IMA International Conference on Cryptography and Coding*, volume 7089 of *LNCS*, pages 451–468. Springer, Heidelberg, December 2011. 117

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658. Springer, Heidelberg, March 2015. 116, 117, 118, 119, 120, 133, 134, 135, 146

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304. Springer, Heidelberg, May 2016. 117

[BJS15] Florian Bergsma, Tibor Jager, and Jörg Schwenk. One-round key exchange with strong security: An efficient and generic construction in the standard model. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 477–494. Springer, Heidelberg, March / April 2015. 120

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994. 119, 133

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797. Springer, Heidelberg, August 2019. 116, 117, 119, 121, 134, 135

[CF12] Cas J. F. Cremers and Michele Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751. Springer, Heidelberg, September 2012. 117, 133, 134

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996. 119

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001. 117, 133, 134

[Cre09] Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 20–33. Springer, Heidelberg, June 2009. 133

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu, and Duncan S. Wong, editors, *ASIACCS 11*, pages 80–91. ACM Press, March 2011. 133

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002. 119, 124

[DG20] Hannah Davis and Felix Günther. Tighter proofs for the sigma and tls 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029, 2020. https://eprint.iacr.org/2020/1029. 117, 119

[DG21] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21, Part II*, volume 12727 of *LNCS*, pages 448–479. Springer, Heidelberg, June 2021. 121

[DJ20] Denis Diemert and Tibor Jager. On the tight security of tls 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Cryptology ePrint Archive, Report 2020/726, 2020. https://eprint.iacr.org/2020/726. 117, 119

[DJ21] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments. *Journal of Cryptology*, 34(3):30, July 2021. 121

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013. 157

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, February / March 2013. 116, 117

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 467–484. Springer, Heidelberg, May 2012. 120

[GHKW16] Romain Gay, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Tightly CCA-secure encryption without pairings. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 1–27. Springer, Heidelberg, May 2016. 116, 117

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 95–125. Springer, Heidelberg, August 2018. 117, 118, 119, 120, 121, 134, 135, 160

[HK09] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 637–653. Springer, Heidelberg, August 2009. 164, 165, 167

[HKSU20] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 389–422. Springer, Heidelberg, May 2020. 128, 133, 134

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Heidelberg, August 2012. 117, 133, 134

[KP05] Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 549–565. Springer, Heidelberg, December 2005. 117

[KPSY09] Eike Kiltz, Krzysztof Pietrzak, Martijn Stam, and Moti Yung. A new randomness extraction paradigm for hybrid encryption. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 590–609. Springer, Heidelberg, April 2009. 124

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005. 117, 119, 121, 129, 133, 134

[LLGW20] Xiangyu Liu, Shengli Liu, Dawu Gu, and Jian Weng. Two-pass authenticated key exchange with explicit authentication and tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 785–814. Springer, Heidelberg, December 2020. 117, 119, 121, 122

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007. 117, 120, 129, 133, 134

[LM06] Kristin Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 378–394. Springer, Heidelberg, April 2006. 117

[LS17] Yong Li and Sven Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1343–1360. ACM Press, October / November 2017. 132, 134

[LSY+14] Yong Li, Sven Schäge, Zheng Yang, Christoph Bader, and Jörg Schwenk. New modular compilers for authenticated key exchange. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS 14*, volume 8479 of *LNCS*, pages 1–18. Springer, Heidelberg, June 2014. 120

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002. 119, 123, 124

[Sch15] Sven Schäge. TOPAS: 2-pass key exchange with full perfect forward secrecy and optimal communication complexity. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1224–1235. ACM Press, October 2015. 117

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://eprint.iacr.org/2004/332. 122, 148

[Yon12] Kazuki Yoneyama. One-round authenticated key exchange with strong forward secrecy in the standard model against constrained adversary. In Goichiro Hanaoka and Toshihiro Yamauchi, editors, *IWSEC 2012, Fukuoka, Japan, November 7-9, 2012. Proceedings*, volume 7631 of *LNCS*, pages 69–86. Springer, 2012. 120

A NCKE from the Higher Residuosity Assumption

We show how to construct an NCKE scheme using the hash proof system based on the higher residuosity (HR) problem described in [HK09]. Therefore, we will first recall some definitions.

Quadratic Residues. An n-bit integer $N = PQ$, where P, Q are two distinct $n/2$-bit odd primes is called an RSA modulus. We assume that N is a Blum integer, i. e., both P and Q are congruent 3 modulo 4. By $\phi(N)$ we denote Euler's totient function, i. e. $\phi(N) = (P-1)(Q-1)$. By $\mathbb{J}_N$ we denote the subgroup of all elements from $\mathbb{Z}_N^*$ with Jacobi symbol 1 and by $\mathbb{QR}_N$ the group of quadratic residues modulo N, which is a subgroup of $\mathbb{J}_N$ with order $(P-1)(Q-1)/4$.

Signed Quadratic Residues. For $x \in \mathbb{Z}_N$, let $|x|$ denote the absolute value of x, where x is represented as a signed integer in the set $\{-(N-1)/2, ..., (N-1)/2\}$. The signed group $\mathbb{G}^+$, where $\mathbb{G}$ is a subgroup of $\mathbb{Z}_N^*$, is defined as $\mathbb{G}^+ := \{|x| : x \in \mathbb{G}\}$. For

$g, h \in \mathbb{G}^+$ and integer x, we define $g \circ h := |g \cdot h \bmod N|$ and $g^{\underline{x}} := |g^x \bmod N|$. Our focus will be on the group of signed quadratic residues $\mathbb{QR}_N^+$.

Lemma 2. [HK09] *Let N be a Blum integer. Then:*
(i) $(\mathbb{QR}_N^+, \circ)$ *is a group of order* $\phi(N)/4$.
(ii) $\mathbb{QR}_N^+ = \mathbb{J}_N^+$. *In particular,* $\mathbb{QR}_N^+$ *is efficiently recognizable (given only* N*).*
(iii) If $\mathbb{QR}_N$ *is cyclic, so is* $\mathbb{QR}_N^+$.

RSA Instance Generator. Let $0 \leq \delta \leq 1/4$ be a constant and $n(\kappa)$ be a function. Let RSAgen be an algorithm that generates elements (N, P, Q, S) such that $N = PQ$ is an n-bit Blum integer. The prime factors of $\phi(N)/4$ are pairwise distinct and at least δn-bit integers. Furthermore, $S > 1$ is a divisor of $\phi(N)/4$ with $1 < \gcd(S, (P-1)/2) < (P-1)/2$ and $1 < \gcd(S, (Q-1)/2) < (Q-1)/2$.

Statistical Distance. The statistical distance between two random variables X and Y having a common domain $\mathcal{X}$ is defined as $\Delta[X, Y] = \frac{1}{2} \sum_{x \in \mathcal{X}} |\Pr[X = x] - \Pr[Y = x]|$. The min-entropy of a random variable X is defined as $H_\infty(X) = -\log(\max_{x \in \mathcal{X}} \Pr[X = x])$.

Now we describe our HPS. Define $group = (N, g)$, where $(N, P, Q, S) \leftarrow \mathsf{RSAgen}(1^\kappa)$ and g is a uniform generator of $\mathbb{G}_S^+$. Define $\mathcal{Y} = \mathbb{QR}_N^+$ and $\mathcal{X} = \mathbb{G}_S^+ = \{g^{\underline{r}} : r \in \mathbb{Z}_S\}$. A value r is a witness that $c \in \mathcal{X}$. It is possible to sample an almost uniform element from $\mathcal{X}$ together with a witness by first picking $r \in [N/4]$ and defining $c = g^{\underline{r}} \in \mathbb{G}_S^+$. We will determine the statistical distance below in Equation (14). Membership in $\mathcal{Y}$ can be efficiently checked by Lemma 2. Define $\mathcal{SK} = [N/4]$, $\mathcal{PK} = \mathbb{G}_S^+$ and $\mathcal{Z} = \mathbb{QR}_N^+$. For $\mathsf{sk} = x \in [N/4]$, define $\mu(\mathsf{sk}) = X = g^{\underline{x}} \in \mathbb{G}_S^+$. This defines the output of the parameter generation algorithm Par.

For $c \in \mathcal{Y}$ define $\Lambda_{\mathsf{sk}}(c) := Z = c^{\underline{x}}$. This defines the private evaluation algorithm $\mathsf{Priv}(\mathsf{sk}, c)$. Given $\mathsf{pk} = \mu(\mathsf{sk}) = X$, $c \in \mathcal{X}$ and a witness $r \in \mathbb{Z}$ such that $c = g^{\underline{r}}$, the public evaluation algorithm $\mathsf{Pub}(\mathsf{pk}, c, r)$ computes $Z = \Lambda_{\mathsf{sk}}(c)$ as $Z = X^{\underline{r}}$.

We want to analyze the statistical distance between the two distributions defined by sampling from $X_1 = [\phi(N)/4]$ and sampling from $X_2 = [N/4]$. As $\phi(N)/4 = ST$ and $N/4 = ST + (P + Q - 1)/4$, we can write the statistical distance as

$$\Delta[X_1, X_2] = \frac{(P+Q-1)/4}{N/4} = \frac{1}{P} + \frac{1}{Q} - \frac{1}{PQ} \leq \frac{1}{P} + \frac{1}{Q} = O(2^{-n/2}) \ , \tag{14}$$

where the last equation holds because P and Q are both $n/2$-bit primes.

Definition 4 (m-fold Higher Residuosity Problem)**.** *Let (N, P, Q, S) be generated by* RSAgen*. The higher residuosity (HR) problem requires to distinguish m random elements from $\mathbb{G}_S^+$ from m random elements from $\mathbb{QR}_N^+$. The advantage of an adversary $\mathcal{A}$ against the* HR *problem is defined as*

$$\mathrm{Adv}^{m\text{-}\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{A}) := |\Pr[\mathcal{A}(N, g, c_1, ..., c_m) \Rightarrow 1] - \Pr[\mathcal{A}(N, g, c'_1, ..., c'_m) \Rightarrow 1]| \ ,$$

where the probability is taken over $(N, P, Q, S) \leftarrow \mathsf{RSAgen}$, $c_1, ..., c_m \xleftarrow{\$} \mathbb{G}_S^+$ and $c'_1, ..., c'_m \xleftarrow{\$} \mathbb{QR}_N^+$ as well as the coin tosses of $\mathcal{A}$.

Next, we want to show that the m-fold HR problem tightly reduces to the (1-)HR problem using random self-reducibility.

Lemma 3 (Random self-reducibility of HR)**.** *For any adversary $\mathcal{A}$ against the m-fold* HR *problem, there exists an adversary $\mathcal{B}$ against the* HR *problem with roughly the same running time such that*

$$\mathrm{Adv}^{m\text{-}\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{B}) + m \cdot O(2^{-\delta n(\kappa)}) + O(2^{-n/2}) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against the m-HR problem. We construct adversary $\mathcal{B}$ against the (1-)HR problem as shown in Figure 31.

$\mathcal{B}(N, g, c)$
00 **for** $i \in [m]$
01 $\quad a_i \xleftarrow{\$} [N/4]$
02 $\quad c_i := c^{a_i}$
03 $b' \leftarrow \mathcal{A}(N, g, c_1, ..., c_m)$
04 **return** b'

Figure 31: Adversary $\mathcal{B}$ against the HR problem for the proof of Lemma 3.

$\mathcal{B}$ inputs (N, g, c), where g is a generator of $\mathbb{G}_S^+$ and c is either a random element from $\mathbb{G}_S^+$ or from $\mathbb{QR}_N^+$. It samples m random elements a_i from $[N/4]$, computes c_i as c^{a_i} and runs adversary $\mathcal{A}$ on input $(N, g, c_1, ..., c_m)$.

First, note that $|\mathbb{G}_S^+| = S$ and $|\mathbb{QR}_N^+| = ST$, where S and T have only prime factors that are distinct and greater than $2^{-\delta n(\kappa)}$. Thus, c is a generator of $\mathbb{G}_S^+$ or $\mathbb{QR}_N^+$ with high probability $1 - O(2^{-\delta n(\kappa)})$.

Second, if c is a generator of $\mathbb{G}_S^+$ resp. $\mathbb{QR}_N^+$, then $c_i := c^{a_i}$, where $a_i \xleftarrow{\$} [\phi(N)/4]$ and $i \in [m]$, are m random and independent elements in the corresponding group. As $\mathcal{B}$ does not know $\phi(N)$, it samples exponents a_i from $[N/4]$. As shown above, the statistical distance between these $(c_1, ..., c_m)$ and the input of $\mathcal{A}$ in the original m-HR experiment is bounded by $m \cdot O(2^{-\delta n(\kappa)})$.

This yields the bound stated in Lemma 3. □

$\mathsf{Gen}_{\mathsf{HR}}(\mathsf{par})$	$\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{HR}}(\mathsf{pk})$	$\mathsf{Decaps}^{\mathsf{H}}_{\mathsf{HR}}(\mathsf{sk}, c)$
00 $x \xleftarrow{\$} [N/4]$	03 $r \xleftarrow{\$} [N/4]$	07 $K := \mathsf{H}(c, c^x)$
01 $X := g^x$	04 $c := g^r$	08 **return** K
02 **return** $(\mathsf{pk} := X,$	05 $K := \mathsf{H}(c, X^r)$	
$\quad \mathsf{sk} := x)$	06 **return** (c, K)	

Figure 32: Key encapsulation mechanism $\mathsf{KEM}_{\mathsf{HR}} = (\mathsf{Gen}_{\mathsf{HR}}, \mathsf{Encaps}_{\mathsf{HR}}, \mathsf{Decaps}_{\mathsf{HR}})$.

Theorem 7. *Under the* HR *assumption and in the random oracle model, $\mathsf{KEM}_{\mathsf{HR}}$ is an N-receiver non-committing key encapsulation mechanism. In particular, for any N-*NCKE-CCA *adversary $\mathcal{A}$ against $\mathsf{KEM}_{\mathsf{HR}}$ and $\mathsf{Sim}_{\mathsf{HR}}$ that issues at most q_E queries per user to* Encaps*, q_D queries to* Decaps *and at most q_{H} queries to each random*

oracle H_n, $n \in [N]$, *there exists an adversary* $\mathcal{B}$ *against* HR *with roughly the same running time such that*

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE}\text{-}\mathsf{CCA}}_{\mathsf{KEM}_{\mathsf{HR}},\mathsf{Sim}_{\mathsf{HR}}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{B}) + (N \cdot q_E)(q_{\mathsf{H}} + q_D + 1) \cdot O(2^{-\delta n(\kappa)}) + O(2^{-n/2}) \ .$$

Proof. As shown in [HK09], HPS is $\delta n(\kappa)$-entropic. Furthermore, the m-fold subset membership problem is hard in HPS by definition of the m-HR assumption. Thus, Theorem 1 yields

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE}\text{-}\mathsf{CCA}}_{\mathsf{KEM}_{\mathsf{HR}},\mathsf{Sim}_{\mathsf{HR}}}(\mathcal{A}) \leq \mathrm{Adv}^{m\text{-}\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{B}) + (N \cdot q_E \cdot q_{\mathsf{H}}) \cdot 2^{-\delta n(\kappa)} + (N \cdot q_E \cdot q_D) \cdot 2^{-\delta n(\kappa)} \ .$$

Applying Lemma 3 yields the bound stated in Theorem 7. □

B Full Attack Tables for our AKE Model

In the following, we want to give the complete tables of possible attacks in the FS and wFS security model. Note that these tables contains a large number of redundant rows. We do this to justify completeness and point out the security properties. In the next step, we will distill the tables for the special case of two-message protocols, thus reducing complexity. We also point out trivial attacks.

B.1 Overview of Allowed Attacks for Full Forward Security

We begin with the complete table of possible attacks for full forward security, which is given in Table 3. The variables used are explained in Section 4. The structure is as follows:

- Attack (0) covers that it is considered a valid attack when a session is recreated due to insufficient randomness. Therefore, if there is more than one partially matching session to a test session, the adversary may also run a trivial attack.
- Attacks (1)-(8) capture all attacks, where a matching session exists.
- Attacks (9)-(16) capture all attacks, where a partially matching session exists, but no full matching session.
- Attacks (17)-(24) capture all attacks, where neither a partially nor a full matching session exists.

We cover all possible combinations of long-term key corruptions and state reveals, also taking into account when a corruption may happen (modeled by variable peerCorrupted). Thereby, the allowed attacks include forward security, KCI security, and security against maximal exposure:

Forward Security is covered, for instance, by attacks (17) and (18), which enable an active adversary (i.e., $|\mathfrak{M}(\mathrm{sID}^*)| = 0$) to obtain the long-term secret of one or both parties. The peer's long-term secret key will be available after the session key has been computed.

Key Compromise Impersonation is covered by attacks (21)-(24), where the adversary obtains at least the long-term secret of one party.

	$\mathcal{A}$ gets (Initiator, Responder)	corrupted[i^*]	corrupted[r^*]	peerCorrupted[sID*]	type[sID*]	revState[sID*]	$\exists$sID $\in \mathfrak{M}$(sID*) : revState[sID]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$	$\exists$sID $\in \mathfrak{P}$(sID*) : revState[sID]	$\lvert\mathfrak{P}(\text{sID}^*)\rvert$
(0)	**multiple partially matching sessions**	–	–	–	–	–	–	–	–	> 1
(1)	**(long-term, long-term)**	–	–	–	"In"	**F**	**F**	1	–	–
(2)	**(long-term, long-term)**	–	–	–	"Re"	**F**	**F**	1	–	–
(3)	**(state, state)**	**F**	**F**	–	"In"	–	–	1	–	–
(4)	**(state, state)**	**F**	**F**	–	"Re"	–	–	1	–	–
(5)	**(long-term, state)**	–	**F**	–	"In"	**F**	–	1	–	–
(6)	**(long-term, state)**	–	**F**	–	"Re"	–	**F**	1	–	–
(7)	**(state, long-term)**	**F**	–	–	"In"	–	**F**	1	–	–
(8)	**(state, long-term)**	**F**	–	–	"Re"	**F**	–	1	–	–
(9)	**(long-term, long-term)**	–	–	**F**	"In"	**F**	n/a	0	**F**	1
(10)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	**F**	1
(11)	**(state, state)**	**F**	**F**	–	"In"	–	n/a	0	–	1
(12)	**(state, state)**	**F**	**F**	–	"Re"	–	n/a	0	–	1
(13)	**(long-term, state)**	–	**F**	–	"In"	**F**	n/a	0	–	1
(14)	**(long-term, state)**	–	**F**	**F**	"Re"	–	n/a	0	**F**	1
(15)	**(state, long-term)**	**F**	–	**F**	"In"	–	n/a	0	**F**	1
(16)	**(state, long-term)**	**F**	–	–	"Re"	**F**	n/a	0	–	1
(17)	**(long-term, long-term)**	–	–	**F**	"In"	**F**	n/a	0	n/a	0
(18)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	n/a	0
(19)	**(state, state)**	**F**	**F**	–	"In"	–	n/a	0	n/a	0
(20)	**(state, state)**	**F**	**F**	–	"Re"	–	n/a	0	n/a	0
(21)	**(long-term, state)**	–	**F**	–	"In"	**F**	n/a	0	n/a	0
(22)	**(long-term, state)**	–	**F**	**F**	"Re"	–	n/a	0	n/a	0
(23)	**(state, long-term)**	**F**	–	**F**	"In"	–	n/a	0	n/a	0
(24)	**(state, long-term)**	**F**	–	–	"Re"	**F**	n/a	0	n/a	0

Table 3: Full table of attacks for full and weak FS adversaries. For two-message protocols, a partial matching session can only be of type "Re" and we can exclude attacks highlighted in green color. Furthermore, for weak FS adversaries we exclude trivial attacks which are highlighted in blue color. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. "**F**" means "false", "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

Maximal Exposure Attacks are covered by the fact that all combinations of long-term secret and state reveals are allowed, except for those that lead to trivial attacks (e.g., reveal of long-term key and state of the same party). In particular, we allow the adversary to obtain both states in attacks (19) and (20).

Distilled Table for Two-Message Protocols. We consider the full table and give a distilled version in Table 2 in Section 4. The simplifications are due to the following reasons:

- Attacks (1) and (2) can be merged by setting the type to arbitrary.
- Attacks (3) and (4) can be removed as the responder does not have a state and thus this is already captured by attacks (7) and (8).
- Attacks (5) and (6) can be removed for the same reason, they are already captured by attacks (1) and (2).
- Attacks (7) and (8) can be merged by setting the type to arbitrary and allowing to reveal both states as only the initiator's state contains meaningful information.
- Attacks (9), (11), (13) and (15) can be removed as by definition, a partially matching session can never be of type "In".
- Attacks (12) and (14) can be removed as the responder does not have a state and thus these attacks are already captured by attacks (16) and (10).
- Attacks (19) and (22) can be removed as the responder does not have a state and thus this is already captured by attacks (23) and (18).
- Attack (20) can be removed as the adversary can compute the state on his own and thus this is already captured by attack (22).
- Attacks (21) and (24) equal attacks (17) and (18) and can be removed as the adversary can compute the state on his own.

B.2 Overview of Allowed Attacks for Weak Forward Security

Compared to full forward security, weak forward security only provides security against a passive adversary (i.e., it must hold that $|\mathfrak{M}(\mathrm{sID}^*)| = 1$) when both secret keys are revealed. This is the strongest form of forward security that implicitly authenticated protocols can achieve. Our model covers this in attacks (1) and (2).

Distilled Table for wFS and Two-Message Protocols. We use Table 3 and distill it such that it considers weak forward security for two-message protocols. The result is given in Table 1 in Section 4. Column peerCorrupted has been removed as we no longer consider the time when a corruption happens. We justify the removal of trivial attacks as follows:

- Attacks (9), (11), (13) and (15) can be removed as by definition, a partial matching session can never be of type "In".
- Attacks (17) and (23) have to be removed as the adversary can trivially win by obtaining the responder's long-term secret and computing the last message.
- Attacks (18) and (22) have to be removed as the active adversary can trivially win by impersonating the initiator and choosing its own state for the first message.

Furthermore, we do some optimizations:

- Attacks (1) and (2) can be merged by setting the type to arbitrary.
- Attacks (3) and (4) can be removed as the responder does not have a state and thus this is already captured by attacks (7) and (8).

– Attacks (5) and (6) can be removed for the same reason, they are already captured by attacks (1) and (2).
– Attacks (7) and (8) can be merged by setting the type to arbitrary and allowing to reveal both states as only the initiator's state contains meaningful information.
– Attacks (12) and (14) can be removed as the responder does not have a state and thus these attacks are already captured by attacks (16) and (10).
– Attack (20) can be removed as the the responder does not have a state and thus this is already captured by attack (24).

Appendix C

Authenticated Key Exchange and Signatures with Tight Security in the Standard Model

An extended abstract of this article appears in the proceedings of CRYPTO 2021. The following version is the full version of this article which is also available in the IACR ePrint archive, ia.cr/2021/863.

Original Publication

S. Han, T. Jager, E. Kiltz, S. Liu, J. Pan, D. Riepel, S. Schäge. Authenticated Key Exchange and Signatures with Tight Security in the Standard Model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 670–700. Springer, Heidelberg, August 2021. https://doi.org/10.1007/978-3-030-84259-8_23

Authenticated Key Exchange and Signatures with Tight Security in the Standard Model

Shuai Han[1,2], Tibor Jager[3], Eike Kiltz[4], Shengli Liu[1,2,5], Jiaxin Pan[6], Doreen Riepel[4], Sven Schäge[4]

[1] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
{dalen17,slliu}@sjtu.edu.cn
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] Bergische Universität Wuppertal, Germany
tibor.jager@uni-wuppertal.de
[4] Ruhr-Universität Bochum, Germany
{eike.kiltz,doreen.riepel,sven.schaege}@rub.de
[5] Westone Cryptologic Research Center, Beijing 100070, China
[6] Department of Mathematical Sciences, NTNU – Norwegian University of Science and Technology, Trondheim, Norway
jiaxin.pan@ntnu.no

Abstract. We construct the first authenticated key exchange protocols that achieve tight security in the *standard model*. Previous works either relied on techniques that seem to inherently require a random oracle, or achieved only "Multi-Bit-Guess" security, which is not known to compose tightly, for instance, to build a secure channel.

Our constructions are generic, based on digital signatures and key encapsulation mechanisms (KEMs). The main technical challenges we resolve is to determine suitable KEM security notions which on the one hand are strong enough to yield tight security, but at the same time weak enough to be efficiently instantiable in the standard model, based on standard techniques such as universal hash proof systems.

Digital signature schemes with tight multi-user security in presence of adaptive corruptions are a central building block, which is used in all known constructions of tightly-secure AKE with full forward security. We identify a subtle gap in the security proof of the only previously known efficient standard model scheme by Bader *et al.* (TCC 2015). We develop a new variant, which yields the currently most efficient signature scheme that achieves this strong security notion without random oracles and based on standard hardness assumptions.

Keywords: Authenticated key exchange, digital signatures, tightness

1 Introduction

A *tight* security proof establishes a close relation between the security of a cryptosystem and its underlying building blocks, *independent* of deployment parameters such as the

number of users, protocol sessions, issued signatures, etc. This enables a theoretically-sound instantiation with optimal parameters, without the need to compensate a security loss by increasing key lengths or group sizes.

AKE. Authenticated key exchange (AKE) protocols enable two parties to authenticate each other and compute a shared session key. In comparison to many other cryptographic primitives, standard security models for AKE are extremely complex. Following the approach of Bellare-Rogaway [BR94] and Canetti-Krawczyk [CK01], a very strong active adversary is considered, which essentially "carries" all protocol messages between parties running the protocol and thus is able to modify, replace, replay, drop, or inject arbitrary messages. Furthermore, the adversary may adaptively corrupt parties and reveal session keys while adaptively choosing which session(s) to "attack".

Achieving security in such a strong and complex model gives very strong security guarantees, but it also makes *tightness* particularly difficult to achieve. Indeed, most security proofs of AKE protocols are extremely lossy, often even with a *quadratic* security loss in the total number of sessions established over the entire lifetime of the protocol. Considering for instance the huge number of TLS connections per day in practice, this loss may be too large to compensate in practice because the resulting increase of deployment parameters would incur an intolerable performance overhead. Hence, such protocols could not be instantiated in a theoretically-sound way.

Therefore tight security of AKE protocols is a well-established research area, with several known constructions [BHJ+15, GJ18, LLGW20, JKRS21, DJ20, DG20]. As recently pointed out by Jager et al. [JKRS21], some of these constructions [BHJ+15, GJ18, LLGW20] consider a *"Multi-Bit-Guess"* (MBG) security experiment, which is not known to compose tightly with primitives that apply the shared session key, e.g., to build a secure channel. The standard and well established security notion in the context of multiple challenges is *"Single-Bit Guess"* (SBG) security. Unfortunately, the only known constructions in the SBG model [JKRS21, DJ20, DG20] apply proof techniques that seem to inherently require the random oracle model [BR93]. For instance, [JKRS21] is based on non-committing encryption, which is known to be not instantiable without random oracles [Nie02], whereas [DJ20, DG20] use a similar approach based on adaptive reprogramming of the random oracle.

Currently, there exists no AKE protocol which achieves tight security in a standard (SBG) AKE security model, with a security proof in the standard model, without random oracles, not even an impractical one.

Digital Signatures. Digital signatures are a foundational cryptographic primitive and often used to build AKE protocols. All known tightly-secure AKE protocols with full forward security [BHJ+15, GJ18, DJ20, DG20, LLGW20, JKRS21] are based on signatures that provide tight existential unforgeability under chosen-message attacks (EUF-CMA), but in a *multi-user* setting and in the presence of an adversary that may *adaptively corrupt* users to obtain their secret keys (MU-EUF-CMA$^{\mathsf{corr}}$ security [BHJ+15]). It is easy to prove that MU-EUF-CMA$^{\mathsf{corr}}$ security is non-tightly implied by standard EUF-CMA security, but with a linear security loss in the number of users.

The construction of a *tightly* MU-EUF-CMA$^{\mathsf{corr}}$ secure signature scheme has to overcome the following, seemingly paradoxical technical problem. On the one hand, the reduction must be able to output user secret keys to the adversary, to respond to

adaptive secret key corruption queries. However, it cannot apply a guessing argument, as this would incur a tightness loss. Therefore it is forced to "know" the secret keys of *all* users. On the other hand, it must be able to extract a solution to a computationally hard problem from a forgery produced by an adversary. This seems to be in conflict with the fact that the reduction has to know secret keys for all users, as knowledge of the secret key should enable the reduction to compute a "forged" signature by itself, without the adversary. In fact, tight multi-user security is known to be impossible for many signature schemes, for example when the public key uniquely defines the matching secret key [BJLS16].

Several previous works have developed techniques to overcome this seeming paradox [Bad14, BHJ+15, GJ18, DGJL21]. Essentially, their approach is to build schemes where secret keys are not uniquely determined by public parameters, along with a reduction that exploits this to evade the paradox. However, all currently known constructions either use the random oracle model, and therefore cannot be used to build tightly-secure AKE in the standard model, or are based on tree-based signatures [BHJ+15], which yields signatures with hundreds of group elements and thus would incur even more overhead than compensating the security loss with larger parameters. Jumping slightly ahead, we remark that [BHJ+15] also describes a pairing-based signature scheme with short constant-size signatures, but we identify a gap in the security proof. Hence, currently there is no practical signature scheme which achieves tight security in the multi-user setting with adaptive corruptions.

1.1 Contributions

Summarizing the previous paragraphs, we can formulate the following natural questions related to AKE and signatures:

> Do there exist efficient AKEs and signature schemes with tight multi-user security in the standard model?

Tightly-secure signatures. We identify a subtle gap in the MU-EUF-CMA$^{\mathsf{corr}}$ security proof of the scheme from [BHJ+15] with constant-size signatures (namely, $\mathsf{SIG}_{\mathsf{C}}$ in [BHJ+15, Section 2.3]). We did not find a way to close this gap and therefore develop a new variant of this scheme and prove tight MU-EUF-CMA$^{\mathsf{corr}}$ security in the standard model. More precisely, $\mathsf{SIG}_{\mathsf{C}}$ follows the blueprint of the Blazy-Kiltz-Pan (BKP) identity-based encryption scheme [BKP14], and transforms an algebraic message authentication code (MAC) scheme into a signature scheme with pairings. If the MAC is tightly-secure in a model with adaptive corruptions, so is the signature scheme. We notice, however, that their MAC does not achieve tight security with adaptive corruptions since the corruption queries leak too much secret information to the adversary.

To overcome this issue, we borrow recent techniques from tightly-secure hierarchical identity-based encryption schemes [LP19, LP20] to construct a new MAC scheme that can be proven tightly secure under adaptive corruptions. Our construction is based on pairings and general random self-reducible matrix Diffie-Hellman (MDDH) assumptions [EHK+17]. When instantiated based on the $\mathcal{D}_k$-MDDH assumption (e.g., k-Lin), a signature consists of $4k+1$ group elements. That is 5 group elements for $k = 1$ (SXDH).

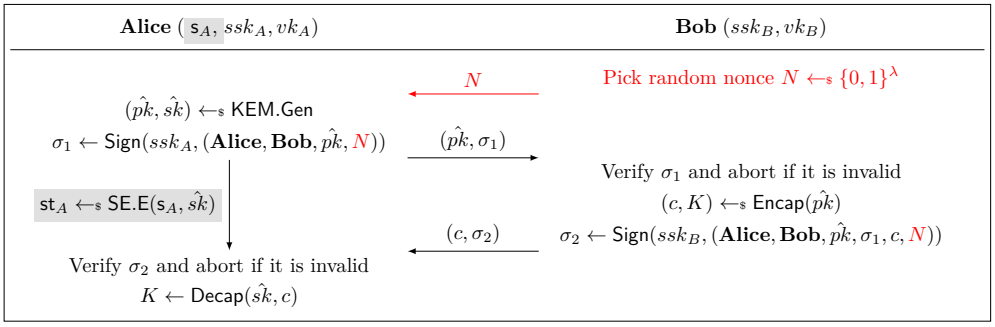

Figure 1: The two-message protocol $\mathsf{AKE}_{\mathsf{2msg}}$ using the "$\mathsf{KEM} + 2 \times \mathsf{SIG}$" approach and the three-message protocols $\mathsf{AKE}_{\mathsf{3msg}}$ (including the red parts) and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ (including the red and gray parts) using the "Nonce + KEM + $2 \times \mathsf{SIG}$" approach. ($\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ additionally uses a symmetric encryption scheme SE.)

This yields the first tightly MU-EUF-CMA$^{\mathsf{corr}}$-secure signature in the standard model with practical efficiency.

We remark that our new signature scheme circumvents known impossibility results for signatures and MACs [BJLS16, MPS20], since these apply only to schemes with re-randomizable signatures or re-randomizable secret keys [BJLS16], or deterministic schemes [MPS20]. Our construction is probabilistic and not efficiently re-randomizable in the sense of [BJLS16].[1]

Tightly-secure AKE in the standard model. The classical *"key encapsulation plus digital signatures"* ($\mathsf{KEM} + 2 \times \mathsf{SIG}$) paradigm to construct AKE protocols gives rise to efficient protocols and is the basis of many constructions, e.g., [CK01, CF12, GJ18, DJ20, DG20, LLGW20, JKRS21]. To establish a session key, two parties Alice and Bob proceed as follows (cf. Figure 1). Alice generates an ephemeral KEM key pair $(\hat{pk}, \hat{sk})$ and sends the signed public key to Bob. Bob then uses this public key to encapsulate a session key, signs the ciphertext, and sends it back to Alice. Alice then obtains the session key K by decapsulating with the KEM secret key. For example, one can view the classical "signed Diffie-Hellman" as a specific instantiation of this paradigm, by considering the Diffie-Hellman protocol as the ElGamal KEM.

Our approach to construct efficient AKE protocols with tight security is based on this $\mathsf{KEM} + 2 \times \mathsf{SIG}$ paradigm. Given a tightly MU-EUF-CMA$^{\mathsf{corr}}$ secure signature scheme, it remains to determine suitable security notions for the underlying KEM, which finds a balance between two properties. The security notion must be *strong enough* to enable a *tight* security proof in presence of adaptive session key reveals and possibly even state reveals. At the same time, it must be *weak enough* to be achievable in the standard model. We now sketch the construction of our three AKE protocols along with the corresponding KEM security notions, see also Figure 2. In terms of AKE security,

[1] Our signatures are only re-randomizable over all strings output by the signing algorithm. The impossibility result from [BJLS16] requires uniform re-randomizability over all strings accepted by the verification algorithm, which does not hold for our scheme.

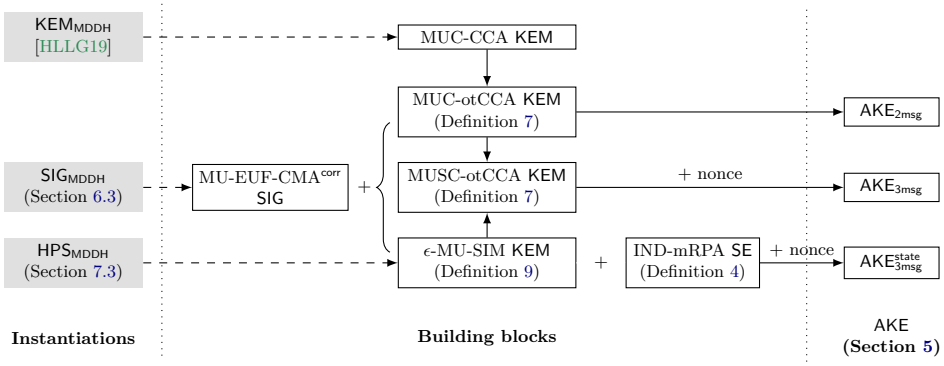

Figure 2: Schematic overview of our AKE constructions.

we consider a generic and versatile security model which provides strong properties, such as full forward security and key-compromise impersonation (KCI) security. "Partnering" of oracles is defined based on *original key partnering* [LS17]. The model is defined in pseudocode to avoid ambiguity.

- Our first result is a new tight security proof for the two-message protocol $\mathsf{AKE}_{\mathsf{2msg}}$, which follows the $\mathsf{KEM}+2\times\mathsf{SIG}$ paradigm. $\mathsf{AKE}_{\mathsf{2msg}}$ is exactly the LLGW protocol [LLGW20] and the main technical difficulty is to adopt the LLGW proof strategy from the "Multi-Bit-Guess" to the standard "Single-Bit-Guess" setting. This requires significant modifications to the proof outline and the underlying KEM security definition. Our new proof relies on Multi-User/Challenge one-time CCA (MUC-otCCA) security for KEMs, allowing the adversary to ask many challenge queries but only one decapsulation query per user. Even though this is a slightly weaker version of the standard Multi-User/Challenge CCA (MUC-CCA) security notion for KEMs (allowing for unbounded decapsulation queries [GHKW16]), the most efficient instantiations we could find are the MUC-CCA-secure schemes with tight security from [GHKW16, GHK17, HLLG19].[2]
- Our second result is a three-message protocol $\mathsf{AKE}_{\mathsf{3msg}}$ resisting replay attacks, which extends the $\mathsf{KEM}+2\times\mathsf{SIG}$ protocol $\mathsf{AKE}_{\mathsf{2msg}}$ with an additional nonce sent at the beginning of the protocol ("$\mathsf{Nonce}+\mathsf{KEM}+2\times\mathsf{SIG}$"). For our security proof we require the KEM security notion of Multi-User Single-Challenge one-time CCA (MUSC-otCCA) security, allowing the adversary to ask only one challenge and one decapsulation query per user. This notion is considerably weaker than MUC-otCCA security and it is achievable from any universal$_2$ hash proof system [CS02]. (For example, based on a standard assumption such as Matrix DDH (MDDH) [EHK+17] which yields highly efficient KEMs.)
- Our third result is a three-message protocol $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$, which extends the $\mathsf{Nonce}+\mathsf{KEM}+2\times\mathsf{SIG}$ protocol $\mathsf{AKE}_{\mathsf{3msg}}$ by encrypting the state with a symmetric encryption (SE) scheme. $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ has tight security in a very strong model that even allows the

[2] We are aware of the generic constructions of bounded-CCA secure KEMs from CPA-secure KEMs [CHH+07], but they do not seem to offer tight security in a multi-challenge setting.

Table 1: Comparison of standard model AKE protocols with full forward security, where T refers to the number of test queries. Protocols $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ and $\mathsf{AKE}_{\mathsf{2msg}}$ refer to our protocols given in Fig. 1, instantiated from $\mathcal{D}_k$-MDDH. The column **Communication** counts the communication complexity of the protocols in terms of the number of group elements, exponents and nonces, where we instantiate all protocols with our new signature scheme from Section 6.3. The column **Security Loss** lists the security loss of the reduction in the "Single-Bit-Guess" (SBG) model, ignoring all symmetric bounds.

Protocol	Communication	#Msg.	Assumption	State Reveal	Security Loss
BHJKL [BHJ+15]	$11 + 11$ $(2k^2 + 6k + 5) + (6k + 9)$	3	SXDH $\mathcal{D}_k$-MDDH	no	$O(\lambda T)$
LLGW [LLGW20]	$9 + 10$ $(k^2 + 7k + 1) + (6k + 4)$	2	SXDH $\mathcal{D}_k$-MDDH	no	$O(\lambda T)$
$\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$	$8 + 7$ $(5k + 3) + (5k + 2)$	3	SXDH $\mathcal{D}_k$-MDDH	yes	$O(\lambda)$
$\mathsf{AKE}_{\mathsf{2msg}}$ (= LLGW)	$9 + 10$ $(k^2 + 7k + 1) + (6k + 4)$	2	SXDH $\mathcal{D}_k$-MDDH	no	$O(\lambda)$

adversary to obtain session states of oracles [CK01]. The fact that the reduction must be able to respond to adaptive queries for session states by an adversary makes it significantly more difficult to achieve *tight* security. Our key technical contribution is a new "Multi-User SIMulatability" (ϵ-MU-SIM) security notion for KEMs, which we also show to be tightly achievable by universal$_2$ hash proof systems. We stress that the reduction to the security of the symmetric encryption scheme is the only part of the security proof which is *not* tight. We tolerate this, since compensating a security loss for symmetric encryption incurs significantly less performance penalty than for public key primitives.[3]

Note that our $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ use nonce to resist replay attacks and admit KEM security with one challenge per user. This can also be achieved generically by assuming synchronized counters between parties, following the approach of [LLGW20]. Consequently, we can also use counter instead of nonce in $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$, and obtain two two-message counter-based AKE protocols which have the same efficiency and security as $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$, respectively.

Instantiations. Table 1 gives example instantiations of our protocols from universal$_2$ hash proof systems from the MDDH assumption and compares them to known protocols. The protocols BHJKL [BHJ+15] and LLGW [LLGW20] only offer tight security in the MBG model which implies security in our standard SBG model with a loss of T, the number of test queries [JKRS21]. For more details on our instantiations we refer to Section 7. Note that there are other works which study AKE in the standard model

[3] For instance, `openssl speed aes` shows that AES-256 is only about 1.5 times slower than AES-128 on a standard laptop computer. Given that the cost of symmetric key operations is already small in comparison to the public key operations, we consider this as negligible.

(e.g., [FSXY12, JKSS12]). However, they do not focus on tightness and have a quadratic security loss.

Technical approach to AKE. In the following, we give a brief overview of our technical approach to tight security under our SBG-type security definition and show how our protocols prevent replay attacks and support state reveals.

To obtain an AKE protocol with a tight security reduction in the $\mathsf{KEM} + 2 \times \mathsf{SIG}$ framework, we rely on the tight MU-EUF-CMA$^{\mathsf{corr}}$ security of the signature scheme to guarantee authentication and deal with corruptions, and on the tight MUC-CCA security of KEM to deal with session key reveals. To this end, recall that the SBG-style security game for MUC-CCA security allows multiple encapsulation and decapsulation queries per user, but considers only a single challenge bit. At the same time, observe that the reduction algorithm can always use the challenge key (which is either the real encapsulated key or a random key) as the session key of the simulated AKE protocol. In combination, these observations immediately lead to a tight security proof for $\mathsf{AKE}_{\mathsf{2msg}}$. We remark that $\mathsf{AKE}_{\mathsf{2msg}}$ can also be proved secure under an even weaker security notion for KEM, namely MUC-otCCA, which allows only one decapsulation query per user. This assumes that parties choose to "close" a session once this session accepts or rejects. In this way we can guarantee that the adversary has only a single opportunity to submit a ciphertext per $\hat{pk}$.

To prevent replay attacks we make use of an exchange of nonces resulting in protocol $\mathsf{AKE}_{\mathsf{3msg}}$. As a byproduct of using nonces (in combination with the signature scheme), we can now guarantee that the adversary cannot replay any message anymore. This includes $\hat{pk}$, and thus we can ensure that the simulator only needs to respond to one encapsulation query per $\hat{pk}$ in the security game. In this way we can further weaken the security requirement that we need from the KEM to MUSC-otCCA.

Now, to support state reveals, we use a symmetric encryption scheme SE that is used to encrypt the ephemeral secret key $\hat{sk}$ of each session, similar to [JKRS21]. More concretely, we require that the state is computed as $\mathsf{st} = \mathsf{SE.E}(\mathsf{s}, \hat{sk})$, where s is the secret key of SE that is made part of the long-term secret key. This modification yields protocol $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$. Having introduced such a state, we now also consider a security model that allows the adversary to issue state reveal queries to obtain the state st. But now the reduction to the MUSC-otCCA security of the KEM cannot work as before, since the reduction algorithm cannot output $\mathsf{SE.E}(\mathsf{s}, \hat{sk})$ to the adversary. A natural way to address this problem is to make use of the security of SE, and make the reduction change the state to an encryption of some dummy random key r , i.e., $\mathsf{st} = \mathsf{SE.E}(\mathsf{s}, r)$. However, now the SE reduction algorithm is faced with a critical decision: If the adversary asks a state reveal query, should the reduction output $\mathsf{st} = \mathsf{SE.E}(\mathsf{s}, \hat{sk})$ or $\mathsf{st} = \mathsf{SE.E}(\mathsf{s}, r)$? It seems that both choices are problematic. If the reduction responds with the encryption of KEM secret key $\hat{sk}$, then the reduction to the KEM will fail in case the adversary asks a test query. If on the other hand the reduction outputs an encryption of a dummy random key, then the reduction will fail in case the adversary queries the secret key via a corrupt query. To solve this problem, the existing approaches rely on a non-committing symmetric encryption scheme that is proven secure in the random oracle model [JKRS21].

To obtain a tight security supporting state reveals in the standard model, we enhance the MUSC-otCCA security of KEM to our new ϵ-MU-SIM-security, so that a symmetric encryption scheme SE with comparatively weak security guarantees suffices. The idea is to rely on a security notion for the symmetric encryption scheme that is as weak as possible while still being able to compensate for this via a stronger KEM. Somewhat surprisingly, our proof shows that when relying on an ϵ-MU-SIM-secure KEM, we only need to require IND-mRPA security (indistinguishability against random plaintext attacks) from SE. Such a symmetric encryption scheme can be easily instantiated using any weakly secure (deterministic) encryption scheme like as AES or even using a weak PRF. Let us now describe ϵ-MU-SIM-secure KEM in slightly more detail. In a nutshell, an ϵ-MU-SIM-secure KEM provides the reduction with access to an additional encapsulation algorithm Encap^* that is keyed with the secret key. We have security requirements as follows:

- Computational indistinguishability between Encap and Encap^*: We require that the reduction can switch to using Encap^* without the adversary noticing even given the secret key $\hat{sk}$ of the KEM. In particular, the resulting indistinguishability notion must tightly reduce to an underlying security assumption.
- Statistical ϵ-uniformity: When using the alternative encapsulation mechanism Encap^*, we require that the encapsulated key in the challenge ciphertext c^* will be indistinguishable from random with *statistical distance* ϵ (even if a decapsulation of some distinct ciphertext $c \neq c^*$ of its choice is given). This is particularly useful when aiming at tight security reductions.
- Since we want to apply ϵ-MU-SIM-secure KEMs in a protocol setting with multiple parties, security must in general hold in a multi-user setting.

Fortunately, such a KEM can be instantiated from universal$_2$ hash proof systems (HPS). In particular, we show that the ϵ-MU-SIM-security is implied by the hardness of subset membership problems and the universal$_2$-property of HPS.

Our new ϵ-MU-SIM-secure KEM now allows us to avoid the above mentioned decision when dealing with state reveals and in this way opens a new avenue towards a tight security reduction. To this end, we use a novel strategy in our security proof.

1. We first switch from using Encap to Encap^*. By the security properties of our KEM, the adversary cannot notice this, even given $\hat{sk}$.
2. Next, we replace the session keys of tested sessions with random keys – one user at a time. We apply a hybrid argument over all users. In the η-th hybrid ($\eta = 1, ..., \mu$ with μ being the number of users), we replace the test session keys related to the η-th user with random keys. We can show that this is not recognizable by the adversary since the key K^* generated by Encap^* is statistically close to uniform even if the adversary gets to see another key for a ciphertext of its choice. We distinguish the following cases.

   Case 1: The adversary corrupts the η-th user. For each session related to this user, the adversary can either reveal the session state or test this session, but not both. If the adversary reveals the state, we do not have to replace the session key at all, so the change is in fact only a conceptual one. If the session is tested, the adversary does not know the state $\mathsf{SE}.\mathsf{E}(\mathsf{s}, \hat{sk})$ and thus we can replace the session key by exploiting the ϵ-uniformity of Encap^*.

Case 2: The adversary does not corrupt the η-th user. In this case, we rely on the IND-mRPA security of SE and replace $\hat{sk}$ in the encrypted state with a random dummy key for this user. Then, we can use ϵ-uniformity to replace all tested keys for that user with random keys, as the state does not contain any information about $\hat{sk}$. After that, we have to switch back to using the original state encryption mechanism and encrypt the real secret key $\hat{sk}$, getting ready for the next hybrid.

After μ hybrids, we change all tested keys to random. At this point the adversary has no advantage in the security game.

Overall, this security proof loses a factor of 2μ – but only when reducing to the IND-mRPA security of the symmetric encryption scheme. All other steps of the proof feature tight security reductions.

2 Preliminaries

Let $\emptyset$ denote an empty string. If x is defined by y or the value of y is assigned to x, we write $x := y$. For $\mu \in \mathbb{N}$, define $[\mu] := \{1, 2, ..., \mu\}$. Denote by $x \leftarrow_{\$} \mathcal{X}$ the procedure of sampling x from set $\mathcal{X}$ uniformly at random. If $\mathcal{D}$ is distribution, $x \leftarrow \mathcal{D}$ means that x is sampled according to $\mathcal{D}$. All our algorithms are probabilistic unless states otherwise. We use $y \leftarrow_{\$} \mathcal{A}(x)$ to define the random variable y obtained by executing algorithm $\mathcal{A}$ on input x. We use $y \in \mathcal{A}(x)$ to indicate that y lies in the support of $\mathcal{A}(x)$. If $\mathcal{A}$ is deterministic we write $y \leftarrow \mathcal{A}(x)$. We also use $y \leftarrow \mathcal{A}(x; r)$ to make the random coins r used in the probabilistic computation explicit. Denote by $\mathbf{T}(\mathcal{A})$ the running time of $\mathcal{A}$. For two distributions X and Y, the statistical distance between them is defined by $\Delta(X; Y) := \frac{1}{2} \cdot \sum_x |\Pr[X = x] - \Pr[Y = x]|$, and conditioned on $Z = z$, the statistical distance between X and Y is denoted by $\Delta(X; Y|Z = z)$. For $0 \leq \epsilon \leq 1$, X and Y are said to be ϵ-close, denoted by $X \approx_\epsilon Y$, if $\Delta(X; Y) \leq \epsilon$.

Definition 1 (Collision-resistant hash functions)**.** *A family of hash functions $\mathcal{H}$ is collision resistant if for any adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{cr}}_{\mathcal{H}}(\mathcal{A}) := \Pr[x_1 \neq x_2 \wedge H(x_1) = H(x_2) | (x_1, x_2) \leftarrow_{\$} \mathcal{A}(H), H \leftarrow_{\$} \mathcal{H}].$$

2.1 Digital Signature

Definition 2 (SIG)**.** *A signature (SIG) scheme* $\mathsf{SIG} = (\mathsf{SIG.Setup}, \mathsf{SIG.Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is defined by the following four algorithms.*

- $\mathsf{SIG.Setup}$: *The setup algorithm outputs a public parameter* $\mathsf{pp}_{\mathsf{SIG}}$*, which defines a message space $\mathcal{M}$, a signature space Σ, and verification key & signing key spaces $\mathcal{VK} \times \mathcal{SK}$.*
- $\mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$: *The key generation algorithm takes as input* $\mathsf{pp}_{\mathsf{SIG}}$ *and outputs a pair of keys $(vk, ssk) \in \mathcal{VK} \times \mathcal{SK}$.*
- $\mathsf{Sign}(ssk, m)$: *Taking as input a signing key ssk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature $\sigma \in \Sigma$.*

– $\mathsf{Ver}(vk, m, \sigma)$: *Taking as input a verification key vk, a message m and a signature σ, the deterministic verification algorithm outputs a bit indicating whether σ is a valid signature for m w.r.t. vk.*

We require that for all $\mathsf{pp}_{\mathsf{SIG}} \in \mathsf{SIG.Setup}$, $(vk, ssk) \in \mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$, *we have* $\mathsf{Ver}(vk, m, \mathsf{Sign}(ssk, m)) = 1$.

Below we present the security notion of existential unforgeability with adaptive corruptions in the multi-user setting (MU-EUF-CMA$^{\mathsf{corr}}$) for SIG, which was originally defined in [BHJ+15].

Definition 3 (MU-EUF-CMA$^{\mathsf{corr}}$ Security for SIG)**.** *To a signature scheme* SIG*, the number of users $\mu \in \mathbb{N}$, and an adversary $\mathcal{A}$ we associate the advantage function* $\mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{A}) := \Pr[\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}} \Rightarrow 1]$*, where* $\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}}$ *is defined in Figure 3.*

$\underline{\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}}}$:
$\mathsf{pp}_{\mathsf{SIG}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}$
For $i \in [\mu]$: $(vk_i, ssk_i) \leftarrow\!\!\$\ \mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$;
$\mathcal{M}_i := \varnothing$ //Record messages from signing queries
$\mathcal{S}^{\mathsf{corr}} := \varnothing$ //Record corruption queries
$(i^*, m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathrm{SIGN}}(\cdot,\cdot),\mathcal{O}_{\mathrm{CORR}}(\cdot)}(\mathsf{pp}_{\mathsf{SIG}}, \mathsf{VKList} := \{vk_i\}_{i\in[\mu]})$

If $(i^* \notin \mathcal{S}^{\mathsf{corr}}) \wedge (m^* \notin \mathcal{M}_{i^*}) \wedge (\mathsf{Ver}(vk_{i^*}, m^*, \sigma^*) = 1)$: Return 1
Else: Return 0

$\underline{\mathcal{O}_{\mathrm{SIGN}}(i, m)}$:
$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(ssk_i, m)$
$\mathcal{M}_i := \mathcal{M}_i \cup \{m\}$
Return σ

$\underline{\mathcal{O}_{\mathrm{CORR}}(i)}$:
$\mathcal{S}^{\mathsf{corr}} := \mathcal{S}^{\mathsf{corr}} \cup \{i\}$
Return ssk_i

Figure 3: The MU-EUF-CMA$^{\mathsf{corr}}$ security experiment $\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}}$ for SIG.

2.2 Symmetric Encryption

Definition 4 (SE)**.** *A symmetric encryption (SE) scheme* $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ *is associated with a key space $\mathcal{K}$, a plaintext space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$. It is defined by the following two algorithms.*

– $\mathsf{E}(k, m)$: *Taking as input a symmetric key $k \in \mathcal{K}$ and a plaintext $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext $c \in \mathcal{C}$.*
– $\mathsf{D}(k, c)$: *Taking as input a symmetric key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, the decryption algorithm outputs a plaintext $m \in \mathcal{M}$.*

We require that for all $k \in \mathcal{K}$, all $m \in \mathcal{M}$, we have $\mathsf{D}(k, \mathsf{E}(k, m)) = m$.

Below we define the indistinguishability against multi-challenge Random Plaintext Attacks (IND-mRPA security) for SE, which asks indistinguishability of encryptions of random plaintexts and encryptions of dummy (random) plaintexts.

Definition 5 (IND-mRPA Security for SE)**.** *To a symmetric encryption scheme* SE*, the number of users $\mu \in \mathbb{N}$, and an adversary $\mathcal{A}$ we associate the advantage function*

$$\mathsf{Adv}^{\mathsf{mrpa}}_{\mathsf{SE},\mu}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}\big(\{m_i, c_i^{(0)}\}_{i\in[\mu]}\big) \Rightarrow 1\right] - \Pr\left[\mathcal{A}\big(\{m_i, c_i^{(1)}\}_{i\in[\mu]}\big) \Rightarrow 1\right]\right|,$$

where $k \leftarrow\!\!\$\ \mathcal{K}$, $m_i, r_i \leftarrow\!\!\$\ \mathcal{M}$, $c_i^{(0)} \leftarrow\!\!\$\ \mathsf{E}(k, m_i)$ *and* $c_i^{(1)} \leftarrow\!\!\$\ \mathsf{E}(k, r_i)$ *for* $\forall i \in [\mu]$.

Remark 1. We note that IND-mRPA is weaker than the traditional IND-CPA (indistinguishability against Chosen Plaintext attacks) security notion. In particular, IND-mRPA is achievable by deterministic SEs, while IND-CPA necessarily requires a probabilistic encryption. Consequently, IND-mRPA secure SE admits more practical instantiations. For example, a PRF or even a weak PRF [NR97] itself is an IND-mRPA secure SE.

3 Security Notions for KEMs

In the section, we present definitions of Key Encapsulation Mechanism (KEM) and its security notions.

Definition 6 (KEM)**.** *A key encapsulation mechanism (KEM) scheme* $\mathsf{KEM} = (\mathsf{KEM.Setup}, \mathsf{KEM.Gen}, \mathsf{Encap}, \mathsf{Decap})$ *consists of four algorithms:*

- $\mathsf{KEM.Setup}$: *The setup algorithm outputs public parameters* $\mathsf{pp}_{\mathsf{KEM}}$*, which determine an encapsulation key space* $\mathcal{K}$*, public key & secret key spaces* $\mathcal{PK} \times \mathcal{SK}$*, and a ciphertext space* $\mathcal{CT}$*.*
- $\mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$: *Taking* $\mathsf{pp}_{\mathsf{KEM}}$ *as input, the key generation algorithm outputs a pair of public key and secret key* $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$*. W.l.o.g., we assume that* $\mathsf{KEM.Gen}$ *first samples* $sk \leftarrow_{\$} \mathcal{SK}$ *uniformly, and then computes* pk *from* sk *deterministically via a polynomial-time algorithm* $\mathsf{KEM.PK}$*, i.e.,* $pk := \mathsf{KEM.PK}(sk)$*. This is reasonable since we can always take the randomness used by* $\mathsf{KEM.Gen}$ *as the secret key.*
- $\mathsf{Encap}(pk)$: *Taking* pk *as input, the encapsulation algorithm outputs a pair of ciphertext* $c \in \mathcal{CT}$ *and encapsulated key* $K \in \mathcal{K}$*.*
- $\mathsf{Decap}(sk, c)$: *Taking as input* sk *and* c*, the deterministic decapsulation algorithm outputs* $K \in \mathcal{K} \cup \{\perp\}$*.*

We require that for all $\mathsf{pp}_{\mathsf{KEM}} \in \mathsf{KEM.Setup}$*,* $(pk, sk) \in \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$*,* $(c, K) \in \mathsf{Encap}(pk)$*, it holds that* $\mathsf{Decap}(sk, c) = K$*.*

We define two security notions for KEMs, the first one in the Multi-User/Challenge (MUC) setting, the second one in the Multi-User and Single Challenge (MUSC) setting. Both notions only allow for one single decapsulation query per user.

Definition 7 (MUC-otCCA/MUSC-otCCA Security for KEM)**.** *To* KEM*, the number of users* $\mu \in \mathbb{N}$*, and an adversary* $\mathcal{A}$ *we associate the advantage functions* $\mathsf{Adv}^{\mathsf{muc\text{-}otcca}}_{\mathsf{KEM},\mu}(\mathcal{A}) := \left|\Pr[\mathsf{Exp}^{\mathsf{muc\text{-}otcca}}_{\mathsf{KEM},\mu,\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|$ *and* $\mathsf{Adv}^{\mathsf{musc\text{-}otcca}}_{\mathsf{KEM},\mu}(\mathcal{A}) := \left|\Pr[\mathsf{Exp}^{\mathsf{musc\text{-}otcca}}_{\mathsf{KEM},\mu,\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|$*, where the experiments are defined in Figure 4.*

Below we recall the definition of the *diversity property* from [LLGW20].

Definition 8 (γ-Diversity of KEM)**.** *A KEM scheme* KEM *is called* γ*-diverse if for all* $\mathsf{pp}_{\mathsf{KEM}} \in \mathsf{KEM.Setup}$*, it holds that*

$$\Pr\left[\begin{array}{c}(pk, sk) \leftarrow_{\$} \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}});\\ r, r' \leftarrow_{\$} \mathcal{R}; (c, K) \leftarrow \mathsf{Encap}(pk; r); (c', K') \leftarrow \mathsf{Encap}(pk; r')\end{array} : K = K'\right] \leq 2^{-\gamma},$$

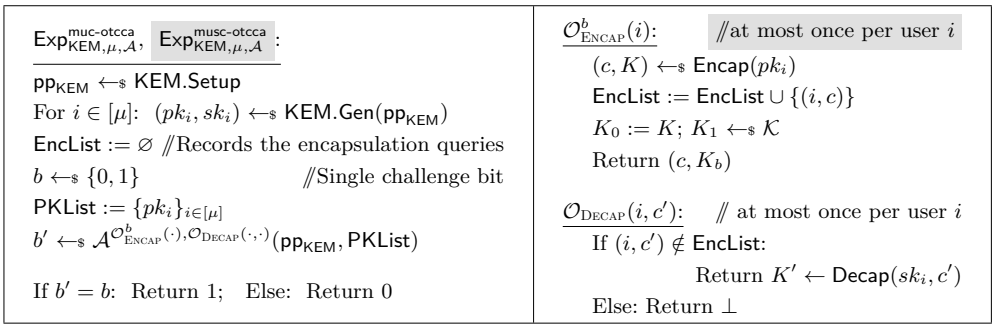

Figure 4: The MUC-otCCA security experiment $\mathsf{Exp}^{\mathsf{muc\text{-}otcca}}_{\mathsf{KEM},\mu,\mathcal{A}}$ and the MUSC-otCCA security experiment $\mathsf{Exp}^{\mathsf{musc\text{-}otcca}}_{\mathsf{KEM},\mu,\mathcal{A}}$ of KEM, where in the latter the adversary can query the encapsulation oracle only once for each user.

$$\Pr\left[\begin{array}{c}(pk, sk) \leftarrow\!\!\$\ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}); (pk', sk') \leftarrow\!\!\$\ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}});\\ r \leftarrow\!\!\$\ \mathcal{R}; (c, K) \leftarrow \mathsf{Encap}(pk; r); (c', K') \leftarrow \mathsf{Encap}(pk'; r)\end{array} : K = K'\right] \leq 2^{-\gamma},$$

where $\mathcal{R}$ is the randomness space of Encap*. If $\gamma = \log|\mathcal{K}|$, then* KEM *is perfectly diverse.*

We also propose a new security notion for KEMs called ϵ-MU-SIM (ϵ-multi-user simulatable) security. Jumping ahead, ϵ-MU-SIM secure KEMs will serve as the main building block in our generic AKE construction with state reveal later. We present the formal definition of ϵ-MU-SIM security (Definition 9) and in Section 7.2, we present simple constructions of ϵ-MU-SIM secure KEMs from universal$_2$-HPS.

Informally, ϵ-MU-SIM security requires that there exists a simulated encapsulation algorithm $\mathsf{Encap}^*(sk)$ which returns simulated ciphertext/key pairs (c^*, K^*) satisfying the following two properties. Firstly, they should be computationally indistinguishable from real ciphertext/key pairs. Secondly, given c^* and an arbitrary single decryption query, the simulated key K^* should be ϵ-close to uniform.

Definition 9 (ϵ-MU-SIM Security for KEM)**.** *We require that there exists a simulated encapsulation algorithm* $\mathsf{Encap}^*(sk)$ *which takes the secret key sk as input, and outputs a pair of simulated $c^* \in \mathcal{CT}$ and simulated $K^* \in \mathcal{K}$. For ϵ-uniformity we require that for any (unbounded) adversary $\mathcal{A}$, it holds that*

$$\begin{array}{l}\Big|\ \Pr[c \leftarrow\!\!\$\ \mathcal{A}(pk, c^*, K^*)\ :\ c \neq c^* \wedge \mathcal{A}(pk, c^*, K^*, \mathsf{Decap}(sk, c)) \Rightarrow 1]\\ -\ \Pr[c \leftarrow\!\!\$\ \mathcal{A}(pk, c^*, R)\ :\ c \neq c^* \wedge \mathcal{A}(pk, c^*, R, \mathsf{Decap}(sk, c)) \Rightarrow 1]\ \Big| \leq \epsilon,\end{array} \tag{1}$$

where the probability is over $\mathsf{pp}_{\mathsf{KEM}} \leftarrow\!\!\$\ \mathsf{KEM.Setup}$*,* $(pk, sk) \leftarrow\!\!\$\ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$*,* $(c^*, K^*) \leftarrow\!\!\$\ \mathsf{Encap}^*(sk)$*, $R \leftarrow\!\!\$\ \mathcal{K}$ and the internal randomness of $\mathcal{A}$.*

Furthermore, to KEM*, a simulated encapsulation algorithm* Encap^**, an adversary $\mathcal{A}$, and $\mu \in \mathbb{N}$ we associate the advantage function* $\mathsf{Adv}^{\mathsf{mu\text{-}sim}}_{\mathsf{KEM},\mathsf{Encap}^*,\mu}(\mathcal{A}) :=$

$$\left|\Pr\left[\mathcal{A}(\{pk_i, sk_i, c_i^{(0)}, K_i^{(0)}\}_{i\in[\mu]}) \Rightarrow 1\right] - \Pr\left[\mathcal{A}(\{pk_i, sk_i, c_i^{(1)}, K_i^{(1)}\}_{i\in[\mu]}) \Rightarrow 1\right]\right|, \tag{2}$$

where $\mathsf{pp}_{\mathsf{KEM}} \leftarrow\!\!\$\ \mathsf{KEM.Setup}$*,* $(pk_i, sk_i) \leftarrow\!\!\$\ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$*,* $(c_i^{(0)}, K_i^{(0)}) \leftarrow\!\!\$\ \mathsf{Encap}(pk_i)$*, and* $(c_i^{(1)}, K_i^{(1)}) \leftarrow\!\!\$\ \mathsf{Encap}^*(sk_i)$ *for $\forall i \in [\mu]$.*

Note that ϵ-MU-SIM security tightly implies MUSC-otCCA$^{\mathsf{rev\&corr}}$ security which is a stronger variant of MUSC-otCCA security supporting key reveal and user corrupt queries. Reveal and corrupt queries can be tolerated since in the security experiment (2), adversary $\mathcal{A}$ also obtains secret keys $sk_1, \ldots, sk_\mu$. By (1) one can see that one single decapsulation query is supported. In particular, ϵ-MU-SIM security tightly implies MUSC-otCCA security. In Section 7 we will define universal$_2$ hash proof systems and show how they imply ϵ-MU-SIM secure KEMs.

4 Authenticated Key Exchange

4.1 Definition of Authenticated Key Exchange

Definition 10 (AKE). *An authenticated key exchange (AKE) scheme* $\mathsf{AKE}=(\mathsf{AKE.Setup}, \mathsf{AKE.Gen}, \mathsf{AKE.Protocol})$ *consists of two probabilistic algorithms and an interactive protocol.*

- $\mathsf{AKE.Setup}$: *The setup algorithm outputs the public parameter* $\mathsf{pp}_{\mathsf{AKE}}$.
- $\mathsf{AKE.Gen}(\mathsf{pp}_{\mathsf{AKE}}, P_i)$: *The generation algorithm takes as input* $\mathsf{pp}_{\mathsf{AKE}}$ *and a party* P_i*, and outputs a key pair* (pk_i, sk_i).
- $\mathsf{AKE.Protocol}(P_i(\mathsf{res}_i) \rightleftharpoons P_j(\mathsf{res}_j))$: *The protocol involves two parties* P_i *and* P_j*, who have access to their own resources,* $\mathsf{res}_i := (sk_i, \mathsf{pp}_{\mathsf{AKE}}, \{pk_u\}_{u\in[\mu]})$ *and* $\mathsf{res}_j := (sk_j, \mathsf{pp}_{\mathsf{AKE}}, \{pk_u\}_{u\in[\mu]})$*, respectively. Here* μ *is the total number of users. After execution,* P_i *outputs a flag* $\Psi_i \in \{\emptyset, \mathbf{accept}, \mathbf{reject}\}$*, and a session key* k_i *(*k_i *might be the empty string* $\emptyset$*), and* P_j *outputs* (Ψ_j, k_j) *similarly.*

Correctness of AKE. For any distinct and honest parties P_i and P_j, they share the same session key after the execution of $\mathsf{AKE.Protocol}(P_i(\mathsf{res}_i) \rightleftharpoons P_j(\mathsf{res}_j))$, i.e., $\Psi_i = \Psi_j = \mathbf{accept}, k_i = k_j \neq \emptyset$.

4.2 Security Model of AKE

We will adapt the security model formalized by [BHJ+15, LS17, GJ18], which in turn followed the model proposed by Bellare and Rogaway [BR94]. We also include replay attacks [LLGW20] and multiple test queries with respect to the same random bit [JKRS21].

First, we will define oracles and their static variables in the model. Then we describe the security experiment and the corresponding security notions.

Oracles. Suppose there are at most μ users $P_1, P_2, ..., P_\mu$, and each user will involve at most ℓ instances. P_i is formalized by a series of oracles, $\pi_i^1, \pi_i^2, ..., \pi_i^\ell$. Oracle π_i^s formalizes P_i's execution of the s-th protocol instance.

Each oracle π_i^s has access to P_i's resource $\mathsf{res}_i := (sk_i, \mathsf{pp}_{\mathsf{AKE}}, \mathsf{PKList} := \{pk_u\}_{u\in[\mu]})$. π_i^s also has its own variables $\mathsf{var}_i^s := (\mathsf{st}_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$.

- st_i^s: State information that has to be stored between two rounds in order to execute the protocol.
- Pid_i^s: The intended communication peer's identity.

- $k_i^s \in \mathcal{K}$: The session key computed by π_i^s. Here $\mathcal{K}$ is the session key space. We assume that $\emptyset \in \mathcal{K}$.
- $\Psi_i^s \in \{\emptyset, \mathbf{accept}, \mathbf{reject}\}$: Ψ_i^s indicates whether π_i^s has completed the protocol execution and accepted k_i^s.

At the beginning, $(\mathsf{st}_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$ are initialized to $(\emptyset, \emptyset, \emptyset, \emptyset)$. We declare that $k_i^s \neq \emptyset$ if and only if $\Psi_i^s = \mathbf{accept}$.

Security Experiment. To define the security notion of AKE, we first formalize the security experiment $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ with the help of the oracles defined above. $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ is a game played between an AKE challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. $\mathcal{C}$ will simulate the executions of the ℓ protocol instances for each of the μ users with oracles π_i^s. We give a formal description in Figure 5.

Adversary $\mathcal{A}$ may copy, delay, erase, replay, and interpolate the messages transmitted in the network. This is formalized by the query Send to oracle π_i^s. With Send, $\mathcal{A}$ can send arbitrary messages to any oracle π_i^s. Then π_i^s will execute the AKE protocol according to the protocol specification for P_i. The $\mathsf{StateReveal}(i, s)$ oracle allows $\mathcal{A}$ to reveal π_i^s's session state.

We also allow the adversary to observe session keys of its choices. This is reflected by a $\mathsf{SessionKeyReveal}$ query to oracle π_i^s.

A $\mathsf{Corrupt}$ query allows $\mathcal{A}$ to corrupt a party P_i and get its long-term secret key sk_i. With a $\mathsf{RegisterCorrupt}$ query, $\mathcal{A}$ can register a new party without public key certification. The public key is then known to all other users.

We introduce a Test query to formalize the pseudorandomness of k_i^s. Therefore, the challenger chooses a bit $b \leftarrow\!\!\$\ \{0,1\}$ at the beginning of the experiment. When $\mathcal{A}$ issues a Test query for π_i^s, the oracle will return $\perp$ if the session key k_i^s is not generated yet. Otherwise, π_i^s will return k_i^s or a truly random key, depending on b. The task of $\mathcal{A}$ is to tell whether the key is the true session key or a random key. The adversary is allowed to make multiple test queries.

Formally, the queries by $\mathcal{A}$ are described as follows.

- $\mathsf{Send}(i, s, j, \mathsf{msg})$: If $\mathsf{msg} = \top$, it means that $\mathcal{A}$ asks oracle π_i^s to send the first protocol message to P_j. Otherwise, $\mathcal{A}$ impersonates P_j to send message msg to π_i^s. Then π_i^s executes the AKE protocol with msg as P_i does, computes a message msg', and updates its own variables $\mathsf{var}_i^s = (\mathsf{st}_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$. The output message msg' is returned to $\mathcal{A}$.
  If $\mathsf{Send}(i, s, j, \mathsf{msg})$ is the τ-th query asked by $\mathcal{A}$ and π_i^s changes Ψ_i^s to $\mathbf{accept}$ after that, then we say that π_i^s is *τ-accepted*.
- $\mathsf{Corrupt}(i)$: $\mathcal{C}$ reveals party P_i's long-term secret key sk_i to $\mathcal{A}$. After corruption, $\pi_i^1, ..., \pi_i^\ell$ will stop answering any query from $\mathcal{A}$.
  If $\mathsf{Corrupt}(i)$ is the τ-th query asked by $\mathcal{A}$, we say that P_i is *τ-corrupted*.
  If $\mathcal{A}$ has never asked $\mathsf{Corrupt}(i)$, we say that P_i is *∞-corrupted*.
- $\mathsf{RegisterCorrupt}(i, pk_i)$: It means that $\mathcal{A}$ registers a new party P_i $(i > \mu)$. $\mathcal{C}$ distributes (P_i, pk_i) to all users. In this case, we say that P_i is *0-corrupted*.
- $\mathsf{StateReveal}(i, s)$: The query means that $\mathcal{A}$ asks $\mathcal{C}$ to reveal π_i^s's session state. $\mathcal{C}$ returns st_i^s to $\mathcal{A}$.

$\underline{\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}}$:
$\mathsf{pp}_{\mathsf{AKE}} \leftarrow \mathsf{AKE.Setup}$
For $i \in [\mu]$:
  $(pk_i, sk_i) \leftarrow \mathsf{AKE.Gen}(\mathsf{pp}_{\mathsf{AKE}}, P_i)$;
  $crp_i := \mathbf{false}$ //Corruption variable
$\mathsf{PKList} := \{pk_i\}_{i\in[\mu]}$; $b \leftarrow_{\$} \{0,1\}$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{st}_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s) := (\emptyset,\emptyset,\emptyset,\emptyset)$;
  $\mathsf{Aflag}_i^s := \mathbf{false}$ //Whether Pid_i^s is corrupted when π_i^s accepts
  $T_i^s := \mathbf{false}$; $kRev_i^s := \mathbf{false}$ // Test, Key Reveal variables
  $stRev_i^s := \mathbf{false}$, $FirstAcc_i^s := \emptyset$
  // State Reveal & First Acceptance variables
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AKE}}(\cdot)}(\mathsf{pp}_{\mathsf{AKE}}, \mathsf{PKList})$

$\mathsf{Win}_{\mathsf{Auth}} := \mathbf{false}$
$\mathsf{Win}_{\mathsf{Auth}} := \mathbf{true}$, If $\exists (i,s) \in [\mu] \times [\ell]$ s.t.
(1) $\Psi_i^s = \mathbf{accept}$ //π_i^s is τ-accepted
(2) $\mathsf{Aflag}_i^s = \mathbf{false}$ //P_j is $\hat{\tau}$-corrupted with $j := \mathsf{Pid}_i^s$ and $\hat{\tau} > \tau$
(3) (3.1) $\vee$ (3.2) $\vee$ (3.3). Let $j := \mathsf{Pid}_i^s$
  (3.1) $\nexists\, t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
  (3.2) $\exists\, t \in [\ell], (j',t') \in [\mu] \times [\ell]$ with $(j,t) \neq (j',t')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \wedge \mathsf{Partner}(\pi_i^s \leftarrow \pi_{j'}^{t'})$
  (3.3) $\exists\, t \in [\ell], (i',s') \in [\mu] \times [\ell]$ with $(i,s) \neq (i',s')$ s.t.
    $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) \wedge \mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$ //Replay attacks

$\mathsf{Win}_{\mathsf{Ind}} := \mathbf{false}$
If $b^* = b$:
  $\mathsf{Win}_{\mathsf{Ind}} := \mathbf{true}$; Return 1
Else: Return 0

$\underline{\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)}$: //Checking whether $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$
If π_i^s sent the first message and $k_i^s = \mathsf{K}(\pi_i^s, \pi_j^t) \neq \emptyset$: Return 1
If π_i^s received the first message and $k_i^s = \mathsf{K}(\pi_j^t, \pi_i^s) \neq \emptyset$: Return 1
Return 0

$\underline{\pi_i^s(\mathsf{msg}, j)}$:
//π_i^s executes AKE according to the protocol specification
If $\mathsf{Pid}_i^s = \emptyset$: $\mathsf{Pid}_i^s := j$
If $\mathsf{Pid}_i^s = j$:
  π_i^s receives msg and uses $\mathsf{res}_i, \mathsf{var}_i^s$ to generate the next message msg' of AKE, and updates $(\mathsf{st}_i^s, \mathsf{Pid}_i^s, k_i^s, \Psi_i^s)$;
  If $\mathsf{msg} = \top$: π_i^s generates the first message msg' as initiator;
  If msg is the last message of AKE: $\mathsf{msg}' := \emptyset$;
  Return msg'
If $\mathsf{Pid}_i^s \neq j$: Return $\perp$

$\underline{\mathcal{O}_{\mathsf{AKE}}(\mathsf{query})}$:
If query=RegisterCorrupt(u, pk_u):
  If $u \in [\mu]$: Return $\perp$
  $\mathsf{PKList} := \mathsf{PKList} \cup \{pk_u\}$
  $crp_u := \mathbf{true}$
  Return PKList

$\underline{\mathcal{O}_{\mathsf{AKE}}(\mathsf{query})}$:
If query=Send(i,s,j,msg):
  If $\Psi_i^s = \mathbf{accept}$: Return $\perp$
  $\mathsf{msg}' \leftarrow \pi_i^s(\mathsf{msg}, j)$
  If $\Psi_i^s = \mathbf{accept}$:
    If $crp_j = \mathbf{true}$: $\mathsf{Aflag}_i^s := \mathbf{true}$;
    // Determine whether π_i^s accepts before its partner
    If $crp_j = \mathbf{false} \wedge \exists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$:
      If $\Psi_j^t \neq \mathbf{accept}$:
        $FirstAcc_i^s := \mathbf{true}$; $FirstAcc_j^t := \mathbf{false}$
      If $\Psi_j^t = \mathbf{accept}$:
        $FirstAcc_i^s := \mathbf{false}$; $FirstAcc_j^t := \mathbf{true}$
  Return msg'

If query=Corrupt(i):
  If $i \notin [\mu]$: Return $\perp$
  For $s \in [\ell]$
    If $FirstAcc_i^s = \mathbf{false} \wedge stRev_i^s = \mathbf{true}$:
      If $T_i^s = \mathbf{true}$: Return $\perp$; //avoid **TA6**
      If $\exists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_j^t \leftarrow \pi_i^s)$:
        If $T_j^t = \mathbf{true}$: Return $\perp$; //avoid **TA7**
  $crp_i := \mathbf{true}$
  Return sk_i

If query=SessionKeyReveal(i,s):
  If $\Psi_i^s \neq \mathbf{accept}$: Return $\perp$
  If $T_i^s = \mathbf{true}$: Return $\perp$ //avoid **TA2**
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$:
    If $T_j^t = \mathbf{true}$: Return $\perp$ //avoid **TA4**
  $kRev_i^s := \mathbf{true}$; Return k_i^s

If query=StateReveal(i,s)
  If $FirstAcc_i^s = \mathbf{false} \wedge crp_i = \mathbf{true}$:
    If $T_i^s = \mathbf{true}$: Return $\perp$; //avoid **TA6**
    Let $j := \mathsf{Pid}_i^s$
    If $\exists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_j^t \leftarrow \pi_i^s)$:
      If $T_j^t = \mathbf{true}$: Return $\perp$; //avoid **TA7**
  $stRev_i^s := \mathbf{true}$; Return st_i^s

If query=Test(i,s):
  If $\Psi_i^s \neq \mathbf{accept} \vee \mathsf{Aflag}_i^s = \mathbf{true} \vee kRev_i^s = \mathbf{true}$
    $\vee\ T_i^s = \mathbf{true}$: Return $\perp$ //avoid **TA1**, **TA2**, **TA3**
  If $FirstAcc_i^s = \mathbf{false}$:
    If $crp_i = \mathbf{true} \wedge stRev_i^s = \mathbf{true}$:
      Return $\perp$ //avoid **TA6**
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$:
    If $kRev_j^t = \mathbf{true} \vee T_j^t = \mathbf{true}$:
      Return $\perp$ //avoid **TA4**, **TA5**
  If $\exists t \in [\ell]$ s.t. $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$:
    If $FirstAcc_j^t = \mathbf{false} \wedge crp_j = \mathbf{true}$
      $\wedge\ stRev_j^t = \mathbf{true}$: Return $\perp$ //avoid **TA7**
  $T_i^s := \mathbf{true}$; $k_0 := k_i^s$; $k_1 \leftarrow_{\$} \mathcal{K}$; Return k_b

Figure 5: The security experiments $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$, $\boxed{\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}}$ (both without red text) and $\boxed{\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}}$ (with red text). The list of trivial attacks is given in Table 2.

- $\mathsf{SessionKeyReveal}(i, s)$: The query means that $\mathcal{A}$ asks $\mathcal{C}$ to reveal π_i^s's session key. If $\Psi_i^s \neq \textbf{accept}$, $\mathcal{C}$ returns $\perp$. Otherwise, $\mathcal{C}$ returns the session key k_i^s of π_i^s.
- $\mathsf{Test}(i, s)$: If $\Psi_i^s \neq \textbf{accept}$, $\mathcal{C}$ returns $\perp$. Otherwise, $\mathcal{C}$ sets $k_0 = k_i^s$, samples $k_1 \leftarrow_\$ \mathcal{K}$, and returns k_b to $\mathcal{A}$. We require that $\mathcal{A}$ can ask $\mathsf{Test}(i, s)$ to each oracle π_i^s only once.

Informally, the pseudorandomness of k_i^s asks that any PPT adversary $\mathcal{A}$ with access to $\mathsf{Test}(i, s)$ cannot distinguish k_i^s from a uniformly random key. Yet, we have to exclude some trivial attacks. We will define them later and first introduce partnering.

Definition 11 (Original Key [LS17])**.** *For two oracles π_i^s and π_j^t, the original key, denoted as $\mathsf{K}(\pi_i^s, \pi_j^t)$, is the session key computed by the two peers of the protocol under a passive adversary only, where π_i^s is the initiator.*

Remark 2. We note that $\mathsf{K}(\pi_i^s, \pi_j^t)$ is determined by the identities of P_i and P_j and the internal randomness.

Definition 12 (Partner [LS17])**.** *Let $\mathsf{K}(\cdot, \cdot)$ denote the original key function. We say that an oracle π_i^s is partnered to π_j^t, denoted as $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$[3], if one of the following requirements holds:*

- *π_i^s has sent the first message and $k_i^s = \mathsf{K}(\pi_i^s, \pi_j^t) \neq \emptyset$, or*
- *π_i^s has received the first message and $k_i^s = \mathsf{K}(\pi_j^t, \pi_i^s) \neq \emptyset$.*

We write $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$ if $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ and $\mathsf{Partner}(\pi_j^t \leftarrow \pi_i^s)$.

Trivial Attacks. In order to prevent the adversary from trivial attacks, we keep track of the following variables for each party P_i and oracle π_i^s:

- crp_i: whether P_i is corrupted.
- Aflag_i^s: whether the intended partner is corrupted when π_i^s accepts.
- T_i^s: whether π_i^s was tested.
- $kRev_i^s$: whether the session key k_i^s was revealed.
- $stRev_i^s$: whether the session state st_i^s was revealed.
- $FirstAcc_i^s$: whether P_i or its partner is the first to accept the key in the session.

Based on that we give a list of trivial attacks **TA1**-**TA7** in Table 2.

Remark 3. We introduced variable $FirstAcc$ to indicate whether the party or its partner is the first to accept the key. This is used to determine whether the state of an oracle is allowed to be revealed when the oracle or its partner is tested.

- In general, the session key of the party which accepts the key after its partner (i.e., $FirstAcc = \textbf{false}$), by the correctness of AKE, must be identical to its partner's. Thus, the session key is fully determined by the state and long-term key of that party (as well as transcripts).
- However, the session key of the party which accepts the key before its partner (i.e., $FirstAcc = \textbf{true}$) might involve fresh randomness beyond its state and long-term key.

[3] The arrow notion $\pi_i^s \leftarrow \pi_j^t$ means π_i^s (not necessarily π_j^t) has computed and accepted the original key.

Table 2: Trivial attacks **TA1**-**TA7** for security experiments $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$, $\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$, where **TA6** and **TA7** are only defined in $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$. Note that "$\mathsf{Aflag}_i^s = \textbf{false}$" is implicitly contained in **TA2**-**TA7** because of **TA1**.

Types	Trivial attacks	Explanation
TA1	$T_i^s = \textbf{true} \wedge \mathsf{Aflag}_i^s = \textbf{true}$	π_i^s is tested but π_i^s's partner is corrupted when π_i^s accepts session key k_i^s
TA2	$T_i^s = \textbf{true} \wedge kRev_i^s = \textbf{true}$	π_i^s is tested and its session key k_i^s is revealed
TA3	$T_i^s = \textbf{true}$ when $\mathsf{Test}(i,s)$ is queried	$\mathsf{Test}(i,s)$ is queried at least twice
TA4	$T_i^s = \textbf{true} \wedge \mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t) \wedge kRev_j^t = \textbf{true}$	π_i^s is tested, π_i^s and π_j^t are partnered to each other, and π_j^t's session key k_j^t is revealed
TA5	$T_i^s = \textbf{true} \wedge \mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t) \wedge T_j^t = \textbf{true}$	π_i^s is tested, π_i^s and π_j^t are partnered to each other, and π_j^t is tested
TA6	$T_i^s = \textbf{true} \wedge FirstAcc_i^s = \textbf{false}$ $\wedge\ stRev_i^s = \textbf{true} \wedge crp_i = \textbf{true}$	π_i^s is tested, π_i^s accepts its key after its partner, and π_i^s is both corrupted and has its state st_i^s revealed
TA7	$T_i^s = \textbf{true} \wedge \mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ $\wedge\ FirstAcc_j^t = \textbf{false}$ $\wedge\ crp_j = \textbf{true} \wedge stRev_j^t = \textbf{true}$	π_i^s is tested, π_i^s accepts its session key before its partner, but its partner π_j^t is both corrupted and state revealed

Thus, it is a trivial attack to reveal the state and the long-term key of the same oracle, if the oracle or its partner is tested and the oracle accepts the key after its partner (i.e., $FirstAcc = \textbf{false}$). This is a minimal trivial attack regarding state reveal[4], and it is formalized as **TA6** and **TA7** in Table 2.

The following definition also captures replay attacks. Given $\mathsf{Partner}(\pi_{i'}^{s'} \leftarrow \pi_j^t)$, a successful replay attack means that $\mathcal{A}$ resends to π_i^s the messages, which were sent from π_j^t to $\pi_{i'}^{s'}$, and π_i^s is fooled to compute a session key, i.e., $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$. Note that a stateless 2-pass AKE protocol cannot be secure against replay attacks [LLGW20]. Therefore, we also define security without replay attacks in Definition 15.

Furthermore, we distinguish between security with state reveals (Definition 13) and without state reveals (Definition 14), where in the latter the adversary cannot query the session state of an oracle.

Definition 13 (Security of AKE with Replay Attacks and State Reveal)**.** *Let μ be the number of users and ℓ the maximum number of protocol executions per user. The security experiment $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ (see Fig. 5) is played between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.*

1. *$\mathcal{C}$ runs $\mathsf{AKE.Setup}$ to get AKE public parameter $\mathsf{pp}_{\mathsf{AKE}}$.*
2. *For each party P_i, $\mathcal{C}$ runs $\mathsf{AKE.Gen}(\mathsf{pp}_{\mathsf{AKE}}, P_i)$ to get the long-term key pair (pk_i, sk_i). Next it chooses a random bit $b \leftarrow_{\$} \{0,1\}$ and provides $\mathcal{A}$ with the public parameter $\mathsf{pp}_{\mathsf{AKE}}$ and the list of public keys $\mathsf{PKList} := \{pk_i\}_{i\in[\mu]}$.*
3. *$\mathcal{A}$ asks $\mathcal{C}$ Send, $\mathsf{Corrupt}$, $\mathsf{RegisterCorrupt}$, $\mathsf{SessionKeyReveal}$, $\mathsf{StateReveal}$ and Test queries adaptively.*

[4] It is also possible to define the trivial attack regardless of $FirstAcc$, but our definition of **TA6** and **TA7** is minimal and makes our security model stronger.

4. At the end of the experiment, $\mathcal{A}$ terminates with an output b^*.

- **Strong Authentication.** *Let* $\mathsf{Win}_{\mathsf{Auth}}$ *denote the event that* $\mathcal{A}$ *breaks authentication in the security experiment.* $\mathsf{Win}_{\mathsf{Auth}}$ *happens iff* $\exists (i,s) \in [\mu] \times [\ell]$ *s.t.*
  (1) π_i^s *is* τ*-accepted.*
  (2) P_j *is* $\hat{\tau}$*-corrupted with* $j := \mathsf{Pid}_i^s$ *and* $\hat{\tau} > \tau$.
  (3) *Either (3.1) or (3.2) or (3.3) happens*[5]*. Let* $j := \mathsf{Pid}_i^s$.
    (3.1) *There is no oracle* π_j^t *that* π_i^s *is partnered to.*
    (3.2) *There exist two distinct oracles* π_j^t *and* $\pi_j^{t'}$*, to which* π_i^s *is partnered.*
    (3.3) *There exist two oracles* $\pi_{i'}^{s'}$ *and* π_j^t *with* $(i',s') \neq (i,s)$*, such that both* π_i^s *and* $\pi_{i'}^{s'}$ *are partnered to* π_j^t.
- **Indistinguishability.** *Let* $\mathsf{Win}_{\mathsf{Ind}}$ *denote the event that* $\mathcal{A}$ *breaks indistinguishability in the experiment* $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *above. Let* b^* *be* $\mathcal{A}$*'s output. Then* $\mathsf{Win}_{\mathsf{Ind}}$ *happens iff* $b^* = b$*. Trivial attacks are already considered during the execution of the experiment. A list of trivial attacks is given in Table 2.*

Note that $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}} \Rightarrow 1$ *iff* $\mathsf{Win}_{\mathsf{Ind}}$ *happens. Hence, the advantage of* $\mathcal{A}$ *is defined as*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell}(\mathcal{A}) :&= \max\{\Pr[\mathsf{Win}_{\mathsf{Auth}}], |\Pr[\mathsf{Win}_{\mathsf{Ind}}] - 1/2|\} \\ &= \max\{\Pr[\mathsf{Win}_{\mathsf{Auth}}], |\Pr[\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}} \Rightarrow 1] - 1/2|\}.\end{aligned}$$

Definition 14 (Security of AKE with Replay Attacks and without State Reveal)**.** *The security experiment* $\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *(see Fig. 5) is defined like* $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *except that we disallow state reveal queries. Similarly, the advantage of an adversary* $\mathcal{A}$ *in* $\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{replay}}_{\mathsf{AKE},\mu,\ell}(\mathcal{A}) := \max\{\Pr[\mathsf{Win}_{\mathsf{Auth}}], |\Pr[\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}} \Rightarrow 1] - 1/2|\}.$$

Definition 15 (Security of AKE without Replay Attack and State Reveal)**.** *The security experiment* $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *(see Fig. 5) is defined like* $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *except that we disallow replay attacks and state reveal queries. Similarly, the advantage of an adversary* $\mathcal{A}$ *in* $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ *is defined as*

$$\mathsf{Adv}_{\mathsf{AKE},\mu,\ell}(\mathcal{A}) := \max\{\Pr[\mathsf{Win}_{\mathsf{Auth}}], |\Pr[\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}} \Rightarrow 1] - 1/2|\}.$$

Remark 4 (Perfect Forward Security and KCI Resistance). The security model of AKE supports (perfect) forward security (a.k.a. forward secrecy [Gün90]).That is, if P_i or its partner P_j has been corrupted at some moment, then the exchanged session keys computed before the corruption remain hidden from the adversary. Meanwhile, π_i^s may be corrupted before $\mathsf{Test}(i,s)$, which provides resistance to key-compromise impersonation (KCI) attacks [Kra05].

[5] Given (1) $\wedge$ (2), (3.1) indicates a successful impersonation of P_j, (3.2) suggests one instance of P_i has multiple partners, and (3.3) corresponds to a successful replay attack.

5 AKE Protocols

We construct AKE protocols $\mathsf{AKE}_{\mathsf{2msg}}$, $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ from a signature scheme SIG and a key encapsulation mechanism KEM. Additionally, we use a symmetric encryption scheme SE with key space $\mathcal{K}_{\mathsf{SE}}$ to encrypt the state in protocol $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$. Apart from that, $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ and $\mathsf{AKE}_{\mathsf{3msg}}$ are the same. The protocols are given in Figure 6.

The setup algorithm generates the public parameter $\mathsf{pp}_{\mathsf{AKE}} := (\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{KEM}})$ by running $\mathsf{SIG.Setup}$ and $\mathsf{KEM.Setup}$. The key generation algorithm inputs the public parameter and a party P_i and generates a signature key pair (vk_i, ssk_i). In $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$, it also chooses a symmetric key s_i uniformly from the key space $\mathcal{K}_{\mathsf{SE}}$. It returns the public key vk_i and the secret key (ssk_i, s_i).

The protocol is executed between two parties P_i and P_j. Each party has access to their own resources res_i and res_j which contain the corresponding secret key, the public parameter and a list PKList consisting of the public keys of all parties. Each party initializes its local variables Ψ_i, k_i and st_i with the empty string. To initiate a session in $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$, the party P_j chooses a bitstring N uniformly from $\{0,1\}^\lambda$ and sends it to P_i. The next message and the first message in protocol $\mathsf{AKE}_{\mathsf{2msg}}$ is sent by P_i. It generates an ephemeral key pair $(\hat{pk}, \hat{sk})$ by running $\mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$ and computes a signature σ_1 over the identities of P_i and P_j, the ephemeral public key and the nonce (only in $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$). When using state encryption, it also encrypts the ephemeral secret key using its symmetric key s_i and stores the ciphertext in st_i. It then sends $(\hat{pk}, \sigma_1)$ to P_j. P_j verifies the signature using vk_i and rejects if it is not valid. Otherwise, it continues the protocol by computing $(c, K) \leftarrow_{\$} \mathsf{Encap}(\hat{pk})$. It computes a signature σ_2 over the identities as well as the previous message, c and the nonce (only in $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$). P_j accepts the session key and sets k_j to K. It sends (c, σ_2) to P_i. P_i verifies the signature and rejects if it is invalid. Otherwise, it retrieves the ephemeral secret key by decrypting the state, computes the session key K from c and accepts.

Correctness. Correctness of $\mathsf{AKE}_{\mathsf{2msg}}$, $\mathsf{AKE}_{\mathsf{3msg}}$ and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ follows directly from the correctness of SIG, KEM and SE.

Theorem 1 (Security of $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ with Replay Attacks and State Reveals)**.** *For any adversary $\mathcal{A}$ against $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ with replay attacks and state reveals, there exist an MU-EUF-CMA$^{\mathsf{corr}}$ adversary $\mathcal{B}_{\mathsf{SIG}}$ against SIG, an ϵ-MU-SIM adversary $\mathcal{B}_{\mathsf{KEM}}$ against KEM and an IND-mRPA adversary $\mathcal{B}_{\mathsf{SE}}$ against SE such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{replay,\,state}}_{\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}},\mu,\ell}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{mu\text{-}sim}}_{\mathsf{KEM},\mathsf{Encap}^*,\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) + 2 \cdot \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) \\ & + 2\mu \cdot \mathsf{Adv}^{\mathsf{mrpa}}_{\mathsf{SE},\mu}(\mathcal{B}_{\mathsf{SE}}) + 2\mu\ell \cdot \epsilon + 2(\mu\ell)^2 \cdot 2^{-\gamma} + \mu\ell^2 \cdot 2^{-\lambda}\ ,\end{aligned}$$

where γ is the diversity parameter of KEM and λ is the length of the nonce N in bits. Furthermore, $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{KEM}})$, $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{SIG}})$ and $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{SE}})$.

We first give a proof sketch, then present the formal proof of Theorem 1.

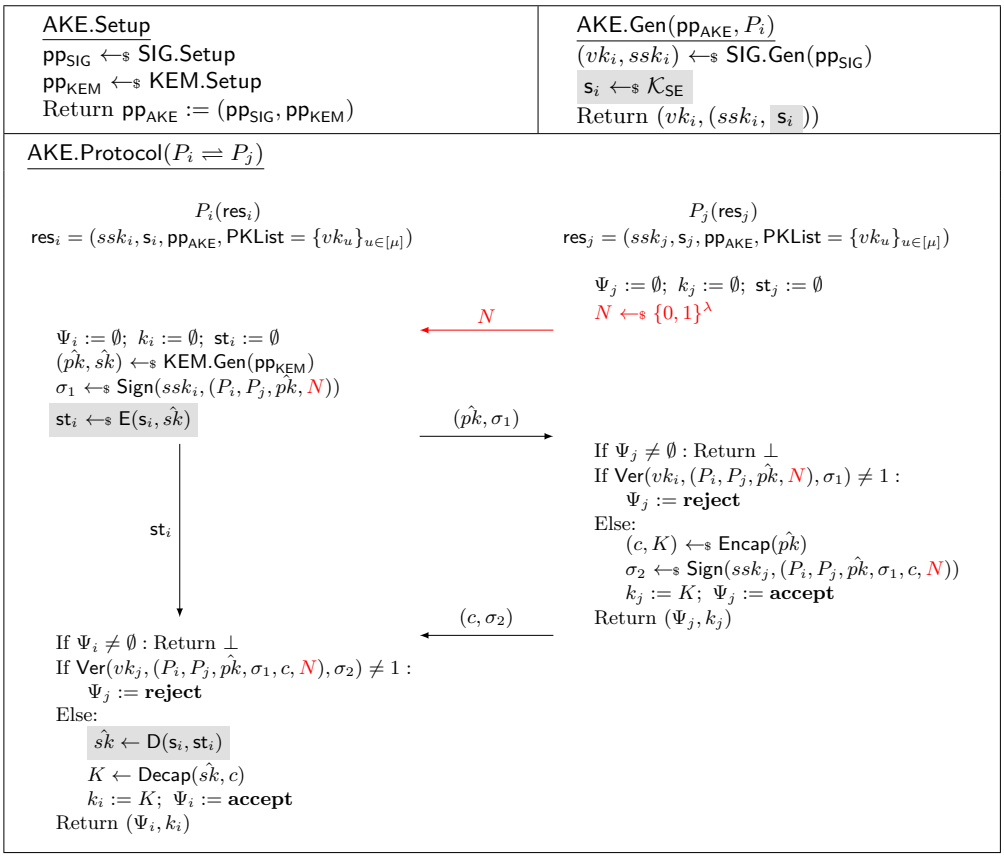

Figure 6: Generic construction of $\mathsf{AKE}_{\mathsf{2msg}}$ (without red and gray parts), $\mathsf{AKE}_{\mathsf{3msg}}$ (with red and without gray parts) and $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ (with red and gray parts) from KEM, SIG and SE. Note that the state of P_j only consists of public parts and is therefore omitted here.

Proof Sketch. The signatures in the protocol ensure that the adversary can only forward messages for those sessions that it wants to test. Thus the experiment can control all ephemeral public keys $\hat{pk}$ and ciphertexts c that are used for test queries. Due to the nonce, the adversary can also not replay a message containing a particular $\hat{pk}$. Thus, each $\hat{pk}$ is used in at most one test query.

A party will close a session when it accepts or rejects the session. Thus, the adversary can submit at most one ciphertext c' which is different from the ciphertext used in the test query. Using a session key reveal query, the adversary will only see at most one more key decapsulated with $\hat{sk}$.

To deal with state reveals, the adversary $\mathcal{A}$ can additionally obtain the state which is the encrypted $\hat{sk}$. The reduction must know $\hat{sk}$ in order to answer those queries. The simulatability property of KEM ensures that Encap and Encap* are indistinguishable,

even given $\hat{sk}$. So, we first switch from Encap to Encap^*. Now, we want to replace the session keys of tested sessions with random keys. Therefore, we have to do a hybrid argument over all users. In the η-th hybrid, we replace the test session keys for party P_η. We can show that this is unnoticeable using that the key K^* generated by Encap^* is statistically close to uniform even if the adversary gets to see another key for a ciphertext of its choice. We distinguish the following cases.

Case 1: The adversary corrupts P_η. For each session, the adversary can either reveal the session state or test this session. If the adversary reveals the state, we do not have to replace the session key. If the session is tested, the adversary does not know the state $\mathsf{E}(\mathsf{s}_\eta, \hat{sk})$ and thus we can replace the session key by ϵ-uniformity of Encap^*.

Case 2: The adversary does not corrupt P_η. In this case, we use that SE is IND-mRPA secure and replace $\hat{sk}$ in the encrypted state with a random secret key for this party. Then we can use ϵ-uniformity to replace all tested keys for that party with random keys, as the state does not contain any information about $\hat{sk}$. After that, we have to switch back the state encryption to encrypt the real secret key $\hat{sk}$, getting ready for the next hybrid.

After these changes, the Test oracle will always output a random key, independent of the bit b.

Overall, the proof loses a factor of 2μ only in the IND-mRPA security of the symmetric encryption scheme. All other parts are tight.

Proof of Theorem 1. For the proof, we will first define two further variables Sent_i^s and Recv_i^s for an oracle π_i^s. The set Sent_i^s will store outgoing messages of the oracle and the set Recv_i^s will store incoming messages, respectively. We stress that Recv_i^s will only store *valid* messages, e.g., the signature needs to be valid.

Message Consistency. For our 3-move protocol given in Figure 6, we say that an oracle π_i^s is *message-consistent* with another oracle π_j^t, denoted by $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$, if $\mathsf{Pid}_i^s := j$ and $\mathsf{Pid}_j^t := i$ and either

(1) π_i^s has sent the first message, the same nonce N is contained in Sent_i^s and Recv_j^t and the same ephemeral key $\hat{pk}$ is contained in Recv_i^s and Sent_j^t, or

(2) π_i^s has received the first message, the same nonce N and ciphertext c are contained in Recv_i^s and Sent_j^t and the same ephemeral key $\hat{pk}$ is contained in Sent_i^s and Recv_j^t.

We write $\mathsf{MsgCon}(\pi_i^s \leftrightarrow \pi_j^t)$ if $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ and $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s)$.

To prove the theorem, we now consider the sequence of games G_0-G_5. In the following, we describe the games and show that adjacent games are indistinguishable. Let Win_i denote the probability that G_i returns 1.

Game G_0: G_0 is the original experiment $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$. In addition to the original game, we add the sets Sent^s_i and Recv^s_i which is only a conceptual change. We have

$$\Pr[\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}},\mu,\ell,\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{Win}_0] \ .$$

Game G_1: In G_1, we define the event Repeat which happens if a nonce repeats for any two oracles of the same party. If Repeat happens, the game aborts (see also Figure 7). Due to the difference lemma,

$$|\Pr[\mathsf{Win}_0] - \Pr[\mathsf{Win}_1]| \leq \Pr[\mathsf{Repeat}] \ .$$

Using the birthday paradox and union bound over the number of parties, we have $\Pr[\mathsf{Repeat}] \leq \mu\ell^2 \cdot 2^{-\lambda}$, where λ is the length of the nonce in bits.

Game G_2: In G_2, we define the event $\mathsf{NoMsgCon}$ which happens if there exists some (i,s) such that π^s_i accepts, the intended partner $j := \mathsf{Pid}^s_i$ is uncorrupted when π^s_i accepts, and there does not exist $t \in [\ell]$ such that π^s_i is message-consistent with π^t_j. If event $\mathsf{NoMsgCon}$ happens, the game will abort (see also Figure 7). Due to the difference lemma,

$$|\Pr[\mathsf{Win}_1] - \Pr[\mathsf{Win}_2]| \leq \Pr[\mathsf{NoMsgCon}] \ .$$

We will prove the following lemma.

Lemma 1. *There exists an adversary $\mathcal{B}_{\mathsf{SIG}}$ against SIG such that*

$$\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] \leq \Pr[\mathsf{NoMsgCon}] \leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}).$$

Proof. If there exists an oracle π^t_j such that π^s_i is message-consistent with π^t_j, then due to correctness of KEM, π^s_i is also partnered to π^t_j. It follows that $\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] \leq \Pr[\mathsf{NoMsgCon}]$.

To prove that $\Pr[\mathsf{NoMsgCon}] \leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}})$, we construct adversary $\mathcal{B}_{\mathsf{SIG}}$ against MU-EUF-CMA$^{\mathsf{corr}}$ security of SIG. $\mathcal{B}_{\mathsf{SIG}}$ inputs the public parameter $\mathsf{pp}_{\mathsf{SIG}}$ and a list of verification keys $\{vk_i\}_{i\in[\mu]}$ and has access to a signing oracle $\mathcal{O}_{\mathrm{SIGN}}(\cdot,\cdot)$ and a corrupt oracle $\mathcal{O}_{\mathrm{CORR}}(\cdot)$. $\mathcal{B}_{\mathsf{SIG}}$ then runs $\mathsf{pp}_{\mathsf{KEM}} \leftarrow\!\!\$\ \mathsf{KEM.Setup}$ and sets $\mathsf{pp}_{\mathsf{AKE}} := (\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{KEM}})$ and $\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$. It chooses symmetric keys s_i for each user $i \in [\mu]$, initializes all variables and then runs $\mathcal{A}$ on $\mathsf{pp}_{\mathsf{AKE}}$ and PKList. If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{AKE}}$, $\mathcal{B}_{\mathsf{SIG}}$ responds as follows.

- $\mathsf{Send}(i,s,j,\mathsf{msg} = N)$: In order to get σ_1, $\mathcal{B}_{\mathsf{SIG}}$ queries its signing oracle $\mathcal{O}_{\mathrm{SIGN}}(i,(P_i, P_j, \hat{pk}, N))$.
- $\mathsf{Send}(i,s,j,\mathsf{msg} = (\hat{pk},\sigma_1))$: In order to get σ_2, $\mathcal{B}_{\mathsf{SIG}}$ queries its signing oracle $\mathcal{O}_{\mathrm{SIGN}}(i,(P_j, P_i, \hat{pk}, \sigma_1, c, N))$.
- $\mathsf{Corrupt}(i)$: $\mathcal{B}_{\mathsf{SIG}}$ queries its own oracle $\mathcal{O}_{\mathrm{CORR}}(i)$ and receives the signing key ssk_i. It returns (ssk_i, s_i) to $\mathcal{A}$.
- Queries $\mathsf{Send}(i,s,j,\top)$, $\mathsf{Send}(i,s,j,(c,\sigma_2))$, $\mathsf{RegisterCorrupt}$, $\mathsf{StateReveal}$, $\mathsf{SessionKeyReveal}$ and Test can be simulated as in the original experiment $\mathsf{Exp}^{\mathsf{replay,\,state}}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$.

During the simulation, $\mathcal{B}_{\mathsf{SIG}}$ checks if $\mathsf{NoMsgCon}$ happens. If this is the case, there exists an oracle π_i^s such that π_i^s has accepted and $j := \mathsf{Pid}_i^s$ is uncorrupted at that point in time.

Now we show that then there is a valid message-signature pair (m^*, σ^*) in Sent_i^s and Recv_i^s such that $\mathsf{Ver}(vk_j, m^*, \sigma^*) = 1$ and m^* is different from any message m signed by π_j^t for all $t \in [\ell]$. Since π_i^s is accepted, $\mathsf{Sent}_i^s \neq \varnothing$ and $\mathsf{Recv}_i^s \neq \varnothing$.

Case 1: π_i^s sent the first message. Let $\mathsf{Sent}_i^s = \{N, (c, \sigma_2)\}$ and $\mathsf{Recv}_i^s = \{(\hat{pk}, \sigma_1)\}$. We have $\mathsf{Ver}(vk_j, (P_j, P_i, \hat{pk}, N), \sigma_1) = 1$, since $\mathsf{Recv}_i^s \neq \varnothing$. For any oracle π_j^t with $\mathsf{Recv}_j^t = \{N', \cdot\}$ and $\mathsf{Sent}_j^t = \{(\hat{pk}', \sigma_1')\} \neq \varnothing$, $\mathsf{NoMsgCon}$ implies that $(\hat{pk}, N) \neq (\hat{pk}', N')$. In this case, $\mathcal{B}_{\mathsf{SIG}}$ sets $(m^*, \sigma^*) := ((P_j, P_i, \hat{pk}, N), \sigma_1)$.

Case 2: π_i^s received the first message. Let $\mathsf{Recv}_i^s = \{N, (c, \sigma_2)\}$ and $\mathsf{Sent}_i^s = \{(\hat{pk}, \sigma_1)\}$. We have $\mathsf{Ver}(vk_j, (P_i, P_j, \hat{pk}, \sigma_1, c, N), \sigma_2) = 1$, since $(c, \sigma_2) \in \mathsf{Recv}_i^s$. For any oracle π_j^t with $\mathsf{Recv}_j^t = \{(\hat{pk}', \sigma_1')\} \neq \varnothing$ and $\mathsf{Sent}_j^t = \{N', (c', \sigma_2')\} \neq \varnothing$, $\mathsf{NoMsgCon}$ implies that $(\hat{pk}, c, N) \neq (\hat{pk}', c', N')$. In this case, $\mathcal{B}_{\mathsf{SIG}}$ sets $(m^*, \sigma^*) := ((P_i, P_j, \hat{pk}, \sigma_1, c, N), \sigma_2)$.

As soon as event $\mathsf{NoMsgCon}$ happens, $\mathcal{B}_{\mathsf{SIG}}$ retrieves the message-signature (m^*, σ^*) pair as just described and outputs (j, m^*, σ^*). As P_j is uncorrupted, $\mathcal{B}_{\mathsf{SIG}}$ has not queried $\mathcal{O}_{\textsc{Corr}}(j)$ and m^* is different from all signing queries for j, which concludes the proof of Lemma 1. □

Before moving to G_3, let us bound $(1) \wedge (2) \wedge (3.2)$ and $(1) \wedge (2) \wedge (3.3)$.

Multiple Partners. Event $(1) \wedge (2) \wedge (3.2)$ happens if there exists any oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$ and has more than one partner oracle. We can show that

$$\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] \leq (\mu\ell)^2 \cdot 2^{-\gamma} \ .$$

The session key only depends on the ephemeral public key $\hat{pk}$ and the ciphertext c. In the following, we assume that there are two oracles π_j^t and $\pi_{j'}^{t'}$ such that π_i^s is partnered to both π_j^t and $\pi_{j'}^{t'}$. We distinguish two cases:

Case 1: π_i^s sent the first message. Let $\hat{pk}$ and $\hat{pk}'$ be the public keys determined by the internal randomness of π_j^t and $\pi_{j'}^{t'}$, respectively. Let r be the internal randomness of π_i^s which is used by Encap. The original keys are derived from $(c, K) \leftarrow \mathsf{Encap}(\hat{pk}; r)$ and $(c', K') \leftarrow \mathsf{Encap}(\hat{pk}'; r)$. As π_i^s is partnered to both oracles, $k_i^s = K = K'$. Due to γ-diversity of KEM, this will happen only with probability at most $2^{-\gamma}$.

Case 2: π_i^s received the first message. Let $\hat{pk}$ be the ephemeral public key determined by the internal randomness of π_i^s. Let $(c, K) \leftarrow \mathsf{Encap}(\hat{pk}; r)$ and $(c', K') \leftarrow \mathsf{Encap}(\hat{pk}; r')$, where r, r' is the internal randomness of π_j^t and $\pi_{j'}^{t'}$, respectively. As π_i^s is partnered to both oracles, this implies that $k_i^s = \mathsf{Decap}(\hat{sk}, c) = \mathsf{Decap}(\hat{sk}, c')$. By the correctness and γ-diversity of KEM, we have $k_i^s = K = K'$ which will happen with probability at most $2^{-\gamma}$.

As there are $\mu\ell$ oracles, we can upper bound the probability for event $(1) \wedge (2) \wedge (3.2)$ by $(\mu\ell)^2 \cdot 2^{-\gamma}$.

Replay Attacks. Event $(1) \wedge (2) \wedge (3.3)$ covers replay attacks and happens if there exists any oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$, is partnered to an oracle π_j^t, and there exists another oracle $\pi_{i'}^{s'}$ such that this oracle is also partnered to π_j^t. We will show

$$\begin{aligned}\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.3)] &\leq \Pr[\mathsf{Repeat}] + \Pr[\mathsf{NoMsgCon}] + \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] \\ &\leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + (\mu\ell)^2 \cdot 2^{-\gamma} + \mu\ell^2 \cdot 2^{-\lambda} \ .\end{aligned}$$

We bound $(1) \wedge (2) \wedge (3.3)$ by using previous observations. Assume that $\mathsf{NoMsgCon}$ and $(1) \wedge (2) \wedge (3.2)$ do not occur. Then for each oracle π_i^s there exists a unique oracle π_j^t such that π_i^s is partnered to and message-consistent with π_j^t. Now assume there exists another oracle $\pi_{i'}^{s'}$ that is also partnered to and message-consistent with π_j^t. Message-consistency implies that $i = i'$. Now let $s \neq s'$.

Case 1: π_i^s sent the first message. Then, the three sets Recv_i^s, $\mathsf{Recv}_{i'}^{s'}$, Sent_j^t all contain the same ephemeral public key $\hat{pk}$ and no other oracle than π_j^t has output $\hat{pk}$. Let $\mathsf{Sent}_i^s = \{N, (c, \sigma_2)\}$, then $\mathsf{Sent}_{i'}^{s'} = \{N, (c', \sigma_2')\}$ shares the same nonce N. However, this will only happen if Repeat happens.

Case 2: π_i^s received the first message. Then, the three sets Sent_i^s, $\mathsf{Sent}_{i'}^{s'}$, Recv_j^t all contain the same ephemeral public key $\hat{pk}$ and the sets Recv_i^s, $\mathsf{Recv}_{i'}^{s'}$, Sent_j^t contain the same nonce N and ciphertext c. This means that π_j^t is partnered to both π_i^s and $\pi_{i'}^{s'}$ and thus contradicts to the fact that each oracle has a unique partner.

At this point note that

$$\begin{aligned}\Pr[\mathsf{Win}_{\mathsf{Auth}}] &= \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge ((3.1) \vee (3.2) \vee (3.3))] \\ &\leq \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] + \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] + \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.3)] \\ &\leq 2 \cdot \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + 2(\mu\ell)^2 \cdot 2^{-\gamma} + \mu\ell^2 \cdot 2^{-\lambda} \ .\end{aligned}$$

The analysis of $\mathsf{Win}_{\mathsf{Auth}}$ will be helpful for the next game hop. The following games G_3-G_5 are also given in Figure 7.

Game G_3: In G_3, we check the partnership $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ by message-consistency $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ if $\Psi_i^s = \textbf{accept}$ and $\mathsf{Aflag}_i^s = \textbf{false}$. We claim that

$$|\Pr[\mathsf{Win}_2] - \Pr[\mathsf{Win}_3]| \leq \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] \leq (\mu\ell)^2 \cdot 2^{-\gamma} \ .$$

Recall that if $\mathsf{NoMsgCon}$ does not happen, we know that each oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$ is partnered to and message-consistent with an oracle π_j^t. If any such oracle π_i^s has a unique partner, then G_2 is identical to G_3. On the other hand, the probability that there exists an oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$

and has multiple partners is $\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)]$, which is bounded by $(\mu\ell)^2 \cdot 2^{-\gamma}$. Thus, the claims follows by the difference lemma.

Game G_4: In G_4, we use the Encap^* algorithm (instead of Encap) whenever $\mathcal{A}$ issues a Send query (i, s, j, msg) with a second protocol message $\mathsf{msg} = (\hat{pk}, \sigma_1)$ and the intended partner P_j is not corrupted. We construct adversary $\mathcal{B}_{\mathsf{KEM}}$ against indistinguishability of Encap and Encap^*.

$\mathcal{B}_{\mathsf{KEM}}$ inputs the public parameter $\mathsf{pp}_{\mathsf{KEM}}$ and $\{pk_n, sk_n, c_n, K_n\}_{n\in[\mu\ell]}$, where (c_n, K_n) are either computed by $\mathsf{Encap}(pk_n)$ or by $\mathsf{Encap}^*(sk_n)$. $\mathcal{B}_{\mathsf{KEM}}$ generates the public parameter for SIG and signature key pairs (vk_i, ssk_i) for $i \in [\mu]$, as well as symmetric keys s_i. It sets $\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$, initializes all variables, chooses $b \leftarrow\!\!\$\ \{0, 1\}$ and runs $\mathcal{A}$. If $\mathcal{A}$ makes a query to $\mathcal{O}_{\mathsf{AKE}}$, $\mathcal{B}_{\mathsf{KEM}}$ simulates the response as follows:

- $\mathsf{Send}(i, s, j, \mathsf{msg} = N)$: $\mathcal{B}_{\mathsf{KEM}}$ uses the key pair with index $(i-1)\mu + s$ as ephemeral key pair, i.e. $(\hat{pk}, \hat{sk}) := (pk_{(i-1)\mu+s}, sk_{(i-1)\mu+s})$.
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (\hat{pk}, \sigma_1))$: If P_j is uncorrupted, then due to the fact that event $\mathsf{NoMsgCon}$ does not happen, there exists a unique oracle π_j^t such that $\hat{pk}$ was output by π_j^t. Furthermore, $n = (j-1)\mu + t$ is the index of that public key. Then $\mathcal{B}_{\mathsf{KEM}}$ uses (c_n, K_n) as ciphertext and key. If P_j is corrupted, $\mathcal{B}_{\mathsf{KEM}}$ runs $\mathsf{Encap}(\hat{pk})$ itself to compute (c, K).
- Queries $\mathsf{Send}(i, s, j, \top)$, $\mathsf{Send}(i, s, j, (c, \sigma_2))$, $\mathsf{Corrupt}$, $\mathsf{RegisterCorrupt}$, $\mathsf{StateReveal}$, Test and $\mathsf{SessionKeyReveal}$ can be simulated as in G_3 and G_4, except for the partnership check $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$. Recall that $\mathcal{B}_{\mathsf{KEM}}$ needs to compute $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ in $\mathsf{Send}(i, s, j, \mathsf{msg})$ to set $FirstAcc$, in $\mathsf{Test}(i, s)$ and $\mathsf{StateReveal}(i, s)$ to detect **TA7**, and compute $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$ in $\mathsf{Test}(i, s)$ and $\mathsf{SessionKeyReveal}(i, s)$ to detect **TA4** and **TA5**.
  - For the set of $FirstAcc$ in $\mathsf{Send}(i, s, j, \mathsf{msg})$ and for the detection of **TA7** in $\mathsf{Test}(i, s)$ and $\mathsf{StateReveal}(i, s)$, $\mathcal{B}_{\mathsf{KEM}}$ simulates $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ with $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$. This simulation is perfect since $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ is involved only when $\mathsf{Aflag}_i^s = \mathbf{false}$.
  - For the detection of **TA4** and **TA5** in $\mathsf{Test}(i, s)$ and $\mathsf{SessionKeyReveal}(i, s)$, $\mathcal{B}_{\mathsf{KEM}}$ simulates $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$ as follows. $\mathcal{B}_{\mathsf{KEM}}$ first checks $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ with $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$. (Again, since $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$ is involved only when $\mathsf{Aflag}_i^s = \mathbf{false}$, $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) = \mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$.) If $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t) = 0$, $\mathcal{B}_{\mathsf{KEM}}$ outputs 0 directly for $\mathsf{Partner}(\pi_i^s \leftrightarrow \pi_j^t)$. Otherwise, π_i^s is partner to and message-consistent with π_j^t, hence k_i^s must be equal to the original key between π_i^s and π_j^t, so $\mathcal{B}_{\mathsf{KEM}}$ can further check $\mathsf{Partner}(\pi_i^s \rightarrow \pi_j^t)$ by simply testing whether $k_j^t = k_i^s$. This simulation is perfect as well.

Finally, $\mathcal{A}$ outputs b^*. If $b = b^*$, $\mathcal{B}_{\mathsf{KEM}}$ outputs 1. Otherwise, it outputs 0.

We want to elaborate in more detail on why a tuple (pk_n, sk_n, c_n, K_n) is used at most once. As $\mathsf{NoMsgCon}$ does not happen, there exists a partner for each oracle that has accepted when the intended partner was not corrupted. Thus, for each query $\mathsf{Send}(i, s, j, (\hat{pk}, \sigma_1))$, where P_j is uncorrupted, there exists a partner oracle π_j^t that has sent $(\hat{pk}, \cdot)$. Finally, as Repeat does not happen, $\mathcal{A}$ cannot replay $(\hat{pk}, \sigma_1)$ to another oracle $\pi_i^{s'}$ because σ_1 includes the identities and the nonce N.

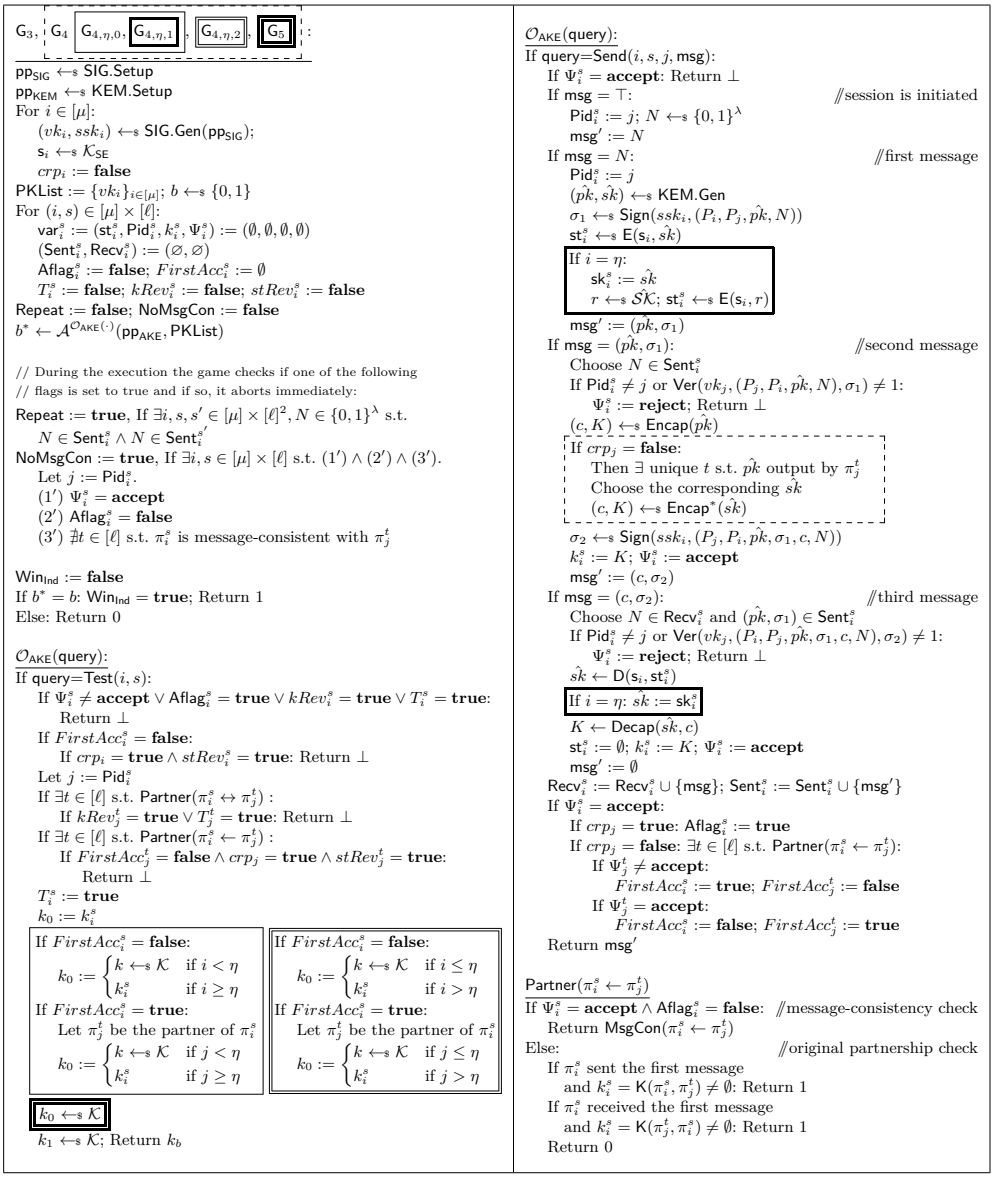

Figure 7: Games G_3-G_5 for the proof of Theorem 1. Queries to $\mathcal{O}_{\mathsf{AKE}}$ where query$\in$ $\{\mathsf{Corrupt}, \mathsf{RegisterCorrupt}, \mathsf{SessionKeyReveal}, \mathsf{StateReveal}\}$ are defined as in the original game in Figure 5.

If $\mathcal{B}_{\mathsf{KEM}}$'s input (c_n, K_n) is computed using the original Encap algorithm, then $\mathcal{B}_{\mathsf{KEM}}$ perfectly simulates G_3. Otherwise, if (c_n, K_n) is computed using the Encap^* algorithm, then $\mathcal{B}_{\mathsf{KEM}}$ perfectly simulates G_4. Hence,

$$|\Pr[\mathsf{Win}_3] - \Pr[\mathsf{Win}_4]| \leq \mathsf{Adv}^{\mathsf{mu\text{-}sim}}_{\mathsf{KEM},\mathsf{Encap}^*,\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) \ .$$

Game $\mathsf{G}_{4,\eta,0}$, $\eta \in \{1, ..., \mu+1\}$: From $\mathsf{G}_{4,1,0}$ to $\mathsf{G}_{4,\mu+1,0}$, we will use hybrid arguments to replace the test keys k_0 with random keys for all oracles π_i^s. Here, we have to take into account whether the oracle sent the first message or whether it received the first message. We will consider one user after another and in each step, we will replace the session key in test queries where that user's oracle has received the first message. At the same time, we replace the session keys in test queries where that user's oracle is a partner oracle that has received the first message. In particular, in game $\mathsf{G}_{4,\eta,0}$, when $\mathcal{A}$ queries $\mathsf{Test}(i, s)$, instead of setting k_0 to the real session key k_i^s, we choose a random key if

(1) π_i^s has received the first message and $i < \eta$, or
(2) π_i^s has sent the first message, π_j^t is the partner oracle and $j < \eta$.

Clearly, $\mathsf{G}_{4,1,0}$ is identical to G_4 and $\mathsf{G}_{4,\mu+1,0}$ is identical to G_5.

Lemma 2. *Let Encap^* be the additional algorithm associated to a KEM and let the key encapsulated by Encap^* be ϵ-uniform. Then for $\eta \in \{1, ..., \mu\}$,*

$$|\Pr[\mathsf{Win}_{4,\eta,0}] - \Pr[\mathsf{Win}_{4,\eta+1,0}]| \leq 2 \cdot \mathsf{Adv}^{\mathsf{mrpa}}_{\mathsf{SE},\ell}(\mathcal{B}_{\mathsf{SE}}) + 2\ell \cdot \epsilon \ .$$

Proof. We will consider two cases: (1) The adversary corrupts P_η and (2) the adversary does not corrupt P_η. We have

$$\begin{aligned}|\Pr[\mathsf{Win}_{4,\eta,0}] - \Pr[\mathsf{Win}_{4,\eta+1,0}]| \leq\ & |\Pr[\mathsf{Win}_{4,\eta,0} \wedge crp_\eta] - \Pr[\mathsf{Win}_{4,\eta+1,0} \wedge crp_\eta]| \\ & + |\Pr[\mathsf{Win}_{4,\eta,0} \wedge \neg crp_\eta] - \Pr[\mathsf{Win}_{4,\eta+1,0} \wedge \neg crp_\eta]| \ .\end{aligned}$$

First, we consider the case that P_η is corrupted. Let π_η^s be any oracle of P_η. If $\mathcal{A}$ does not issue a test query on any π_η^s directly or where π_η^s is the partner, then $\Pr[\mathsf{Win}_{4,\eta,0} \wedge crp_\eta] = \Pr[\mathsf{Win}_{4,\eta+1,0} \wedge crp_\eta]$.

Otherwise, we have to consider the following two cases.

Case 1: $\mathcal{A}$ asks $\mathsf{Test}(i, s)$ for $i = \eta$ and π_η^s received the first message ($FirstAcc_\eta^s =$ **false**). We know that the partner oracle π_j^t received the ephemeral public key $\hat{pk}$ output by π_η^s and has sent a ciphertext c computed by Encap^*. Then, as P_η is corrupted and $\mathcal{A}$ queries $\mathsf{Test}(\eta, s)$, $\mathcal{A}$ is disallowed to ask $\mathsf{StateReveal}(\eta, s)$ (**TA6**). So the information of the ephemeral secret key $\hat{sk}$ leaked to $\mathcal{A}$ is limited in $\hat{pk}$. By the ϵ-uniformity of Encap^*, we can replace the corresponding session key in $\mathsf{Test}(\eta, s)$ with a random key. Note that in this case, the π_j^t is message-consistent with π_η^s (i.e., $\mathsf{Partner}(\pi_\eta^s \leftrightarrow \pi_j^t)$) and $\mathcal{A}$ can neither test nor reveal the key of π_j^t (**TA4**, **TA5**).

Case 2: $\mathcal{A}$ asks $\mathsf{Test}(i, s)$, where π_i^s sent the first message ($FirstAcc_i^s =$ **true**) and is partnered to π_η^t. We know that the ephemeral public key $\hat{pk}$ received by π_i^s was sent by π_η^t as P_η has to be uncorrupted when π_i^s accepts. The ciphertext c sent by π_i^s was computed by Encap^*. As we consider that P_η is corrupted later and $\mathcal{A}$ queries $\mathsf{Test}(i, s)$, $\mathcal{A}$ is disallowed to ask $\mathsf{StateReveal}(\eta, t)$ (**TA7**).

However, P_i may be corrupted and $\mathcal{A}$ can create a new ciphertext $c' \neq c$, sent it to π_η^t. In this case π_η^t is not message-consistent with π_i^s. If $\mathcal{A}$ reveals the session key of π_η^t, it will get $\mathsf{Decap}(\hat{sk}, c')$. Overall, the information of the ephemeral secret key $\hat{sk}$ leaked to $\mathcal{A}$ is limited in $\hat{pk}$ and $\mathsf{Decap}(\hat{sk}, c')$. By ϵ-uniformity of Encap^*, we can still replace the session key of π_i^s with a random key.

As one party has at most ℓ sessions, union bound yields

$$|\Pr[\mathsf{Win}_{4,\eta,0} \wedge crp_\eta] - \Pr[\mathsf{Win}_{4,\eta+1,0} \wedge crp_\eta]| \leq \ell \cdot \epsilon \ .$$

Now we will look at the case that P_η is not corrupted and we introduce two further intermediate games $\mathsf{G}_{4,\eta,1}$ and $\mathsf{G}_{4,\eta,2}$.

Game $\mathsf{G}_{4,\eta,1}$, $\eta \in \{1, ..., \mu\}$: On a query $\mathsf{Send}(i, s, j, N)$, where $i = \eta$, we do not encrypt the ephemeral secret key $\hat{sk}$ in the state, but a random secret key $r \leftarrow_\$ \hat{\mathcal{SK}}$. We store the real secret key in an additional variable sk_η^s such that we can later access it for decapsulation. The rest remains unchanged.

We now construct an IND-mRPA adversary $\mathcal{B}_{\mathsf{SE},\eta}$ against the symmetric encryption scheme SE with message space $\hat{\mathcal{SK}}$. $\mathcal{B}_{\mathsf{SE},\eta}$ inputs ℓ message-ciphertext pairs $\{(\hat{sk}_n, c_n)\}_{n\in[\ell]}$ for ℓ random messages $\hat{sk}_n \leftarrow_\$ \hat{\mathcal{SK}}$, where c_n is either an encryption of $\hat{sk}_n$ or that of a random message $r_n \leftarrow_\$ \hat{\mathcal{SK}}$. $\mathcal{B}_{\mathsf{SE},\eta}$ generates the public parameter and signature key pairs (vk_i, ssk_i) for $i \in [\mu]$. For all $i \neq \eta$, it also generates symmetric keys s_i. It sets $\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$, initializes all variables, chooses $b \leftarrow_\$ \{0, 1\}$ and then runs $\mathcal{A}$. If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{AKE}}$, $\mathcal{B}_{\mathsf{SE},\eta}$ responds as follows:

- $\mathsf{Send}(i, s, j, \mathsf{msg} = N)$: If $i = \eta$, $\mathcal{B}_{\mathsf{SE},\eta}$ computes $\hat{pk} := \mathsf{KEM.PK}(\hat{sk}_s)$ from the s-th message. It sets $\mathsf{sk}_\eta^s := \hat{sk}_s$ and the state variable $\mathsf{st}_\eta^s := c_s$.
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (c, \sigma_2))$: If $i = \eta$, $\mathcal{B}_{\mathsf{SE},\eta}$ chooses $\hat{sk}$ from sk_η^s instead of decrypting the state.
- $\mathsf{Corrupt}(i)$: If $i = \eta$, $\mathcal{B}_{\mathsf{SE},\eta}$ aborts.
- Queries $\mathsf{Send}(i, s, j, \top)$, $\mathsf{Send}(i, s, j, (\hat{pk}, \sigma_1))$, $\mathsf{RegisterCorrupt}$, $\mathsf{StateReveal}$, $\mathsf{SessionKeyReveal}$ and Test can be simulated as in $\mathsf{G}_{4,\eta,0}$.

Finally, $\mathcal{A}$ outputs b^*. If $b = b^*$ and $\mathcal{B}_{\mathsf{SE},\eta}$ does not abort, $\mathcal{B}_{\mathsf{SE},\eta}$ outputs 1. Otherwise, it outputs 0. If the input ciphertexts are encryptions of the messages $\hat{sk}_1, ..., \hat{sk}_\ell$ and $\mathcal{B}_{\mathsf{SE},\eta}$ does not abort, it perfectly simulates $\mathsf{G}_{4,\eta,0} \wedge \neg crp_\eta$. If the input ciphertexts are encryptions of random messages and $\mathcal{B}_{\mathsf{SE},\eta}$ does not abort, it perfectly simulates $\mathsf{G}_{4,\eta,1} \wedge \neg crp_\eta$. Thus,

$$|\Pr[\mathsf{Win}_{4,\eta,0} \wedge \neg crp_\eta] - \Pr[\mathsf{Win}_{4,\eta,1} \wedge \neg crp_\eta]| \leq \mathsf{Adv}^{\mathsf{mrpa}}_{\mathsf{SE},\ell}(\mathcal{B}_{\mathsf{SE},\eta}) \ .$$

Game $\mathsf{G}_{4,\eta,2}$, $\eta \in \{1, ..., \mu\}$: In game $\mathsf{G}_{4,\eta,2}$, we switch all session keys output by Test where oracle π_η^s received the first message to random. Also, we switch all session keys output by Test where oracle π_η^s is the partner that has sent the first message to random. In particular, when $\mathcal{A}$ queries $\mathsf{Test}(i, s)$, instead of setting k_0 to the real session key k_i^s, we choose a random key if

(1) π_i^s has received the first message and $i \leq \eta$, or
(2) π_i^s has sent the first message, π_j^t is the partner oracle and $j \leq \eta$.

Similar to the case where P_η is corrupted, we will argue that the difference between the two games is bounded by the ϵ-uniformity of Encap^*. Again, we consider the two cases where we deviate from the previous game.

Case 1: $\mathcal{A}$ asks $\mathsf{Test}(i,s)$ for $i=\eta$ and π_η^s received the first message ($FirstAcc_\eta^s = \mathbf{false}$). We know that the partner oracle π_j^t received the ephemeral public key $\hat{pk}$ output by π_η^s and has sent a ciphertext c computed by Encap^*. $\mathcal{A}$ may query $\mathsf{StateReveal}(\eta,s)$, but will receive only an encryption of a random secret key. So the information of the ephemeral secret key $\hat{sk}$ leaked to $\mathcal{A}$ is limited in $\hat{pk}$. If $\mathcal{A}$ queries $\mathsf{Test}(\eta,s)$, $\mathcal{A}$ can neither test nor reveal the key of π_j^t as π_j^t is also partnered to π_η^s. Due to ϵ-uniformity of Encap^*, we can replace the session key of π_η^s with a random key.

Case 2: $\mathcal{A}$ asks $\mathsf{Test}(i,s)$, where π_i^s sent the first message ($FirstAcc_i^s = \mathbf{true}$) and is partnered to π_η^t. We know that the ephemeral public key $\hat{pk}$ received by π_i^s was sent by the partner oracle π_η^t as P_η is uncorrupted. The ciphertext c sent by π_i^s was computed by Encap^*. $\mathcal{A}$ may reveal the state of π_η^t, but will receive only an encryption of a random secret key. $\mathcal{A}$ may corrupt P_i, create a new ciphertext $c' \neq c$ and sent it to π_η^t. In this case, if π_η^t is not message-consistent with π_i^s, $\mathcal{A}$ can reveal the session key of π_η^t and receives $\mathsf{Decap}(\hat{sk},c')$. Overall, the information of the ephemeral secret key $\hat{sk}$ leaked to $\mathcal{A}$ is limited in $\hat{pk}$ and $\mathsf{Decap}(\hat{sk},c')$. Thus, we can still replace the session key of π_i^s with a random key due to ϵ-uniformity of Encap^*.

As there are at most ℓ test sessions for one party, we have

$$|\Pr[\mathsf{Win}_{4,\eta,1} \wedge \neg crp_\eta] - \Pr[\mathsf{Win}_{4,\eta,2} \wedge \neg crp_\eta]| \leq \ell \cdot \epsilon \ .$$

Now, we can switch back the encryption of a random ephemeral secret key to the real ephemeral key. Note that this is $\mathsf{G}_{4,\eta+1,0}$. We can construct an IND-mRPA adversary $\mathcal{B}'_{\mathsf{SE},\eta}$ against SE such that

$$|\Pr[\mathsf{Win}_{4,\eta,2} \wedge \neg crp_\eta] - \Pr[\mathsf{Win}_{4,\eta+1,0} \wedge \neg crp_\eta]| \leq \mathsf{Adv}^{\mathsf{mrpa}}_{\mathsf{SE},\ell}(\mathcal{B}'_{\mathsf{SE},\eta}) \ ,$$

where $\mathcal{B}'_{\mathsf{SE},\eta}$ deviates from $\mathcal{B}_{\mathsf{SE},\eta}$ only in the simulation of the Test oracle as introduced in game $\mathsf{G}_{4,\eta,2}$.

Lemma 2 now follows from collecting the probabilities and folding adversaries $\mathcal{B}_{\mathsf{SE},\eta}$ and $\mathcal{B}'_{\mathsf{SE},\eta}$ into a single adversary $\mathcal{B}_{\mathsf{SE}}$. □

Game G_5: Finally, game G_5 is identical to $\mathsf{G}_{4,\mu+1,0}$. In this game, the Test oracle always outputs a random key, independent of the bit b. Hence,

$$\Pr[\mathsf{Win}_5] = \frac{1}{2} \, ,$$

which concludes the proof of Theorem 1. □

Theorem 2 (Security of $\mathsf{AKE}_{\mathsf{3msg}}$ with Replay Attacks and without State Reveals)**.** *For any adversary $\mathcal{A}$ against $\mathsf{AKE}_{\mathsf{3msg}}$ with replay attacks and without state reveals, there*

exist an MU-EUF-CMA$^{\mathsf{corr}}$ adversary $\mathcal{B}_{\mathsf{SIG}}$ against SIG *and an MUSC-otCCA adversary $\mathcal{B}_{\mathsf{KEM}}$ against* KEM *such that*

$$\mathsf{Adv}^{\mathsf{replay}}_{\mathsf{AKE}_{\mathsf{3msg}},\mu,\ell}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{musc\text{-}otcca}}_{\mathsf{KEM},\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) + 2 \cdot \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + 2(\mu\ell)^2 \cdot 2^{-\gamma} + \mu\ell^2 \cdot 2^{-\lambda} \ ,$$

where γ is the diversity parameter of KEM *and λ is the length of the nonce N in bits. Furthermore, $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{KEM}})$ and $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{SIG}})$.*

We first give a proof sketch, then present the formal proof of Theorem 2.

Proof Sketch. This proof is very similar to the proof of Theorem 1. The signatures and nonce in the protocol ensure that the adversary can only forward messages for test sessions and cannot replay a particular ephemeral public key $\hat{pk}$.

As we do not consider state reveals, the reduction does not have to know the corresponding secret key $\hat{sk}$. Instead, we can use a weaker security notion for KEM, which allows for one challenge query and one decapsulation query for each $\hat{pk}$. Also, there is no need for a hybrid argument and we can output a random key for sessions which will be tested as well as for sessions that will be revealed, using MUSC-otCCA security of KEM.

Proof of Theorem 2. The proof is very similar to that of Theorem 1. We consider a sequence of games G_0-G_3, which are the same as in the proof of Theorem 1, only that we start with G_0 as the $\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE}_{\mathsf{3msg}},\mu,\ell,\mathcal{A}}$ experiment. Note that the game changes from G_0-G_3 in Theorem 1 do not involve state reveals. Thus by a similar analysis, we establish

$$|\Pr[\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE}_{\mathsf{3msg}},\mu,\ell,\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{Win}_3]| \leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + (\mu\ell)^2 \cdot 2^{-\gamma} + \mu\ell^2 \cdot 2^{-\lambda} \ ,$$

and

$$\Pr[\mathsf{Win}_{\mathsf{Auth}}] \leq 2 \cdot \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + 2(\mu\ell)^2 \cdot 2^{-\gamma} + \mu\ell^2 \cdot 2^{-\lambda} \ .$$

Recall that in G_3, the game defines partnering using message-consistency for all oracles π_i^s that have accepted with $\mathsf{Aflag}_i^s = \textbf{false}$. In order to bound $\Pr[\mathsf{Win}_3]$, we construct an adversary $\mathcal{B}_{\mathsf{KEM}}$ against MUSC-otCCA security of KEM (see Figure 8). We will show that

$$\left|\Pr[\mathsf{Win}_3] - \tfrac{1}{2}\right| \leq 2 \cdot \mathsf{Adv}^{\mathsf{musc\text{-}otcca}}_{\mathsf{KEM},\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) \ .$$

The idea is that we do not only replace the session key of an oracle when it is tested, but we replace the session keys of all oracles that are possibly tested. In particular, these are sessions where the intended partner is uncorrupted when the oracle accepts. These oracles can then either be tested or revealed. However, as we do not consider state reveals, the adversary will never see the ephemeral secret key and thus we can also output a random key for a SessionKeyReveal query.

Let β be the random bit of $\mathcal{B}_{\mathsf{KEM}}$'s challenger. $\mathcal{B}_{\mathsf{KEM}}$ inputs the public parameter $\mathsf{pp}_{\mathsf{KEM}}$ and $\{pk_n\}_{n\in[\mu\ell]}$. $\mathcal{B}_{\mathsf{KEM}}$ generates the public parameter for SIG and signature key pairs (vk_i, ssk_i) for $i \in [\mu]$ and sets $\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$. It initializes all variables, chooses a random challenge bit $b \leftarrow_{\$} \{0,1\}$ and runs $\mathcal{A}$. If $\mathcal{A}$ makes a query to $\mathcal{O}_{\mathsf{AKE}}$, $\mathcal{B}_{\mathsf{KEM}}$ simulates the response as follows:

- $\mathsf{Send}(i, s, j, \mathsf{msg} = N)$: $\mathcal{B}_{\mathsf{KEM}}$ uses the public key with index $(i-1)\mu+s$ as ephemeral public key, i.e. $\hat{pk} := pk_{(i-1)\mu+s}$.
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (\hat{pk}, \sigma_1))$: If P_j is uncorrupted, then due to the fact that $\mathsf{NoMsgCon}$ does not happen, there exists a unique oracle π_j^t such that $\hat{pk}$ was output by π_j^t. Furthermore, $n = (j-1)\mu + t$ is the index of that public key. Then $\mathcal{B}_{\mathsf{KEM}}$ queries $\mathcal{O}^{\beta}_{\textsc{Encap}}(n)$, receives a ciphertext and key (c, K_β) and sets $k_i^s := K_\beta$. If P_j is corrupted, $\mathcal{B}_{\mathsf{KEM}}$ runs $\mathsf{Encap}(\hat{pk})$ itself to compute (c, K). It also computes a signature σ_2 as the protocol specifies and outputs (c, σ_2).
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (c, \sigma_2))$: Let $n = (i-1)\mu + s$. Then, π_i^s sent $\hat{pk}_n$. If there exists an oracle π_j^t that has received $\hat{pk}_n$ and has sent c, then $\mathcal{B}_{\mathsf{KEM}}$ sets $k_i^s := k_j^t$. Otherwise, $\mathcal{B}_{\mathsf{KEM}}$ queries $\mathcal{O}_{\textsc{Decap}}(n, c)$, receives K and sets $k_i^s := K$.
- $\mathsf{Test}(i, s)$: After ruling out trivial attacks **TA1**, **TA2** and **TA3**, $\mathcal{B}_{\mathsf{KEM}}$ checks for trivial attacks **TA4** and **TA5** using message-consistency check $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ and tests if $k_j^t = k_i^s$. If it does not output $\perp$, $\mathcal{B}_{\mathsf{KEM}}$ sets $k_0 = k_i^s$ and $k_1 \leftarrow_{\$} \mathcal{K}$ and outputs k_b.
- $\mathsf{SessionKeyReveal}(i, s)$: After ruling out trivial attack **TA2**, $\mathcal{B}_{\mathsf{KEM}}$ checks for trivial attack **TA4** by checking if there exists an oracle π_j^t such that π_j^t is tested and $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s)$. If further $k_j^t = k_i^s$, $\mathcal{B}_{\mathsf{KEM}}$ returns $\perp$. Otherwise, it outputs k_i^s.
- Queries $\mathsf{Send}(i, s, j, \top)$, $\mathsf{Corrupt}$ and $\mathsf{RegisterCorrupt}$ can be simulated as in G_3.

Finally, $\mathcal{A}$ outputs b^* and $\mathcal{B}_{\mathsf{KEM}}$ outputs $\beta^* := 0$ if $b^* = b$ and $\beta^* := 1$ otherwise.

As events $\mathsf{NoMsgCon}$ and Repeat do not happen, $\mathcal{B}_{\mathsf{KEM}}$ queries $\mathcal{O}^{\beta}_{\textsc{Encap}}$ only once for each $\hat{pk}$, equivalently to the proof of Theorem 1. Also, $\mathcal{O}_{\textsc{Decap}}$ is queried at most once, as each ephemeral public key is only output by one oracle which accepts the session key after calling $\mathcal{O}_{\textsc{Decap}}$. In the following, we will argue that $\mathcal{B}_{\mathsf{KEM}}$ perfectly simulates G_3 if $\beta = 0$, and that $\mathcal{A}$'s view is independent of b if $\beta = 1$.

Case $\beta = 0$**:** For each query to $\mathcal{O}^0_{\textsc{Encap}}$, $\mathcal{B}_{\mathsf{KEM}}$ receives the real key. When $\mathcal{A}$ queries $\mathsf{Test}(i, s)$, $\mathcal{B}_{\mathsf{KEM}}$ needs to check partnering to avoid **TA4** and **TA5**. We know that $\mathsf{Aflag}_i^s = \textbf{false}$ because otherwise $\mathcal{B}_{\mathsf{KEM}}$ would have returned $\perp$. Thus, $\mathcal{B}_{\mathsf{KEM}}$ checks $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ as in G_3, but instead of checking $\mathsf{Partner}(\pi_j^t \leftarrow \pi_i^s)$, it checks whether $k_j^t = k_i^s$. As $\beta = 0$, k_i^s is the real session key and original key between π_i^s and π_j^t. Thus, $\mathsf{Partner}(\pi_j^t \leftarrow \pi_i^s)$ can be efficiently checked by testing if $k_j^t = k_i^s$. $\mathcal{B}_{\mathsf{KEM}}$ simulates Test queries as in G_3. When $\mathcal{A}$ queries $\mathsf{SessionKeyReveal}(i, s)$, $\mathcal{B}_{\mathsf{KEM}}$ also needs to check partnering to avoid **TA4**. Therefore, for each oracle π_j^t that is tested and thus $\mathsf{Aflag}_j^t = \textbf{false}$, $\mathcal{B}_{\mathsf{KEM}}$ checks if π_j^t is partnered to π_i^s by $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s)$ as in G_3. Instead of checking $\mathsf{Partner}(\pi_i^s \leftarrow \pi_t^j)$, it checks whether $k_j^t = k_i^s$. We know that k_j^t is the real session key, thus $\mathcal{B}_{\mathsf{KEM}}$ simulates $\mathsf{SessionKeyReveal}$ queries as in G_3. Consequently, $\mathcal{B}_{\mathsf{KEM}}$ simulates G_3 perfectly for $\mathcal{A}$ in this case, and $\Pr[\beta^* = \beta \mid \beta = 0] = \Pr[\mathsf{Win}_3]$.

Case $\beta = 1$**:** For each query to $\mathcal{O}^1_{\textsc{Encap}}$, $\mathcal{B}_{\mathsf{KEM}}$ receives a random key. When $\mathcal{A}$ queries $\mathsf{Test}(i, s)$, $\mathcal{B}_{\mathsf{KEM}}$ checks $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ and if $k_j^t = k_i^s$. As $\beta = 1$, $k_0 = k_i^s$ and k_1 are both random keys. Thus, $\mathcal{B}_{\mathsf{KEM}}$'s output is independent of b. When $\mathcal{A}$ queries $\mathsf{SessionKeyReveal}(i, s)$, $\mathcal{B}_{\mathsf{KEM}}$ checks for each oracle π_j^t that is tested and thus $\mathsf{Aflag}_j^t = \textbf{false}$, if π_j^t is partnered to π_i^s by $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s)$. If $k_j^t \neq k_i^s$,

$\underline{\mathcal{B}_{\mathsf{KEM}}^{\mathcal{O}^{\beta}_{\text{ENCAP}}(\cdot),\mathcal{O}_{\text{DECAP}}(\cdot,\cdot)}(\mathsf{pp}_{\mathsf{KEM}}, pk_1, ..., pk_{\mu\ell})}$:
$\mathsf{pp}_{\mathsf{SIG}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}$
For $i \in [\mu]$:
  $(vk_i, ssk_i) \leftarrow\!\!\$\ \mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$;
  $crp_i := \mathbf{false}$
$\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$; $b \leftarrow\!\!\$\ \{0,1\}$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{Pid}_i^s, k_i^s, \Psi_i^s) := (\emptyset, \emptyset, \emptyset)$
  $(\mathsf{Sent}_i^s, \mathsf{Recv}_i^s) := (\varnothing, \varnothing)$
  $\mathsf{Aflag}_i^s := \mathbf{false}$; $T_i^s := \mathbf{false}$; $kRev_i^s := \mathbf{false}$
$\mathsf{Repeat} := \mathbf{false}$; $\mathsf{NoMsgCon} := \mathbf{false}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AKE}}(\cdot)}(\mathsf{pp}_{\mathsf{AKE}}, \mathsf{PKList})$
If $b^* = b$: Return $\beta^* := 0$
Else: Return $\beta^* := 1$

// During the execution $\mathcal{B}_{\mathsf{KEM}}$ checks if one of the following
// flags is set to true and if so, it aborts immediately:
$\mathsf{Repeat} := \mathbf{true}$, If $\exists i, s, s' \in [\mu] \times [\ell]^2, N \in \{0,1\}^\lambda$ s.t.
  $N \in \mathsf{Sent}_i^s \wedge N \in \mathsf{Sent}_i^{s'}$
$\mathsf{NoMsgCon} := \mathbf{true}$, If $\exists i, s \in [\mu] \times [\ell]$ s.t. $(1') \wedge (2') \wedge (3')$.
  Let $j := \mathsf{Pid}_i^s$.
  $(1')$ $\Psi_i^s = \mathbf{accept}$
  $(2')$ $\mathsf{Aflag}_i^s = \mathbf{false}$
  $(3')$ $\nexists t \in [\ell]$ s.t. $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$

$\underline{\mathcal{O}_{\mathsf{AKE}}(\mathsf{query})}$:
If query=$\mathsf{Test}(i,s)$:
  If $\Psi_i^s \neq \mathbf{accept} \vee \mathsf{Aflag}_i^s = \mathbf{true} \vee kRev_i^s = \mathbf{true}$
    $\vee\ T_i^s = \mathbf{true}$:
      Return $\perp$
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [\ell]$ s.t. $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t) \wedge k_j^t = k_i^s$:
    If $kRev_j^t = \mathbf{true} \vee T_j^t = \mathbf{true}$: Return $\perp$
  $T_i^s := \mathbf{true}$
  $k_0 := k_i^s$; $k_1 \leftarrow\!\!\$\ \mathcal{K}$
  Return k_b

If query=$\mathsf{SessionKeyReveal}(i,s)$:
  If $\Psi_i^s \neq \mathbf{accept}$: Return $\perp$
  If $T_i^s = \mathbf{true}$: Return $\perp$
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [\ell]$ s.t. $T_j^t = \mathbf{true}$:
    If $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s) \wedge k_j^t = k_i^s$: Return $\perp$
  $kRev_i^s := \mathbf{true}$; Return k_i^s

$\underline{\mathcal{O}_{\mathsf{AKE}}(\mathsf{query})}$:
If query=$\mathsf{Send}(i,s,j,\mathsf{msg})$:
  If $\Psi_i^s = \mathbf{accept}$: Return $\perp$
  If $\mathsf{msg} = \top$: //session is initiated
    $\mathsf{Pid}_i^s := j$
    $N \leftarrow\!\!\$\ \{0,1\}^\lambda$
    $\mathsf{msg}' := N$
  If $\mathsf{msg} = N$: //first message
    $\mathsf{Pid}_i^s := j$
    Let $n := (i-1)\mu + s$; $\hat{pk} := pk_n$
    $\sigma_1 \leftarrow\!\!\$\ \mathsf{Sign}(ssk_i, (P_i, P_j, \hat{pk}, N))$
    $\mathsf{msg}' := (\hat{pk}, \sigma_1)$
  If $\mathsf{msg} = (\hat{pk}, \sigma_1)$: //second message
    Choose $N \in \mathsf{Sent}_i^s$
    If $\mathsf{Pid}_i^s \neq j$ or $\mathsf{Ver}(vk_j, (P_j, P_i, \hat{pk}, N), \sigma_1) \neq 1$:
      $\Psi_i^s := \mathbf{reject}$; Return $\perp$
    If $crp_j = \mathbf{false}$:
      Then $\exists$ unique t s.t. $\hat{pk}$ output by π_j^t
      Let $n := (j-1)\mu + t$
      $(c, K_\beta) \leftarrow \mathcal{O}^{\beta}_{\text{ENCAP}}(n)$; $k_i^s := K_\beta$
    Else:
      $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(\hat{pk})$; $k_i^s := K$
    $\sigma_2 \leftarrow\!\!\$\ \mathsf{Sign}(ssk_i, (P_j, P_i, \hat{pk}, \sigma_1, c, N))$
    $\Psi_i^s := \mathbf{accept}$
    $\mathsf{msg}' := (c, \sigma_2)$
  If $\mathsf{msg} = (c, \sigma_2)$: //third message
    Choose $N \in \mathsf{Recv}_i^s$ and $(\hat{pk}, \sigma_1) \in \mathsf{Sent}_i^s$
    If $\mathsf{Pid} \neq j$ or $\mathsf{Ver}(vk_j, (P_i, P_j, \hat{pk}, \sigma_1, c, N), \sigma_2) \neq 1$:
      $\Psi_i^s := \mathbf{reject}$; Return $\perp$
    Let $n := (i-1)\mu + s$ and $j := \mathsf{Pid}_i^s$
    If $\exists t$ s. t. $\mathsf{Recv}_j^t = \{(\hat{pk}, \cdot)\} \wedge \mathsf{Sent}_j^t = \{N, (c, \cdot)\}$:
      $k_i^s := k_j^t$
    Else:
      $K \leftarrow \mathcal{O}_{\text{DECAP}}(n, c)$; $k_i^s := K$
    $\Psi_i^s := \mathbf{accept}$
    $\mathsf{msg}' := \emptyset$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{\mathsf{msg}\}$; $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{\mathsf{msg}'\}$
  If $\Psi_i^s = \mathbf{accept}$:
    If $crp_j = \mathbf{true}$: $\mathsf{Aflag}_i^s := \mathbf{true}$
  Return msg'

Figure 8: Adversary $\mathcal{B}_{\mathsf{KEM}}$ against MUSC-otCCA security of KEM for the proof of Theorem 2. Queries to $\mathcal{O}_{\mathsf{AKE}}$ where query$\in \{\mathsf{Corrupt}, \mathsf{RegisterCorrupt}\}$ are defined as in the original game $\mathsf{Exp}^{\mathsf{replay}}_{\mathsf{AKE}_{\mathsf{3msg}},\mu,\ell,\mathcal{A}}$ in Figure 5.

$\mathcal{B}_{\mathsf{KEM}}$ outputs k_i^s. As k_j^t is a random key, $\mathcal{A}$ learns nothing about the bit b. We have $\Pr[b^* = b] = \frac{1}{2}$ in this case and further $\Pr[\beta^* = \beta \mid \beta = 1] = \frac{1}{2}$.

It follows that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{musc\text{-}otcca}}_{\mathsf{KEM},\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) &= \left|\Pr[\beta^* = \beta] - \tfrac{1}{2}\right| \\ &= \left|\tfrac{1}{2} \cdot \Pr[\beta^* = \beta \mid \beta = 0] + \tfrac{1}{2} \cdot \Pr[\beta^* = \beta \mid \beta = 1] - \tfrac{1}{2}\right| \\ &= \left|\tfrac{1}{2} \cdot \Pr[\mathsf{Win}_3] + \tfrac{1}{2} \cdot \tfrac{1}{2} - \tfrac{1}{2}\right| = \tfrac{1}{2}\left|\Pr[\mathsf{Win}_3] - \tfrac{1}{2}\right| \ .\end{aligned}$$

Collecting the probabilities yields the bound in Theorem 2. □

Theorem 3 (Security of $\mathsf{AKE}_{\mathsf{2msg}}$ without State Reveals and Replay Attacks)**.** *For any adversary $\mathcal{A}$ against $\mathsf{AKE}_{\mathsf{2msg}}$ without state reveals and replay attacks, there exist an MU-EUF-CMA$^{\mathsf{corr}}$ adversary $\mathcal{B}_{\mathsf{SIG}}$ against SIG and an MUC-otCCA adversary $\mathcal{B}_{\mathsf{KEM}}$ against KEM such that*

$$\mathsf{Adv}_{\mathsf{AKE}_{\mathsf{2msg}},\mu,\ell}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{muc\text{-}otcca}}_{\mathsf{KEM},\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) + \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + (\mu\ell)^2 \cdot 2^{-\gamma} \ ,$$

where γ is the diversity parameter of KEM. Furthermore, $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{KEM}})$ and $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_{\mathsf{SIG}})$.

Proof Sketch. In the two-message protocol, the signatures ensure that the adversary can only forward messages for test sessions. However, the adversary may also replay a message containing a particular ephemeral public key $\hat{pk}$ in another session. Thus, we require multi-challenge security of the KEM. We still only need one decapsulation query as the session is closed after it receives the last message and has accepted or rejected the session key.

As we do not consider state reveals and the adversary will not see any ephemeral secret key $\hat{sk}$, we can follow the strategy of the proof of Theorem 2. Thus, we do not only replace the session keys of test sessions, but also of those that will be revealed, using MUC-otCCA security of KEM. The full proof is given in Appendix A.

6 Signatures with Tight Adaptive Corruptions

6.1 Pairing Groups and MDDH Assumptions

Let GGen be a pairing group generation algorithm that returns a description $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, \mathcal{P}_1, \mathcal{P}_2, e)$ of asymmetric pairing groups where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are cyclic groups of order q for a λ-bit prime q, $\mathcal{P}_1$ and $\mathcal{P}_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2$ is an efficient computable (non-degenerated) bilinear map. $\mathcal{P}_T := e(\mathcal{P}_1, \mathcal{P}_2)$ is a generator in $\mathbb{G}_T$. In this paper, we only consider Type III pairings, where $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no efficient homomorphism between them. All constructions in this paper can be easily instantiated with Type I pairings by setting $\mathbb{G}_1 = \mathbb{G}_2$ and defining the dimension k to be greater than 1.

We use the implicit representation of group elements as in [EHK+13]. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_q$ define $[a]_s = a\mathcal{P}_s \in \mathbb{G}_s$ as the implicit representation of a

in $\mathbb{G}_s$. Similarly, for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_q^{n\times m}$ we define $[\mathbf{A}]_s$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}_s$. $\mathsf{Span}(\mathbf{A}) := \{\mathbf{Ar} \mid \mathbf{r} \in \mathbb{Z}_q^m\} \subset \mathbb{Z}_q^n$ denotes the linear span of $\mathbf{A}$, and similarly $\mathsf{Span}([\mathbf{A}]_s) := \{[\mathbf{Ar}]_s \mid \mathbf{r} \in \mathbb{Z}_q^m\} \subset \mathbb{G}_s^n$. Note that it is efficient to compute $[\mathbf{AB}]_s$ given $([\mathbf{A}]_s, \mathbf{B})$ or $(\mathbf{A}, [\mathbf{B}]_s)$ with matching dimensions. We define $[\mathbf{A}]_1 \circ [\mathbf{B}]_2 := e([\mathbf{A}]_1, [\mathbf{B}]_2) = [\mathbf{AB}]_T$, which can be efficiently computed given $[\mathbf{A}]_1$ and $[\mathbf{B}]_2$.

We recall the definition of the Matrix Decisional Diffie-Hellman (MDDH) and related assumptions from [EHK+13].

Definition 16 (Matrix distribution)**.** *Let $k, \ell \in \mathbb{N}$ with $\ell > k$. We call $\mathcal{D}_{\ell,k}$ a matrix distribution if it outputs matrices in $\mathbb{Z}_q^{\ell\times k}$ of full rank k in polynomial time. Let $\mathcal{D}_k := \mathcal{D}_{k+1,k}$.*

For positive integers $k, \eta, n \in \mathbb{N}^+$ and a matrix $\mathbf{A} \in \mathbb{Z}_q^{(k+\eta)\times n}$, we denote the k rows of $\mathbf{A}$ by $\overline{\mathbf{A}} \in \mathbb{Z}_q^{k\times n}$ and the lower η rows of $\mathbf{A}$ by $\underline{\mathbf{A}} \in \mathbb{Z}_q^{\eta\times n}$. Without loss of generality, we assume $\overline{\mathbf{A}}$ for $\mathbf{A} \leftarrow_\$ \mathcal{D}_{\ell,k}$ form an invertible square matrix in $\mathbb{Z}_q^{k\times k}$. The $\mathcal{D}_{\ell,k}$-MDDH problem is to distinguish the two distributions $([\mathbf{A}], [\mathbf{Aw}])$ and $([\mathbf{A}], [\mathbf{u}])$ where $\mathbf{A} \leftarrow_\$ \mathcal{D}_{\ell,k}$, $\mathbf{w} \leftarrow_\$ \mathbb{Z}_q^k$ and $\mathbf{u} \leftarrow_\$ \mathbb{Z}_q^\ell$.

Definition 17 ($\mathcal{D}_{\ell,k}$-MDDH assumption)**.** *Let $\mathcal{D}_{\ell,k}$ be a matrix distribution and $s \in \{1, 2, T\}$. We say that the $\mathcal{D}_{\ell,k}$-MDDH assumption holds relative to GGen in group $\mathbb{G}_s$ if for all adversaries $\mathcal{A}$, it holds that*

$$\mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{D}_{\ell,k},\mathbb{G}_s}(\mathcal{A}) := |\Pr[\mathcal{A}(\mathcal{PG}, [\mathbf{A}]_s, [\mathbf{Aw}]_s) \Rightarrow 1] - \Pr[\mathcal{A}(\mathcal{PG}, [\mathbf{A}]_s, [\mathbf{u}]_s) \Rightarrow 1]|$$

is negligible where the probability is taken over $\mathcal{PG} \leftarrow_\$ \mathsf{GGen}(1^\lambda)$, $\mathbf{A} \leftarrow_\$ \mathcal{D}_{\ell,k}, \mathbf{w} \leftarrow_\$ \mathbb{Z}_q^k$ and $\mathbf{u} \leftarrow_\$ \mathbb{Z}_q^\ell$.

Definition 18 (Uniform distribution)**.** *Let $k, \ell \in \mathbb{N}^+$ with $\ell > k$. We call $\mathcal{U}_{\ell,k}$ a uniform distribution if it outputs uniformly random matrices in $\mathbb{Z}_q^{\ell\times k}$ of rank k in polynomial time. Let $\mathcal{U}_k := \mathcal{U}_{k+1,k}$.*

Lemma 3 ($\mathcal{D}_{\ell,k}$-MDDH $\Rightarrow$ $\mathcal{U}_k$-MDDH [EHK+13])**.** *Let $\ell, k \in \mathbb{N}_+$ with $\ell > k$ and let $\mathcal{D}_{\ell,k}$ be a matrix distribution. A $\mathcal{U}_k$-MDDH instance is at least as hard as an $\mathcal{D}_{\ell,k}$ instance. More precisely, for each adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$ with*

$$\mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{U}_k,\mathbb{G}_s}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{D}_{\ell,k},\mathbb{G}_s}(\mathcal{B})$$

and $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$.

The Kernel-Diffie-Hellman assumption ($\mathcal{D}_k$-KMDH) [MRV16] is a (weaker) computational analogue of the $\mathcal{D}_k$-MDDH Assumption.

Definition 19 ($\mathcal{D}_k$-KMDH)**.** *Let $\mathcal{D}_k$ be a matrix distribution. We say that the $\mathcal{D}_k$-Kernel Diffie-Hellman ($\mathcal{D}_k$-KMDH) assumption holds relative to a prime order group $\mathbb{G}_s$ for $s \in \{1, 2\}$ if for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathrm{KMDH}}_{\mathsf{GGen},\mathcal{D}_k,\mathbb{G}_s}(\mathcal{A}) := \Pr[\mathbf{c}^\top \mathbf{A} = \mathbf{0} \wedge \mathbf{c} \neq \mathbf{0} \mid [\mathbf{c}]_{3-s} \leftarrow_\$ \mathcal{A}(\mathcal{PG}, [\mathbf{A}]_s)],$$

where the probabilities are taken over $\mathcal{PG} \leftarrow_\$ \mathsf{GGen}(1^\lambda)$ and $\mathbf{A} \leftarrow_\$ \mathcal{D}_k$.

6.2 Previous Schemes with Tight Adaptive Corruptions

We will construct a signature scheme with tight MU-EUF-CMA$^{\mathsf{corr}}$ security and only small constant number of elements in signatures. Such a scheme has been proposed in [BHJ+15, Section 2.3] (called $\mathsf{SIG}_{\mathsf{C}}$), but we identify a gap in their proof. We now present the gap in their security proof and why we think it will be hard to close it.

The construction of $\mathsf{SIG}_{\mathsf{C}}$ follows the BKP IBE schemes [BKP14], namely, it tightly transforms an affine MAC [BKP14] into a signature in the multi-user setting. In order to have a tightly MU-EUF-CMA$^{\mathsf{corr}}$ secure signature scheme, the underlying MAC needs to be tightly secure against adaptive corruption of its secret keys in the multi-user setting. We will now point to potential problems in formally proving it.

We abstract the underlying MAC of $\mathsf{SIG}_{\mathsf{C}}$ as $\mathsf{MAC}_{\mathsf{BHJKL}}$: For message space $\{0,1\}^\ell$, it chooses $\mathbf{A}' \leftarrow_\$ \mathcal{D}_k$ and random vectors $\mathbf{x}_{i,j} \leftarrow_\$ \mathbb{Z}_q^k$ (for $1 \le i \le \ell$ and $j = 0, 1$). Then it defines $\mathbf{B} := \overline{\mathbf{A}'} \in \mathbb{Z}_q^{k\times k}$ and publishes parameters $\mathsf{pp} := ([\mathbf{B}]_1, ([\mathbf{B}^\top \mathbf{x}_{i,j}]_1)_{1\le i\le \ell, j=0,1})$. For each user n, it chooses its MAC secret key as $[x'_n]_1 \leftarrow_\$ \mathbb{G}_1$, and its MAC tag consist of $([\mathbf{t}]_1, [u]_1)$, where

$$\begin{aligned} \mathbf{t} &= \mathbf{B}\mathbf{s} \in \mathbb{Z}_q^k \quad \text{for} \quad \mathbf{s} \leftarrow_\$ \mathbb{Z}_q^k \\ u &= x'_n + \mathbf{t}^\top \underbrace{\sum\nolimits_i \mathbf{x}_{i,\mathsf{m}_i}}_{=:\mathbf{x}(\mathsf{m})} \in \mathbb{Z}_q. \end{aligned} \tag{3}$$

In their security proof, they argue that $[u]_1$ in the MAC tagging queries is pseudorandom, given pp and some of the secret keys $[x'_n]_1$ (via the adaptive corruption queries) to an adversary.[6] In achieving this, they define a sequence of hybrids H_j for $1 \le j \le \ell$. In each H_j, they replace x'_n for each user n with $\mathsf{RF}_{n,j}(\mathsf{m}_{|j})$, where $\mathsf{RF}_{n,j} : \{0,1\}^j \to \mathbb{Z}_q$ is a random function and m is the first tagging query to user n, and generate the MAC tag of m' as

$$u = \mathsf{RF}_{n,j}(\mathsf{m}'_{|j}) + \mathbf{t}^\top \mathbf{x}(\mathsf{m}') \tag{4}$$

with $\mathbf{t}$ as in Equation (3).

In their final step (between H_ℓ and Game 4), they argue that the distribution of $u = \mathsf{RF}_{n,\ell}(\mathsf{m}') + \mathbf{t}^\top \mathbf{x}(\mathsf{m}')$ is uniformly random (as in Game 4) even for an unbounded adversary, given pp and adaptive corruptions. Then they conclude that H_ℓ (where $u = \mathsf{RF}_{n,\ell}(\mathsf{m}') + \mathbf{t}^\top \mathbf{x}(\mathsf{m}')$) and Game 4 (where u is chosen uniformly at random) are identical and $\Pr[\chi_4] = \Pr[H_\ell = 1]$ (according to their notation). However, this is not the case: $\mathbf{B} \in \mathbb{Z}_q^{k\times k}$ is full-rank and thus, given $[\mathbf{B}^\top \mathbf{x}_{i,j}]_1$ in pp, $\mathbf{x}_{i,j} \in \mathbb{Z}_q^k$ is uniquely defined. (For concreteness, imagine a simple example where an (unbounded) adversary $\mathcal{A}$ only queries one MAC tag for message m for user n and then asks for the secret key $[x'_n]_1 := \mathsf{RF}_{n,\ell}(\mathsf{m})$ of user n. Then, $\mathcal{A}$ sees that $u = \mathsf{RF}_{n,\ell}(\mathsf{m}) + \mathbf{t}^\top \mathbf{x}(\mathsf{m})$ is uniquely defined by $[x'_n]_1, [\mathbf{t}]_1$ and pp in H_ℓ, while u is uniformly at random in Game 4.) We suppose this gap is inherent, since the terms $\mathbf{B}^\top \mathbf{x}_{i,j}$ completely leak the information about $\mathbf{x}_{i,j}$. This is also the same reason why the BKP MAC cannot be used to construct a tightly secure hierarchical IBE (HIBE) (cf. [LP19] for more discussion).

[6] This is different to the BKP IBE where $[\mathbf{B}^\top \mathbf{x}_{i,j}]_1$ and $[x'_n]_1$ are not available to an adversary.

To resolve this, we follow the tightly secure HIBE approach in [LP19] and choose $\mathbf{B} \leftarrow_\$ \mathbb{Z}_q^{3k\times k}$. Now, there is a non-zero kernel matrix $\mathbf{B}^\perp \in \mathbb{Z}_q^{3k\times 2k}$ for $\mathbf{B}$ (with overwhelming probability), and the mapping $\mathbf{x}_{i,j} \in \mathbb{Z}_q^{3k} \mapsto \mathbf{B}^\top \mathbf{x}_{i,j} \in \mathbb{Z}_q^k$ is lossy. In particular, the information about $\mathbf{x}_{i,j}$ in the orthogonal space of $\mathbf{B}$ is perfectly hidden from (unbounded) adversaries, given $\mathbf{B}^\top \mathbf{x}_{i,j}$.

6.3 Our Construction

Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a function chosen from a collision-resistant hash function family $\mathcal{H}$. Our signature scheme $\mathsf{SIG}_{\mathsf{MDDH}} := (\mathsf{SIG.Setup}, \mathsf{SIG.Gen}, \mathsf{Sign}, \mathsf{Ver})$ is defined in Figure 9. Correctness can be verified as

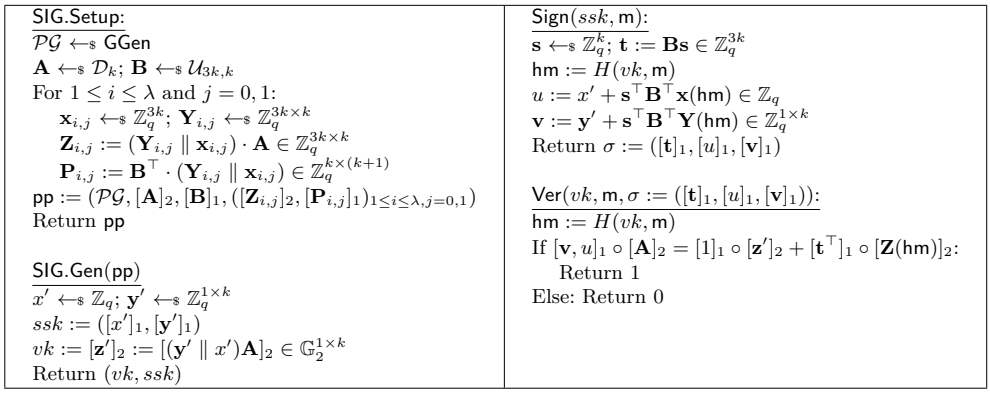

Figure 9: Our signature scheme with tight adaptive corruptions, where for $\mathsf{hm} \in \{0,1\}^\lambda$ we define the functions $\mathbf{x}(\mathsf{hm}) := \sum_{i=1}^\lambda \mathbf{x}_{i,\mathsf{hm}_i}$, $\mathbf{Y}(\mathsf{hm}) := \sum_{i=1}^\lambda \mathbf{Y}_{i,\mathsf{hm}_i}$, $\mathbf{Z}(\mathsf{hm}) := \sum_{i=1}^\lambda \mathbf{Z}_{i,\mathsf{hm}_i}$, and $\mathbf{P}(\mathsf{hm}) := \sum_{i=1}^\lambda \mathbf{P}_{i,\mathsf{hm}_i}$.

$$[\mathbf{v}, u]_1 \circ [\mathbf{A}]_2 = [(\mathbf{y}', x') \cdot \mathbf{A} + \mathbf{t}^\top \cdot (\mathbf{Y}(\mathsf{hm}) \mid \mathbf{x}(\mathsf{hm})) \cdot \mathbf{A}]_T$$

for $([\mathbf{t}]_1, [u]_1, [\mathbf{v}]_1) \leftarrow_\$ \mathsf{Sign}(ssk, \mathsf{m})$.

Theorem 4 (Security of $\mathsf{SIG}_{\mathsf{MDDH}}$)**.** *For any adversary $\mathcal{A}$ against the* MU-EUF-CMA$^{\mathsf{corr}}$ *security of $\mathsf{SIG}_{\mathsf{MDDH}}$, there are adversaries $\mathcal{B}$ against the collision resistance of $\mathcal{H}$, $\mathcal{B}_1$ against the $\mathcal{U}_{3k,k}$-*MDDH *assumption over $\mathbb{G}_1$ and $\mathcal{B}_2$ against the $\mathcal{D}_k$-*KMDH *assumption over $\mathbb{G}_2$ with*

$$\begin{aligned}\Pr[\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}} \Rightarrow 1] \le & \mathsf{Adv}^{\mathsf{cr}}_{\mathcal{H}}(\mathcal{B}) + (8k\lambda + 2k)\mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{U}_{3k,k},\mathbb{G}_1}(\mathcal{B}_1) \\ & + \mathsf{Adv}^{\mathrm{KMDH}}_{\mathsf{GGen},\mathcal{D}_k,\mathbb{G}_2}(\mathcal{B}_2) + \frac{4\lambda + 2k + 2}{q-1},\end{aligned}$$

where $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{B}_2)$.

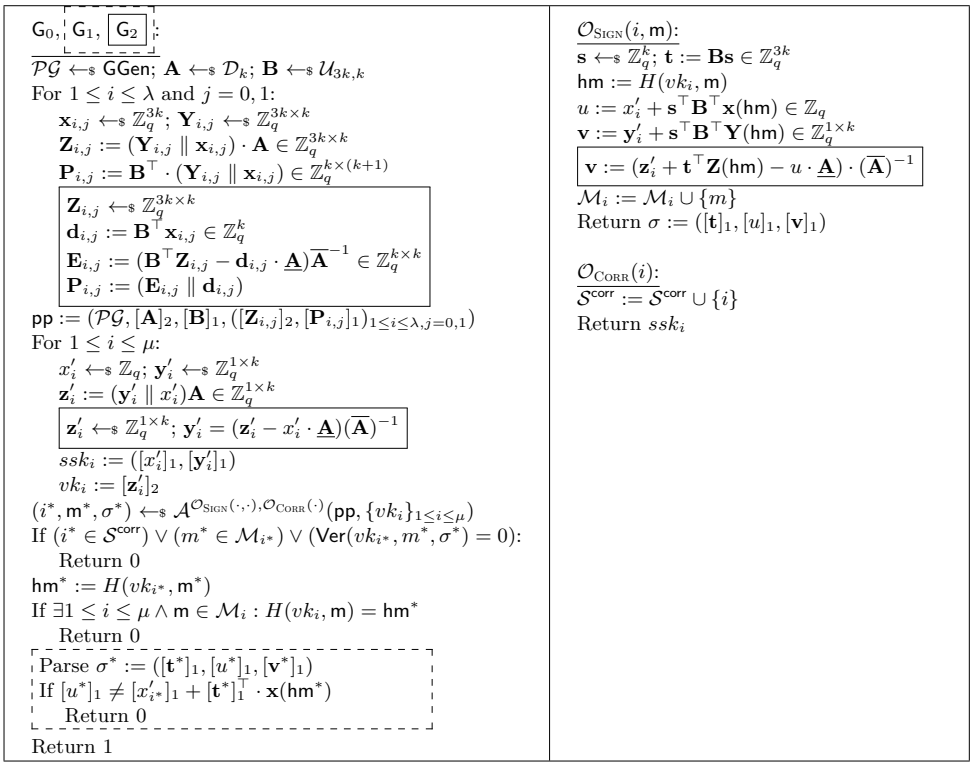

Figure 10: Games used to prove Theorem 4.

Proof. We prove the tight MU-EUF-CMA$^{\mathsf{corr}}$ security of $\mathsf{SIG}_{\mathsf{MDDH}}$ with a sequence of games given in Figure 10. Let $\mathcal{A}$ be an adversary against the MU-EUF-CMA$^{\mathsf{corr}}$ security of $\mathsf{SIG}_{\mathsf{MDDH}}$, and let Win_i denote the probability that G_i returns 1.

Game G_0: G_0 is the original experiment $\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}}$ (cf. Definition 3). In addition to the original game, we add a rejection rule if there is a collision between the forgery and a signing query, namely, $H(vk_{i^*}, \mathsf{m}^*) = H(vk_i, \mathsf{m})$ where (i, m) is one of the signing queries. By the collision resistance of H, we have

$$|\Pr[\mathsf{Exp}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu,\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{Win}_0]| \le \mathsf{Adv}^{\mathsf{cr}}_{\mathcal{H}}(\mathcal{B}).$$

For better readability, we assume all the signing queries are distinct for the following games. If the same (i, m) is asked multiple times, we can take the first response $([\mathbf{t}]_1, [u]_1, [\mathbf{v}]_1)$ and answer the repeated queries with the re-randomization $([\mathbf{t}']_1, [u']_1, [\mathbf{v}']_1)$ as $\mathbf{t}' := \mathbf{t} + \mathbf{Bs}'$ (for $\mathbf{s}' \leftarrow\!\!\$\ \mathbb{Z}_q^k$), $u' := u + \mathbf{s}'^\top(\mathbf{B}^\top \mathbf{x}(\mathsf{hm}))$ and $\mathbf{v}' := \mathbf{v} + \mathbf{s}'^\top(\mathbf{B}^\top \mathbf{x}(\mathsf{hm}))$ and $\mathsf{hm} := H(vk_i, \mathsf{m})$. Note that this will not change the view of $\mathcal{A}$.

Game G_1: For verifying the forgery, in addition to using Ver, we use the secret $[x'_{i^*}]_1$ and $([\mathbf{x}_{j,b}]_1)_{1\le j\le\lambda}$ to check if $([\mathbf{t}^*]_1, [u^*]_1)$ in the forgery satisfies the following equation:

$$[u^*]_1 = [x'_{i^*}]_1 + [\mathbf{t}^*]_1^\top \cdot \mathbf{x}(\mathsf{hm}^*). \tag{5}$$

We note that

$$\begin{aligned}&\mathsf{Ver}(vk_{i^*}, m^*, \sigma^*) = 1\\ \Leftrightarrow &(\mathbf{v} \parallel u) \cdot \mathbf{A} = (\mathbf{y}'_{i^*} \parallel x'_{i^*})\mathbf{A} + \mathbf{t}^{*\top} \cdot (\mathbf{Y}(\mathsf{hm}) \parallel \mathbf{x}(\mathsf{hm})) \cdot \mathbf{A}.\end{aligned}$$

Thus, if Equation (5) does not hold, then the vector $[(\mathbf{v} \parallel u)]_1 - ([\mathbf{y}'_{i^*} \parallel x'_{i^*}]_1 + [\mathbf{t}^{*\top}]_1 \cdot \mathbf{x}(\mathsf{hm}^*)) \in \mathbb{G}_1^{1\times(k+1)}$ is non-zero and orthogonal to $[\mathbf{A}]_2$. Therefore, we bound the difference between G_0 and G_1 with the $\mathcal{D}_k$-KMDH assumption as

$$|\Pr[\mathsf{Win}_0] - \Pr[\mathsf{Win}_1]| \leq \mathsf{Adv}^{\mathrm{KMDH}}_{\mathsf{GGen},\mathcal{D}_k,\mathbb{G}_2}(\mathcal{B}).$$

Game G_2: We do not use the values $\mathbf{Y}_{j,b}$ (for $1 \leq j \leq \lambda$ and $b = 0, 1$) and $\mathbf{y}'_i$ (for $1 \leq i \leq \mu$) to simulate G_2. We make this change by substituting all $\mathbf{Y}_{j,b}$ and $\mathbf{y}'_i$ using the formulas

$$\mathbf{Y}^{\top}_{j,b} = (\mathbf{Z}_{j,b} - \mathbf{x}_{j,b} \cdot \underline{\mathbf{A}})(\overline{\mathbf{A}})^{-1} \text{ and } \mathbf{y}'_i = (\mathbf{z}'_i - x'_i \cdot \underline{\mathbf{A}})(\overline{\mathbf{A}})^{-1}, \tag{6}$$

respectively. More precisely, the public parameters pp are computed by picking $\mathbf{Z}_{j,b}$ and $\mathbf{x}_{j,b}$ at random and then defining $\mathbf{Y}_{j,b}$ using Equation (6). The verification keys vk_i for user i ($1 \leq i \leq \mu$) are computed by picking $\mathbf{z}'_i$ and x'_i at random. For $\mathcal{O}_{\mathrm{SIGN}}(i, \mathsf{m})$, we now compute

$$\begin{aligned}\mathbf{v} &:= \mathbf{y}'_i + \mathbf{t}^{\top}\mathbf{Y}(\mathsf{hm}) \in \mathbb{Z}_q^{1\times k}\\ &= (\mathbf{z}'_i - x'_i \cdot \underline{\mathbf{A}})(\overline{\mathbf{A}})^{-1} + \mathbf{t}^{\top}(\mathbf{Z}(\mathsf{hm}) - \mathbf{x}(\mathsf{hm}) \cdot \underline{\mathbf{A}})(\overline{\mathbf{A}})^{-1}\\ &= (\mathbf{z}'_i + \mathbf{t}^{\top}\mathbf{Z}(\mathsf{hm}) - \underbrace{(x'_i + \mathbf{t}^{\top}\mathbf{x}(\mathsf{hm}))}_{=u} \cdot \underline{\mathbf{A}})(\overline{\mathbf{A}})^{-1}.\end{aligned}$$

The secret verification of the forgery can be done by knowing x'_{i^*} and $\mathbf{x}_{j,b}$.

The changes in G_2 are only conceptual, since Equations (6) are equivalent to $\mathbf{Z}_{j,b} = (\mathbf{Y}_{j,b} \parallel \mathbf{x}_{j,b})\mathbf{A}$ and $\mathbf{z}'_i = (\mathbf{y}'_i \parallel x'_i)\mathbf{A}$. Thus, we have

$$\Pr[\mathsf{Win}_1] = \Pr[\mathsf{Win}_2].$$

In order to bound $\Pr[\mathsf{Win}_2]$, consider a "message authentication code" MAC which is defined as follows.

- The public parameters consist of $\mathsf{pp}_{\mathsf{MAC}} := (\mathcal{PG}, [\mathbf{B}]_1, ([\mathbf{d}_{i,j}]_1)_{1\leq i\leq\lambda, j=0,1})$, where $\mathbf{d}_{i,j} := \mathbf{B}^{\top}\mathbf{x}_{i,j} \in \mathbb{Z}_q^k$ for $\mathbf{x}_{i,j} \leftarrow_{\$} \mathbb{Z}_q^{3k}$ and $\mathbf{B} \leftarrow_{\$} \mathcal{U}_{3k,k}$.
- The secret key is $[x']_1$.
- The MAC tag on hm is $([\mathbf{t}]_1, [u]_1)$, where $\mathbf{t} := \mathbf{B}\mathbf{s}$ and $u := x' + \mathbf{t}^{\top}\mathbf{x}(\mathsf{hm})$, for $\mathbf{s} \leftarrow_{\$} \mathbb{Z}_q^k$.

Note that strictly speaking MAC is not a MAC since verification cannot only be done efficiently by knowing the values $\mathbf{x}_{i,j}$.

The following lemma states MU-EUF-CMA$^{\mathsf{corr}}$ security of MAC.

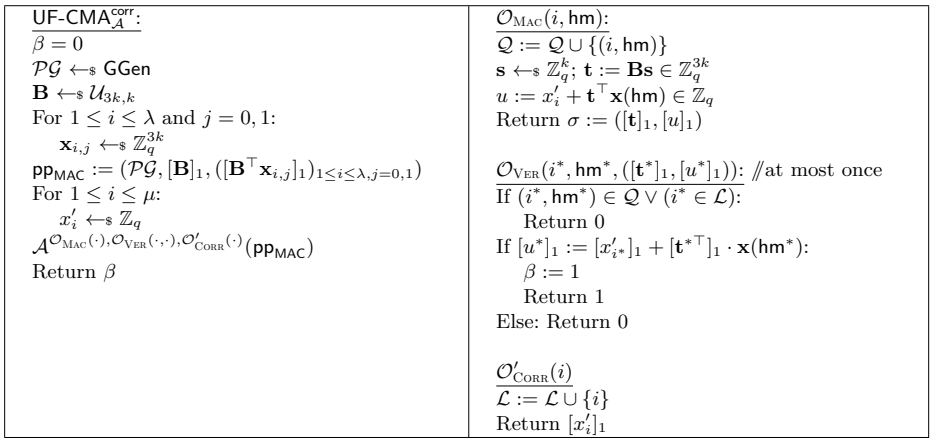

Figure 11: Game $\mathsf{UF\text{-}CMA}^{\mathsf{corr}}$ for Lemma 4.

Lemma 4 (Core Lemma)**.** *For every adversaries $\mathcal{A}$ interacting with $\mathsf{UF\text{-}CMA}^{\mathsf{corr}}$, there exists an adversary $\mathcal{B}$ against the $\mathcal{U}_{3k,k}$-*MDDH *assumption in $\mathbb{G}_1$ with*

$$\Pr[\mathsf{UF\text{-}CMA}^{\mathsf{corr}}_{\mathcal{A}} \Rightarrow 1] \le (8k\lambda + 2k) \cdot \mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen}, \mathcal{U}_{3k,k}, \mathbb{G}_1}(\mathcal{B}_1) + \frac{4\lambda + 2k + 2}{q - 1},$$

and $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$, where Q_e is the number of $\mathcal{A}$'s queries to $\mathcal{O}_{MAC}$.

The proof is postponed to Appendix B.

Finally, we bound the probability that the adversary wins in G_2 using our Core Lemma (Lemma 4) by constructing an adversary $\mathcal{B}_{\mathsf{MAC}}$ as in Figure 12.

$$\Pr[\mathsf{Win}_2] = \Pr[\mathsf{UF\text{-}CMA}^{\mathsf{corr}}_{\mathcal{B}_{\mathsf{MAC}}} \Rightarrow 1].$$

In order to analyze $\Pr[\mathsf{Win}_2]$ we argue as follows. The simulated pp and $(vk_i)_{1 \le i \le \mu}$ are distributed as in G_2. Further, queries to $\mathcal{O}_{\mathrm{SIGN}}$ and $\mathcal{O}_{\mathrm{CORR}}$ from ssk_i can be perfectly simulated using $\mathcal{O}_{\mathrm{MAC}}$ and $\mathcal{O}'_{\mathrm{CORR}}$, respectively. The additional group elements $[\mathbf{v}]_1$ from σ and $[\mathbf{y}'_i]_1$ can be simulated as in G_2. Finally, using a valid forgery $(i^*, \mathsf{m}^*, \sigma^*)$ output by $\mathcal{A}$, $\mathcal{B}_{\mathsf{MAC}}$ wins its own game by calling $\mathcal{O}_{\mathrm{VER}}(i^*, \mathsf{hm}^*, ([\mathbf{t}^*]_1, [u^*]_1)$, where $([\mathbf{t}^*]_1, [u^*]_1)$ is a valid MAC tag on hm^* for user i^*. □

$\mathcal{B}_{\mathsf{MAC}}^{\mathcal{O}_{\mathrm{MAC}}(\cdot),\mathcal{O}_{\mathrm{VER}}(\cdot),\mathcal{O}'_{\mathrm{CORR}}(\cdot)}(\mathsf{pp}_{\mathsf{MAC}})$:
Parse $\mathsf{pp}_{\mathsf{MAC}} =: (\mathcal{PG}, [\mathbf{B}]_1, ([\mathbf{d}_{i,j}]_1)_{1\le i\lambda, j=0,1})$
$\mathbf{A} \leftarrow\!\!\$\ \mathcal{D}_k$
For $1 \le i \le \lambda$ and $j = 0, 1$:
  $\mathbf{Z}_{i,j} \leftarrow\!\!\$\ \mathbb{Z}_q^{3k\times k}$
  $\mathbf{E}_{i,j} := (\mathbf{B}^\top \mathbf{Z}_{i,j} - \mathbf{d}_{i,j} \cdot \underline{\mathbf{A}})\overline{\mathbf{A}}^{-1} \in \mathbb{Z}_q^{k\times k}$
  $\mathbf{P}_{i,j} := (\mathbf{E}_{i,j} \parallel \mathbf{d}_{i,j})$
$\mathsf{pp} := (\mathcal{PG}, [\mathbf{A}]_2, [\mathbf{B}]_1, ([\mathbf{Z}_{i,j}]_2, [\mathbf{P}_{i,j}]_1)_{1\le i\le\lambda, j=0,1})$
For $1 \le i \le \mu$:
  $\mathbf{z}'_i \leftarrow\!\!\$\ \mathbb{Z}_q^{1\times k}$
  $vk_i := [\mathbf{z}'_i]_2$ // ssk_i is undefined
$(i^*, \mathsf{m}^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathrm{SIGN}}(\cdot,\cdot),\mathcal{O}_{\mathrm{CORR}}(\cdot)}(\mathsf{pp}, \{vk_i\}_{1\le i\le\mu})$
If $(i^* \in \mathcal{S}^{\mathsf{corr}}) \vee (m^* \in \mathcal{M}_{i^*}) \vee (\mathsf{Ver}(vk_{i^*}, m^*, \sigma^*) = 0)$:
  Return 0
$\mathsf{hm}^* := H(vk_{i^*}, \mathsf{m}^*)$
If $\exists 1 \le i \le \mu \wedge \mathsf{m} \in \mathcal{M}_i : H(vk_i, \mathsf{m}) = \mathsf{hm}^*$
  Return 0
Parse $\sigma^* := ([\mathbf{t}^*]_1, [u^*]_1, [\mathbf{v}^*]_1)$
$\mathcal{O}_{\mathrm{VER}}(i^*, \mathsf{hm}^*, [\mathbf{t}^*]_1, [u^*]_1)$
Return 1

$\mathcal{O}_{\mathrm{SIGN}}(i, \mathsf{m})$:
$\mathsf{hm} := H(vk_i, \mathsf{m})$
$([\mathbf{t}]_1, [u]_1) \leftarrow\!\!\$\ \mathcal{O}_{\mathrm{MAC}}(\mathsf{hm})$
$\mathbf{v} := (\mathbf{z}'_i + \mathbf{t}^\top \mathbf{Z}(\mathsf{hm}) - u \cdot \underline{\mathbf{A}}) \cdot (\overline{\mathbf{A}})^{-1}$
$\mathcal{M}_i := \mathcal{M}_i \cup \{m\}$
Return $\sigma := ([\mathbf{t}]_1, [u]_1, [\mathbf{v}]_1)$

$\mathcal{O}_{\mathrm{CORR}}(i)$:
$\mathcal{S}^{\mathsf{corr}} := \mathcal{S}^{\mathsf{corr}} \cup \{i\}$
$[x'_i]_1 \leftarrow \mathcal{O}'_{\mathrm{CORR}}(i)$
$\mathbf{y}'_i = (\mathbf{z}'_i - x'_i \cdot \underline{\mathbf{A}})(\overline{\mathbf{A}})^{-1}$
Return $ssk_i := ([x'_i]_1, [\mathbf{y}'_i]_1)$

Figure 12: Reduction $\mathcal{B}_{\mathsf{MAC}}$ to bound the winning probability in G_2. $\mathcal{B}_{\mathsf{MAC}}$ receives $\mathsf{pp}_{\mathsf{MAC}}$ and gets oracle access to $\mathcal{O}_{\mathrm{MAC}}$ and $\mathcal{O}_{\mathrm{VER}}$, and $\mathcal{O}'_{\mathrm{CORR}}$ as in Figure 11.

7 Concrete Instantiation of Our AKE Protocols

In this section, we present concrete instantiation of our AKE protocols. We first provide a generic construction of ϵ-MU-SIM KEM from Universal$_2$ Hash Proof System (HPS) in Subsection 7.2, then give an instantiation of HPS from MDDH (and a function H) in Subsection 7.3. This yields concrete ϵ-MU-SIM KEM schemes based on the MDDH assumptions.

For $\mathsf{AKE}_{\mathsf{3msg}}$, we use our new signature scheme $\mathsf{SIG}_{\mathsf{MDDH}}$ (Figure 9) and the ϵ-MU-SIM KEM constructed from the MDDH-based hash proof system. For $\mathsf{AKE}_{\mathsf{3msg}}^{\mathsf{state}}$, the symmetric encryption scheme to protect against state reveals can be instantiated using any weakly secure (deterministic) encryption scheme such as AES or even a weak PRF (cf. Remark 1).

In practice, we can consider the function H used in the HPS instantiation in Subsection 7.3 as a collision-resistant hash function and thus choose parameter $t = 1$ (see Remark 7 and Remark 8). Then, the resulting KEM public key consists of $2k$ group elements and the ciphertext of $k + 1$ group elements. A signature consists of $4k + 1$ group elements, cf. Figure 9. Therefore, the first message is a bitstring of length λ, the second message consists of $6k + 1$ group elements and the third message consists of $5k + 2$ group elements. For $k = 1$, we get an efficient SXDH-based scheme with 15 elements in total.

We instantiate protocol $\mathsf{AKE}_{\mathsf{2msg}}$ using our signature scheme from Figure 9 and the MUC-otCCA secure KEM from Han *et al.* [HLLG19]. γ-diversity of the KEM is proven in [LLGW20, Appendix D.2]. We analyze the communication complexity of $\mathsf{AKE}_{\mathsf{2msg}}$ as follows. The KEM public key consists of $k^2 + 3k$ group elements and the ciphertext of $2k+3$ group elements. A signature consists of $4k+1$ group elements. Therefore, the first

message consists of k^2+7k+1 group elements and the second message consists of $6k+4$ group elements. For $k=1$, we get an efficient SXDH-based scheme with $9+10=19$ group elements in total.

For an overview we refer to Table 1 of the introduction.

7.1 Definitions of HPS

We give the formal definition of Hash Proof System (HPS) according to [CS02].

Definition 20 (HPS)**.** *A hash proof system* $\mathsf{HPS} = (\mathsf{HPS.Setup}, \mathsf{Pub}, \mathsf{Priv})$ *consists of a tuple of PPT algorithms:*

- $\mathsf{pp} \leftarrow_\$ \mathsf{HPS.Setup}$*: The setup algorithm outputs a public parameter* pp*, which implicitly defines* $(\mathcal{L}, \mathcal{X}, \mathcal{SK}, \mathcal{PK}, \Pi, \Lambda_{(\cdot)}, \alpha)$*, where* $\mathcal{L} \subseteq \mathcal{X}$ *is an NP-language with universe* $\mathcal{X}$*,* $\mathcal{SK}$ *is the hashing key space,* $\mathcal{PK}$ *is the projection key space,* Π *is the hash value space,* $\Lambda_{(\cdot)} : \mathcal{X} \longrightarrow \Pi$ *is a family of efficiently computable hash functions indexed by a hashing key* $sk \in \mathcal{SK}$*, and* $\alpha : \mathcal{SK} \longrightarrow \mathcal{PK}$ *is an efficiently computable projection function.*

  We assume that there are PPT algorithms for sampling $x \leftarrow_\$ \mathcal{L}$ *uniformly together with a witness* w*, sampling* $x \leftarrow_\$ \mathcal{X} \setminus \mathcal{L}$ *uniformly, sampling* $x \leftarrow_\$ \mathcal{X}$ *uniformly, and sampling* $sk \leftarrow_\$ \mathcal{SK}$ *uniformly. We require* pp *to be an implicit input of other algorithms.*

- $\pi \leftarrow \mathsf{Pub}(pk, x, w)$*: The deterministic public evaluation algorithm outputs the hash value* $\pi = \Lambda_{sk}(x) \in \Pi$ *of* $x \in \mathcal{L}$*, with help of a projection key* $pk = \alpha(sk)$ *and a witness* w *for* $x \in \mathcal{L}$*.*

- $\pi \leftarrow \mathsf{Priv}(sk, x)$*: The deterministic private evaluation algorithm outputs the hash value* $\pi = \Lambda_{sk}(x) \in \Pi$ *of* $x \in \mathcal{X}$ *with help of the hashing key* sk*.*

We require that for all $\mathsf{pp} \in \mathsf{HPS.Setup}$*, all hashing keys* $sk \in \mathcal{SK}$ *with the corresponding projection key* $pk := \alpha(sk)$*, all* $x \in \mathcal{L}$ *with all possible witnesses* w*, it holds that* $\mathsf{Pub}(pk, x, w) = \Lambda_{sk}(x) = \mathsf{Priv}(sk, x)$*.*

HPS is associated with a subset membership problem (SMP). Any SMP can be extended to multi-fold SMP with a security loss of the number of folds.

Definition 21 (SMP)**.** *Let* $\mathcal{A}$ *be an adversary against the subset membership problem (SMP) of* HPS*. The advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{smp}}_{\mathsf{HPS}}(\mathcal{A}) := \left|\Pr\left[\mathcal{A}(\mathsf{pp}, x) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{pp}, x') = 1\right]\right|,$$

where $\mathsf{pp} \leftarrow_\$ \mathsf{HPS.Setup}$*,* $x \leftarrow_\$ \mathcal{L}$*, and* $x' \leftarrow_\$ \mathcal{X} \setminus \mathcal{L}$*.*

Definition 22 (Multi-fold SMP)**.** *Let* $\mathcal{A}$ *be an adversary against the multi-fold subset membership problem (SMP) of* HPS*. The advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{msmp}}_{\mathsf{HPS},\mu}(\mathcal{A}) := \left|\Pr\left[\mathcal{A}(\mathsf{pp}, \{x_j\}_{j\in[\mu]}) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{pp}, \{x'_j\}_{j\in[\mu]}) = 1\right]\right|$$

where $\mathsf{pp} \leftarrow_\$ \mathsf{HPS.Setup}$*,* $x_1, \cdots, x_\mu \leftarrow_\$ \mathcal{L}$ *and* $x'_1, \cdots, x'_\mu \leftarrow_\$ \mathcal{X} \setminus \mathcal{L}$*.*

For some random-self reducible problems like MDDH, the hardness of multi-fold SMP can be tightly reduced to that of SMP, i.e., $\mathsf{Adv}^{\mathsf{msmp}}_{\mathsf{HPS},\mu}(\mathcal{A}) \approx \mathsf{Adv}^{\mathsf{smp}}_{\mathsf{HPS}}(\mathcal{B})$. See Section 6.1 for more details. Our instantiations of HPS from MDDH is shown in Section 7.3.

Definition 23 (ϵ-Universal$_2$ of HPS)**.** *A hash proof system* HPS *is* ϵ*-universal*$_2$*, if for all* $\mathsf{pp} \in \mathsf{HPS.Setup}$*, for all* $pk \in \mathcal{PK}$*, all* $x, x^* \in \mathcal{X}$ *with* $x^* \notin \mathcal{L} \cup \{x\}$*, and all* $\pi, \pi^* \in \Pi$*, it holds that*

$$\Pr[\Lambda_{sk}(x^*) = \pi^* \mid \alpha(sk) = pk, \Lambda_{sk}(x) = \pi] \leq \epsilon,$$

where the probability is over $sk \leftarrow_\$ \mathcal{SK}$*. If* $\epsilon = 1/|\Pi|$*, then* HPS *is perfectly universal*$_2$*.*

Below we define an extracting notion, which is adapted from [DKPW12].

Definition 24 (γ-Extracting of HPS)**.** *A hash proof system* HPS *is* γ*-extracting, if it is* γ*-extracting*$_1$ *and* γ*-extracting*$_2$*.*

(1) γ***-Extracting*****$_1$*****:** *For all* $\mathsf{pp} \in \mathsf{HPS.Setup}$*, all* $x \in \mathcal{L}$ *and all* $\pi \in \Pi$*, it holds that* $\Pr_{sk \leftarrow_\$ \mathcal{SK}}[\Lambda_{sk}(x) = \pi] \leq 2^{-\gamma}$*;*

(2) γ***-Extracting*****$_2$*****:** *For all* $\mathsf{pp} \in \mathsf{HPS.Setup}$*, all* $x, x^* \in \mathcal{L}$ *with* $x^* \neq x$*, and all* $\pi, \pi^* \in \Pi$*, it holds that* $\Pr_{sk \leftarrow_\$ \mathcal{SK}}[\Lambda_{sk}(x^*) = \pi^* \mid \Lambda_{sk}(x) = \pi] \leq 2^{-\gamma}$*.*

If $\gamma = \log|\Pi|$*, then* HPS *is perfectly extracting.*

7.2 ϵ-MU-SIM Secure KEM from Universal$_2$-HPS

Let $\mathsf{HPS} = (\mathsf{HPS.Setup}, \mathsf{Pub}, \mathsf{Priv})$ be a universal$_2$-HPS associated with a hard multi-fold subset membership problem (SMP) and enjoying an extracting property. We present a simple construction $\mathsf{KEM}_{\mathsf{HPS}} = (\mathsf{KEM.Setup}, \mathsf{KEM.Gen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Encap}^*)$ from HPS as follows.

- $\mathsf{KEM.Setup}$: $\mathsf{pp} \leftarrow_\$ \mathsf{HPS.Setup}$, where $\mathsf{pp} = (\mathcal{L}, \mathcal{X}, \mathcal{SK}, \mathcal{PK}, \Pi, \Lambda_{(\cdot)}, \alpha)$. Return $\mathsf{pp}_{\mathsf{KEM}} := \mathsf{pp}$. Here the encapsulation key space $\mathcal{K} := \Pi$, the ciphertext space $\mathcal{CT} := \mathcal{X}$ and the public key & secret key spaces are $\mathcal{PK} \times \mathcal{SK}$.
- $\mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$: Choose $sk \leftarrow_\$ \mathcal{SK}$ and return $(pk := \alpha(sk), sk)$.
- $\mathsf{Encap}(pk)$: Sample $x \leftarrow_\$ \mathcal{L}$ with witness w and return $(c := x, K := \mathsf{Pub}(pk, x, w) = \Lambda_{sk}(x))$.
- $\mathsf{Decap}(sk, c)$: Compute and return $K' := \mathsf{Priv}(sk, c)$.
- $\mathsf{Encap}^*(sk)$: Sample $x \leftarrow_\$ \mathcal{X} \setminus \mathcal{L}$ and return $(c := x, K := \mathsf{Priv}(sk, x) = \Lambda_{sk}(x))$.

The correctness of $\mathsf{KEM}_{\mathsf{HPS}}$ follows from the correctness of HPS. Now we prove the strong ϵ-MU-SIM security of $\mathsf{KEM}_{\mathsf{HPS}}$.

Lemma 5 (ϵ-MU-SIM Security of $\mathsf{KEM}_{\mathsf{HPS}}$)**.** *Let* ϵ' *be a real number and* Π *the hash value space of* HPS*. If* HPS *is an* ϵ'*-universal*$_2$ *HPS associated with a hard multi-fold SMP, then* $\mathsf{KEM}_{\mathsf{HPS}}$ *is* ϵ*-MU-SIM secure with uniformity parameter* $\epsilon = |\Pi|^2 \cdot (\epsilon' - 1/|\Pi|)$*.*

Concretely, for any polynomial μ*, any adversary* $\mathcal{A}$*, there exists an adversary* $\mathcal{B}$*, such that* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *and*

$$\mathsf{Adv}^{\mathsf{mu\text{-}sim}}_{\mathsf{KEM},\mathsf{Encap}^*,\mu}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{msmp}}_{\mathsf{HPS},\mu}(\mathcal{B}). \tag{7}$$

Remark 5. If HPS is perfectly universal$_2$ (i.e., $\epsilon' = 1/|\Pi|$), then $\mathsf{KEM}_{\mathsf{HPS}}$ has 0-uniformity for the key encapsulated by Encap^* (i.e, $\epsilon = 0$).

Proof of Lemma 5. To prove (7), we build an adversary $\mathcal{B}$ solving the multi-fold SMP by invoking $\mathcal{A}$. Given a challenge $(\mathsf{pp}, \{x_i\}_{i\in[\mu]})$, $\mathcal{B}$ wants to determine whether $x_i \leftarrow_\$ \mathcal{L}$ or $x_i \leftarrow_\$ \mathcal{X} \setminus \mathcal{L}$. $\mathcal{B}$ sets $\mathsf{pp}_{\mathsf{KEM}} := \mathsf{pp}$, samples $(pk_i, sk_i) \leftarrow_\$ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$, and computes $(c_i, K_i) := (x_i, \Lambda_{sk_i}(x_i))$. Then $\mathcal{B}$ invokes $\mathcal{A}\big(\{pk_i, sk_i, c_i, K_i\}_{i\in[\mu]}\big)$, and returns the output of $\mathcal{A}$ to its challenger. If $x_i \leftarrow_\$ \mathcal{L}$, $(c_i, K_i) = (x_i, \Lambda_{sk_i}(x_i))$ has the same distribution as the output of $\mathsf{Encap}(pk_i)$; if $x_i \leftarrow_\$ \mathcal{X} \setminus \mathcal{L}$, $(c_i, K_i) = (x_i, \Lambda_{sk_i}(x_i))$ has the same distribution as the output of $\mathsf{Encap}^*(sk_i)$. Consequently, $\mathcal{B}$ solves the multi-fold SMP as long as $\mathcal{A}$ distinguishes Encap and Encap^*, and (7) holds.

We now proceed to prove ϵ-Uniformity of Encap^* as defined in (1). Firstly, ϵ'-universal$_2$ of HPS means

$$\Pr[\Lambda_{sk}(c^*) = \pi^* \mid \alpha(sk) = pk, \Lambda_{sk}(c) = \pi] \le \epsilon',$$

for all $\mathsf{pp} \leftarrow_\$ \mathsf{HPS.Setup}$, all $pk \in \mathcal{PK}$, all $c, c^* \in \mathcal{X}$ with $c^* \notin \mathcal{L} \cup \{c\}$, and all $\pi, \pi^* \in \Pi$, where the probability is over $sk \leftarrow_\$ \mathcal{SK}$. By an averaging argument over (c, c^*, pk), ϵ'-universal$_2$ implies that for any (unbounded) adversary $\mathcal{B}$, it holds that

$$\begin{aligned} &\big|\ \Pr[c \leftarrow_\$ \mathcal{B}(pk, c^*) \ :\ c \neq c^* \wedge \mathcal{B}(pk, c^*, K^*, \mathsf{Decap}(sk, c)) \Rightarrow 1] \\ &\quad - \ \Pr[c \leftarrow_\$ \mathcal{B}(pk, c^*) \ :\ c \neq c^* \wedge \mathcal{B}(pk, c^*, R, \mathsf{Decap}(sk, c)) \Rightarrow 1]\ \big| \\ &\le\ |\Pi| \cdot (\epsilon' - 1/|\Pi|), \end{aligned} \tag{8}$$

where the probability is over $\mathsf{pp}_{\mathsf{KEM}} \leftarrow_\$ \mathsf{KEM.Setup}$, $(pk, sk) \leftarrow_\$ \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}})$, $(c^*, K^*) \leftarrow_\$ \mathsf{Encap}^*(sk)$, $R \leftarrow_\$ \mathcal{K}$ and the internal randomness of $\mathcal{B}$. The averaging argument essentially uses the law of total probability over (c, c^*, pk). We note that here c is not allowed to depend on K^* or R, but ϵ-Uniformity allows c arbitrarily dependent on K^* or R. This gap can be filled by a leveraging argument, as shown below.

Suppose towards a contradiction that there exists an (unbounded) adversary $\mathcal{A}$, so that

$$\begin{aligned} &\big|\ \Pr[c \leftarrow_\$ \mathcal{A}(pk, c^*, K^*) \ :\ c \neq c^* \wedge \mathcal{A}(pk, c^*, K^*, \mathsf{Decap}(sk, c)) \Rightarrow 1] \\ &-\ \Pr[c \leftarrow_\$ \mathcal{A}(pk, c^*, R) \ :\ c \neq c^* \wedge \mathcal{A}(pk, c^*, R, \mathsf{Decap}(sk, c)) \Rightarrow 1]\ \big| > \epsilon. \end{aligned} \tag{9}$$

Then we can construct an (unbounded) adversary $\mathcal{B}$ to contradict with (8). $\mathcal{B}$ is constructed by the following leveraging argument.

- Given (pk, c^*), $\mathcal{B}$ samples a $T' \leftarrow_\$ \Pi$ uniformly, invokes $c \leftarrow_\$ \mathcal{A}(pk, c^*, T')$ and outputs c.
- Then $\mathcal{B}$ receives $(pk, c^*, T, \mathsf{Decap}(sk, c))$, where $T = K^*$ or $T = R$, and invokes $b \leftarrow_\$ \mathcal{A}(pk, c^*, T, \mathsf{Decap}(sk, c))$.
- If $T' = T$, $\mathcal{B}$ outputs the bit b output by $\mathcal{A}$; otherwise, $\mathcal{B}$ outputs 0.

If $T' = T$, which happens with probability $1/|\Pi|$, $\mathcal{B}$ perfectly simulates the experiment defined in (9) for $\mathcal{A}$, thus $\mathcal{B}$ distinguishes $T = K^*$ from $T = R$ as long as $\mathcal{A}$ does; if

$T' \neq T$, $\mathcal{B}$ always outputs 0 no matter $T = K^*$ or $T = R$. Overall,

$$\begin{aligned} & \mathcal{B}\text{'s distinguishing advantage in the experiment defined in (8)} \\ = \; & \mathcal{A}\text{'s distinguishing advantage in the experiment defined in (9)} \; /|\Pi| \\ > \; & \epsilon/|\Pi| = |\Pi| \cdot (\epsilon' - 1/|\Pi|), \end{aligned}$$

which contradicts with (8). This completes the proof of ϵ-Uniformity. □

Moreover, we show the diversity of $\mathsf{KEM}_{\mathsf{HPS}}$ as long as HPS is extracting.

Lemma 6 (γ-Diversity of $\mathsf{KEM}_{\mathsf{HPS}}$)**.** *Let γ' be a real number, Π the hash value space of HPS and $\mathcal{L}$ the language space. If HPS is γ'-extracting, then $\mathsf{KEM}_{\mathsf{HPS}}$ has γ-diversity with $\gamma = 2\gamma' + \log|\mathcal{L}| - \log(|\Pi| \cdot |\mathcal{L}| + 2^{2\gamma'})$.*

Remark 6. If HPS is perfectly extracting (i.e., $\gamma' = \log|\Pi|$), then $\mathsf{KEM}_{\mathsf{HPS}}$ is γ-diverse with $\gamma = \log|\Pi| + \log|\mathcal{L}| - \log(|\Pi| + |\mathcal{L}|)$. For our concrete instantiation $\mathsf{HPS}_{\mathsf{MDDH}}$ shown in Section 7.3, which is perfectly extracting and has $|\Pi| = q$ and $|\mathcal{L}| = q^k - 1$, the resulting $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ is γ-diverse with $\gamma = \log q + \log(q^k - 1) - \log(q + q^k - 1) \geq \log(q/3)$.

Proof of Lemma 6. By construction, we have

$$\begin{aligned}
(1) \quad & \Pr\left[\begin{array}{c}(pk, sk) \leftarrow_{\$} \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}); \\ r, r' \leftarrow_{\$} \mathcal{R}; (c, K) \leftarrow \mathsf{Encap}(pk; r); (c', K') \leftarrow \mathsf{Encap}(pk; r')\end{array} : K = K'\right] \\
= \; & \Pr\left[sk \leftarrow_{\$} \mathcal{SK}; x, x' \leftarrow_{\$} \mathcal{L} : \Lambda_{sk}(x) = \Lambda_{sk}(x')\right] \\
= \; & \sum_{a \in \mathcal{L}} \Pr_{x \leftarrow_{\$} \mathcal{L}}[x = a] \cdot \sum_{a' \in \mathcal{L}} \Pr_{x' \leftarrow_{\$} \mathcal{L}}[x' = a'] \cdot \Pr_{sk \leftarrow_{\$} \mathcal{SK}}\left[\Lambda_{sk}(a) = \Lambda_{sk}(a')\right] \\
= \; & \sum_{a \in \mathcal{L}} \frac{1}{|\mathcal{L}|} \cdot \left(\sum_{a' \in \mathcal{L} \setminus \{a\}} \frac{1}{|\mathcal{L}|} \cdot \sum_{\pi \in \Pi} \Pr_{sk \leftarrow_{\$} \mathcal{SK}}\left[\Lambda_{sk}(a) = \Lambda_{sk}(a') = \pi\right] + \frac{1}{|\mathcal{L}|} \cdot 1 \right) \\
= \; & \sum_{a \in \mathcal{L}} \frac{1}{|\mathcal{L}|} \cdot \left(\sum_{a' \in \mathcal{L} \setminus \{a\}} \frac{1}{|\mathcal{L}|} \cdot \sum_{\pi \in \Pi} \underbrace{\Pr_{sk \leftarrow_{\$} \mathcal{SK}}\left[\Lambda_{sk}(a) = \pi\right]}_{\leq 2^{-\gamma'} \text{ by } \gamma'\text{-extracting}_1} \cdot \underbrace{\Pr_{sk \leftarrow_{\$} \mathcal{SK}}\left[\Lambda_{sk}(a') = \pi \,|\, \Lambda_{sk}(a) = \pi\right]}_{\leq 2^{-\gamma'} \text{ by } \gamma'\text{-extracting}_2 \text{ of } \mathsf{HPS}} + \frac{1}{|\mathcal{L}|} \right) \\
\leq \; & |\Pi| \cdot (2^{-\gamma'})^2 + 1/|\mathcal{L}|, \\
(2) \quad & \Pr\left[\begin{array}{c}(pk, sk) \leftarrow_{\$} \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}); (pk', sk') \leftarrow_{\$} \mathsf{KEM.Gen}(\mathsf{pp}_{\mathsf{KEM}}); \\ r \leftarrow_{\$} \mathcal{R}; (c, K) \leftarrow \mathsf{Encap}(pk; r); (c', K') \leftarrow \mathsf{Encap}(pk'; r)\end{array} : K = K'\right] \\
= \; & \Pr\left[sk, sk' \leftarrow_{\$} \mathcal{SK}; x \leftarrow_{\$} \mathcal{L} : \Lambda_{sk}(x) = \Lambda_{sk'}(x)\right] \\
= \; & \sum_{a \in \mathcal{L}} \Pr_{x \leftarrow_{\$} \mathcal{L}}[x = a] \cdot \sum_{\pi \in \Pi} \Pr\left[sk, sk' \leftarrow_{\$} \mathcal{SK} : \Lambda_{sk}(a) = \Lambda_{sk'}(a) = \pi\right] \\
= \; & \sum_{a \in \mathcal{L}} \frac{1}{|\mathcal{L}|} \cdot \sum_{\pi \in \Pi} \underbrace{\Pr_{sk \leftarrow_{\$} \mathcal{SK}}[\Lambda_{sk}(a) = \pi]}_{\leq 2^{-\gamma'} \text{ by } \gamma'\text{-extracting}_1 \text{ of } \mathsf{HPS}} \cdot \underbrace{\Pr_{sk' \leftarrow_{\$} \mathcal{SK}}[\Lambda_{sk'}(a) = \pi]}_{\leq 2^{-\gamma'} \text{ by } \gamma'\text{-extracting}_1 \text{ of } \mathsf{HPS}} \\
\leq \; & |\Pi| \cdot (2^{-\gamma'})^2.
\end{aligned}$$

Thus, $\mathsf{KEM}_{\mathsf{HPS}}$ has γ-diversity with $\gamma = 2\gamma' + \log|\mathcal{L}| - \log(|\Pi| \cdot |\mathcal{L}| + 2^{2\gamma'})$. □

7.3 Universal$_2$ Hash Proof System from MDDH

In this subsection, we construct an MDDH-based universal$_2$ hash proof system $\mathsf{HPS}_{\mathsf{MDDH}}$ which is also extracting. Then together with the transformation in Section 7.2, we immediately obtain an MDDH-based ϵ-MU-SIM secure KEM which also enjoys γ-diversity.

Our $\mathsf{HPS}_{\mathsf{MDDH}}$ extends the DDH-based hash proof system proposed by Cramer and Shoup in [CS02] to MDDH assumptions. Let $\mathcal{G} = (\mathbb{G}, q, \mathcal{P})$ be a description of cyclic group $\mathbb{G}$ of prime order q and with generator $\mathcal{P}$. Let $\mathcal{D}_k$ be a matrix distribution, $t \in \mathbb{N}$, and $\mathcal{H} = \{\mathsf{H} : \mathbb{G}^{k+1} \to \mathbb{Z}_q^t\}$ a family of hash functions from $\mathbb{G}^{k+1}$ to $\mathbb{Z}_q^t$.

- $\mathsf{HPS.Setup}$ picks $\mathbf{A} \leftarrow_\$ \mathcal{D}_k$, $\mathsf{H} \leftarrow_\$ \mathcal{H}$, and outputs public parameter $\mathsf{pp} := ([\mathbf{A}], \mathsf{H})$. pp implicitly defines $(\mathcal{L}, \mathcal{X}, \mathcal{SK}, \mathcal{PK}, \Pi, \Lambda_{(\cdot)}, \alpha)$ as follows.
  - $\mathcal{X} := \mathbb{G}^{k+1} \setminus \{[\mathbf{0}]\}$ and the language $\mathcal{L} := \mathcal{L}_{\mathbf{A}} := \{[\mathbf{c}] = [\mathbf{Aw}] \in \mathbb{G}^{k+1} : \mathbf{w} \in \mathbb{Z}_q^k \setminus \{\mathbf{0}\}\} \subseteq \mathcal{X}$. The value $\mathbf{w}$ is a witness of $[\mathbf{c}] \in \mathcal{L}$.
  - $\mathcal{SK} := \mathbb{Z}_q^{(t+1)\times(k+1)}$, $\mathcal{PK} := \mathbb{G}^{(t+1)\times k}$, and hash value space $\Pi := \mathbb{G}$.
  - For $sk = \mathbf{K} \in \mathcal{SK}$, define $pk = \alpha(sk) := [\mathbf{KA}] \in \mathcal{PK}$.
  - For $[\mathbf{c}] \in \mathcal{X}$ and $sk = \mathbf{K} \in \mathcal{SK}$, define $\Lambda_{sk}([\mathbf{c}]) := (1, \boldsymbol{\tau}^\top) \cdot \mathbf{K} \cdot [\mathbf{c}] \in \mathbb{G}$ with $\boldsymbol{\tau} := \mathsf{H}([\mathbf{c}]) \in \mathbb{Z}_q^t$.
- $\mathsf{Pub}(pk = [\mathbf{KA}], [\mathbf{c}] \in \mathcal{L}, \mathbf{w} \in \mathbb{Z}_q^k)$: Compute $\boldsymbol{\tau} := \mathsf{H}([\mathbf{c}]) \in \mathbb{Z}_q^t$, and return $[\pi] := (1, \boldsymbol{\tau}^\top) \cdot [\mathbf{KA}] \cdot \mathbf{w} \in \mathbb{G}$.
- $\mathsf{Priv}(sk = \mathbf{K}, [\mathbf{c}] \in \mathcal{X})$: Compute $\boldsymbol{\tau} := \mathsf{H}([\mathbf{c}]) \in \mathbb{Z}_q^t$, and return $[\pi] := (1, \boldsymbol{\tau}^\top) \cdot \mathbf{K} \cdot [\mathbf{c}] \in \mathbb{G}$.

It is straightforward to check the correctness of $\mathsf{HPS}_{\mathsf{MDDH}}$. The associated SMP is exactly the $\mathcal{D}_k$-MDDH (cf. Definition 17), and the associated multi-fold SMP is exactly multi-fold $\mathcal{D}_k$-MDDH. By the random self-reducibility of $\mathcal{D}_k$-MDDH (cf. Lemma 10), we have the following corollary.

Corollary 4 (Multi-fold SMP)**.** *For any $\mu \in \mathbb{N}$ and any $\mathcal{A}$ there exists an adversary $\mathcal{B}$ with $\mathsf{Adv}^{\mathsf{msmp}}_{\mathsf{HPS}_{\mathsf{MDDH}},\mu}(\mathcal{A}) = \mathsf{Adv}^{\mu\text{-MDDH}}_{\mathsf{GGen},\mathcal{D}_k,\mathbb{G}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{D}_k,\mathbb{G}}(\mathcal{B}) + \frac{1}{q-1}$.*

Below we show the perfect universal$_2$ and extracting properties, respectively.

Lemma 7 (Perfectly Universal$_2$ of $\mathsf{HPS}_{\mathsf{MDDH}}$)**.** *(1) If $\mathcal{H} = \mathcal{H}_{\mathsf{inj}} = \{\mathsf{H} : \mathbb{G}^{k+1} \to \mathbb{Z}_q^t\}$ is a family of injective functions (in this case $t \geq k+1$), then $\mathsf{HPS}_{\mathsf{MDDH}}$ is perfectly universal$_2$. (2) If $\mathcal{H} = \mathcal{H}_{\mathsf{cr}} = \{\mathsf{H} : \mathbb{G}^{k+1} \to \mathbb{Z}_q^t\}$ is a family of collision-resistant hash functions (in this case $t = 1$), then $\mathsf{HPS}_{\mathsf{MDDH}}$ is perfectly universal$_2$ assuming that there are no collisions.*

Proof of Lemma 7. For $sk = \mathbf{K} \leftarrow_\$ \mathbb{Z}_q^{(t+1)\times(k+1)}$, for any $[\mathbf{c}] \in \mathcal{X}$, any $[\mathbf{c}^*] \in \mathcal{X} \setminus (\mathcal{L} \cup \{[\mathbf{c}]\})$, any $pk \in \mathcal{PK}$ and any $[\pi] \in \mathbb{G}$, we consider the distribution of $\Lambda_{sk}([\mathbf{c}^*]) = (1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{K} \cdot [\mathbf{c}^*]$ conditioned on $pk = \alpha(sk) = [\mathbf{KA}]$ and $[\pi] = \Lambda_{sk}([\mathbf{c}]) = (1, \boldsymbol{\tau}^\top) \cdot \mathbf{K} \cdot [\mathbf{c}]$, where $\boldsymbol{\tau}^* := \mathsf{H}([\mathbf{c}^*]), \boldsymbol{\tau} := \mathsf{H}([\mathbf{c}]) \in \mathbb{Z}_q^t$.

Firstly, we prove (1). Since $[\mathbf{c}^*] \neq [\mathbf{c}]$ and H is injective, we have $\boldsymbol{\tau}^* \neq \boldsymbol{\tau}$. Let $\mathbf{a}^\perp \in \mathbb{Z}_q^{k+1}$ be an arbitrary non-zero vector in the kernel space of $\mathbf{A}$ such that $(\mathbf{a}^\perp)^\top \mathbf{A} = \mathbf{0}$ holds. It is clear that $(\mathbf{a}^\perp)^\top \cdot [\mathbf{c}^*] \neq [0]$ since $[\mathbf{c}^*] \notin \mathsf{span}([\mathbf{A}])$. Let $\mathbf{b} \in \mathbb{Z}_q^{t+1}$ be an arbitrary non-zero vector such that $(1, \boldsymbol{\tau}^\top) \cdot \mathbf{b} = 0$ but $(1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{b} = 1$. We can always find such $\mathbf{b}$ since $(1, \boldsymbol{\tau}^\top)$ and $(1, \boldsymbol{\tau}^{*\top})$ are linearly independent (due to $\boldsymbol{\tau}^* \neq \boldsymbol{\tau}$). Equivalently, sk can be sampled via $sk = \mathbf{K} := \widetilde{\mathbf{K}} + \mu \cdot \mathbf{b}(\mathbf{a}^\perp)^\top \in \mathbb{Z}_q^{(t+1)\times(k+1)}$, where $\widetilde{\mathbf{K}} \leftarrow_\$ \mathbb{Z}_q^{(t+1)\times(k+1)}$ and $\mu \leftarrow_\$ \mathbb{Z}_q$. In this case, we have $pk = \alpha(sk) = [\mathbf{K}\mathbf{A}] = [\widetilde{\mathbf{K}}\mathbf{A}]$, $[\pi] = \Lambda_{sk}([\mathbf{c}]) = (1, \boldsymbol{\tau}^\top) \cdot \mathbf{K} \cdot [\mathbf{c}] = (1, \boldsymbol{\tau}^\top) \cdot \widetilde{\mathbf{K}} \cdot [\mathbf{c}]$, which may leak $\widetilde{\mathbf{K}}$, but the value of μ is completely hidden. Moreover,

$$\Lambda_{sk}([\mathbf{c}^*]) = (1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{K} \cdot [\mathbf{c}^*] = (1, \boldsymbol{\tau}^{*\top}) \cdot \widetilde{\mathbf{K}} \cdot [\mathbf{c}^*] + \mu \cdot \underbrace{(1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{b}}_{=1} \cdot \underbrace{(\mathbf{a}^\perp)^\top \cdot [\mathbf{c}^*]}_{\neq [0]}.$$

Thanks to the uniformity of μ, $\Lambda_{sk}([\mathbf{c}^*])$ is uniformly distributed over $\mathbb{G}$ conditioned on $pk = \alpha(sk)$ and $[\pi] = \Lambda_{sk}([\mathbf{c}])$. This shows that $\mathsf{HPS}_{\mathsf{MDDH}}$ is perfectly universal$_2$.

For (2), we also have $\boldsymbol{\tau}^* \neq \boldsymbol{\tau}$ as long as no collision happens. Then the analysis of perfectly universal$_2$ is similar to (1). □

Remark 7. Instantiating $\mathsf{HPS}_{\mathsf{MDDH}}$ using $\mathcal{H}_{\mathsf{cr}}$ is more efficient than using $\mathcal{H}_{\mathsf{inj}}$ since t can be as small as 1. Taking into account the collision resistance of $\mathcal{H}_{\mathsf{cr}}$, $\mathsf{HPS}_{\mathsf{MDDH}}$ using $\mathcal{H}_{\mathsf{cr}}$ is perfectly universal$_2$ only in a computational sense, and the $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ derived from such $\mathsf{HPS}_{\mathsf{MDDH}}$ (cf. Section 7.2) has only computational ϵ-uniformity. Nevertheless, this does not affect the tight security reduction from $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ to $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ in Theorem 1 and $\mathsf{AKE}_{\mathsf{3msg}}$ to $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ in Theorem 2, since we can always add an extra game (e.g., between G_0 and G_1) to deal with collisions in the security proofs. In this extra game, the challenger aborts immediately when collision happens. Then in subsequent games, the adversary wins only if no collision happens, and in such scenarios, $\mathsf{HPS}_{\mathsf{MDDH}}$ using $\mathcal{H}_{\mathsf{cr}}$ is perfect universal$_2$ and the $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ derived from $\mathsf{HPS}_{\mathsf{MDDH}}$ (cf. Section 7.2) has ϵ-uniformity. On the other hand, we note that it is easy to construct hash functions $\mathcal{H}_{\mathsf{cr}}$ whose collision resistance can be tightly reduced to the discrete logarithm assumption [Dam88, CvP92], so this extra game does not affect the tightness of our AKE protocols.

Lemma 8 (Perfect Extracting of $\mathsf{HPS}_{\mathsf{MDDH}}$)**.** $\mathsf{HPS}_{\mathsf{MDDH}}$ *is perfectly extracting*$_1$*. Moreover, (1) If* $\mathcal{H} = \mathcal{H}_{\mathsf{inj}} = \{\mathsf{H} : \mathbb{G}^{k+1} \to \mathbb{Z}_q^t\}$ *is a family of injective functions (in this case* $t \geq k+1$*), then* $\mathsf{HPS}_{\mathsf{MDDH}}$ *is perfectly extracting*$_2$*. (2) If* $\mathcal{H} = \mathcal{H}_{\mathsf{cr}} = \{\mathsf{H} : \mathbb{G}^{k+1} \to \mathbb{Z}_q^t\}$ *is a family of collision-resistant hash functions (in this case* $t = 1$*), then* $\mathsf{HPS}_{\mathsf{MDDH}}$ *is perfectly extracting*$_2$ *assuming that there are no collisions.*

Proof of Lemma 8. Firstly, we prove the perfect extracting$_1$ property. For $sk = \mathbf{K} \leftarrow_\$ \mathbb{Z}_q^{(t+1)\times(k+1)}$, for any $[\mathbf{c}] \in \mathcal{L} = \mathsf{span}[\mathbf{A}] \setminus \{[\mathbf{0}]\}$, we consider the distribution of $\Lambda_{sk}([\mathbf{c}]) = (1, \boldsymbol{\tau}^\top) \cdot \mathbf{K} \cdot [\mathbf{c}]$, where $\boldsymbol{\tau} := \mathsf{H}([\mathbf{c}]) \in \mathbb{Z}_q^t$. Since $[\mathbf{c}] \neq [\mathbf{0}]$ and $\mathbf{K} \leftarrow_\$ \mathbb{Z}_q^{(t+1)\times(k+1)}$, it follows that $\mathbf{K} \cdot [\mathbf{c}]$ is uniformly distributed over $\mathbb{G}^{t+1}$. Moreover, $(1, \boldsymbol{\tau}^\top)$ is non-zero, thus $\Lambda_{sk}([\mathbf{c}]) = (1, \boldsymbol{\tau}^\top) \cdot \mathbf{K} \cdot [\mathbf{c}]$ is uniformly distributed over $\mathbb{G}$. This shows the perfect extracting$_1$ of $\mathsf{HPS}_{\mathsf{MDDH}}$.

Next, we prove the perfect extracting$_2$ property. For $sk = \mathbf{K} \leftarrow_\$ \mathbb{Z}_q^{(t+1)\times(k+1)}$, for any $[\mathbf{c}] \in \mathcal{L}$, any $[\mathbf{c}^*] \in \mathcal{L} \setminus \{[\mathbf{c}]\}$, and any $[\pi] \in \mathbb{G}$, we consider the distribution of

$\Lambda_{sk}([\mathbf{c}^*]) = (1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{K} \cdot [\mathbf{c}^*]$ conditioned on $[\pi] = \Lambda_{sk}([\mathbf{c}]) = (1, \boldsymbol{\tau}^{\top}) \cdot \mathbf{K} \cdot [\mathbf{c}]$, where $\boldsymbol{\tau}^* := \mathsf{H}([\mathbf{c}^*]), \boldsymbol{\tau} := \mathsf{H}([\mathbf{c}]) \in \mathbb{Z}_q^t$.

- For (1), since H is injective, we have $\boldsymbol{\tau}^* \neq \boldsymbol{\tau}$. Let $\mathbf{b} \in \mathbb{Z}_q^{t+1}$ be an arbitrary non-zero vector such that $(1, \boldsymbol{\tau}^{\top}) \cdot \mathbf{b} = 0$ but $(1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{b} = 1$. We can always find such $\mathbf{b}$ since $(1, \boldsymbol{\tau}^{\top})$ and $(1, \boldsymbol{\tau}^{*\top})$ are linearly independent (due to $\boldsymbol{\tau}^* \neq \boldsymbol{\tau}$). Equivalently, sk can be sampled via $sk = \mathbf{K} := \widetilde{\mathbf{K}} + \mathbf{b} \cdot \mathbf{r}^{\top} \in \mathbb{Z}_q^{(t+1)\times(k+1)}$, where $\widetilde{\mathbf{K}} \leftarrow_{\$} \mathbb{Z}_q^{(t+1)\times(k+1)}$ and $\mathbf{r} \leftarrow_{\$} \mathbb{Z}_q^{k+1}$. In this case, we have $[\pi] = \Lambda_{sk}([\mathbf{c}]) = (1, \boldsymbol{\tau}^{\top}) \cdot \mathbf{K} \cdot [\mathbf{c}] = (1, \boldsymbol{\tau}^{\top}) \cdot \widetilde{\mathbf{K}} \cdot [\mathbf{c}]$, which may leak $\widetilde{\mathbf{K}}$, but the value of $\mathbf{r}$ is completely hidden. Moreover,

$$\Lambda_{sk}([\mathbf{c}^*]) = (1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{K} \cdot [\mathbf{c}^*] = (1, \boldsymbol{\tau}^{*\top}) \cdot \widetilde{\mathbf{K}} \cdot [\mathbf{c}^*] + \underbrace{(1, \boldsymbol{\tau}^{*\top}) \cdot \mathbf{b}}_{=1} \cdot \mathbf{r}^{\top} \cdot [\mathbf{c}^*].$$

  Thanks to the uniformity of $\mathbf{r}$ and the fact that $[\mathbf{c}^*] \neq [\mathbf{0}]$, $\mathbf{r}^{\top} \cdot [\mathbf{c}^*]$ is uniformly distributed over $\mathbb{G}$, and this implies that uniformity of $\Lambda_{sk}([\mathbf{c}^*])$ conditioned on $[\pi] = \Lambda_{sk}([\mathbf{c}])$. Hence the perfectly extracting$_2$ of $\mathsf{HPS}_{\mathsf{MDDH}}$ follows.
- For (2), we also have $\boldsymbol{\tau}^* \neq \boldsymbol{\tau}$ as long as no collision happens. Then the analysis of perfect extracting$_2$ is similar to (1). □

Remark 8. As in Remark 7, $\mathsf{HPS}_{\mathsf{MDDH}}$ using $\mathcal{H}_{\mathsf{cr}}$ is perfectly extracting$_2$ only in a computational sense, and the $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ derived from such $\mathsf{HPS}_{\mathsf{MDDH}}$ (cf. Section 7.2) has only computational diversity. Nevertheless, this does not affect the tight security reduction from $\mathsf{AKE}^{\mathsf{state}}_{\mathsf{3msg}}$ to $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ in Theorem 1 and $\mathsf{AKE}_{\mathsf{3msg}}$ to $\mathsf{KEM}_{\mathsf{HPS}_{\mathsf{MDDH}}}$ in Theorem 2, since we can always add an extra game to deal with collisions in the security proofs.

Acknowledgments. We would like to thank the reviewers for their helpful comments. Shuai Han and Shengli Liu were partially supported by National Natural Science Foundation of China (Grant Nos. 61925207, 62002223), Guangdong Major Project of Basic and Applied Basic Research (2019B030302008), Shanghai Sailing Program (20YF1421100), Young Elite Scientists Sponsorship Program by China Association for Science and Technology, and the National Key Research and Development Project 2020YFA0712300. Tibor Jager was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823. Eike Kiltz was supported by the BMBF iBlockchain project, the EU H2020 PROMETHEUS project 780701, DFG SPP 1736 Big Data, and the DFG Cluster of Excellence 2092 CASA. Doreen Riepel was supported by the Deutsche Forschungsgemeinschaft (DFG) Cluster of Excellence 2092 CASA. Sven Schäge was supported by the German Federal Ministry of Education and Research (BMBF), Project DigiSeal (16KIS0695) and Huawei Technologies Düsseldorf, Project vHSM.

References

[Bad14] Christoph Bader. Efficient signatures with tight real world security in the random-oracle model. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G.

Askoxylakis, editors, *CANS 14*, volume 8813 of *LNCS*, pages 370–383. Springer, Heidelberg, October 2014. 175

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658. Springer, Heidelberg, March 2015. 174, 175, 178, 182, 185, 207

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304. Springer, Heidelberg, May 2016. 175, 176

[BKP14] Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (Hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 408–425. Springer, Heidelberg, August 2014. 175, 207

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 174

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994. 174, 185

[CF12] Cas J. F. Cremers and Michele Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751. Springer, Heidelberg, September 2012. 176

[CHH+07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. Bounded CCA2-secure encryption. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 502–518. Springer, Heidelberg, December 2007. 177

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001. 174, 176, 178

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002. 177, 213, 217

[CvP92] David Chaum, Eugène van Heijst, and Birgit Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 470–484. Springer, Heidelberg, August 1992. 218

[Dam88] Ivan Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *EUROCRYPT'87*, volume 304 of *LNCS*, pages 203–216. Springer, Heidelberg, April 1988. 218

[DG20] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029, 2020. https://eprint.iacr.org/2020/1029. 174, 176

[DGJL21] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. More efficient digital signatures with tight multi-user security. 24th International Conference on Practice and Theory of Public-Key Cryptography, PKC 2021, 2021. 175

[DJ20] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Cryptology ePrint Archive, Report 2020/726, 2020. https://eprint.iacr.org/2020/726. 174, 176

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 355–374. Springer, Heidelberg, April 2012. 214

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013. 205, 206, 228

[EHK+17] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Luis Villar. An algebraic framework for Diffie-Hellman assumptions. *Journal of Cryptology*, 30(1):242–288, January 2017. 175, 177

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 467–484. Springer, Heidelberg, May 2012. 179

[GHK17] Romain Gay, Dennis Hofheinz, and Lisa Kohl. Kurosawa-desmedt meets tight security. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 133–160. Springer, Heidelberg, August 2017. 177

[GHKW16] Romain Gay, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Tightly CCA-secure encryption without pairings. In Marc Fischlin and Jean-Sébastien

Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 1–27. Springer, Heidelberg, May 2016. 177

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 95–125. Springer, Heidelberg, August 2018. 174, 175, 176, 185

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990. 190

[HLLG19] Shuai Han, Shengli Liu, Lin Lyu, and Dawu Gu. Tight leakage-resilient CCA-security from quasi-adaptive hash proof system. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 417–447. Springer, Heidelberg, August 2019. 177, 212

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2021, 2021. 174, 176, 178, 179, 185

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Heidelberg, August 2012. 179

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005. 190

[LLGW20] Xiangyu Liu, Shengli Liu, Dawu Gu, and Jian Weng. Two-pass authenticated key exchange with explicit authentication and tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 785–814. Springer, Heidelberg, December 2020. 174, 176, 177, 178, 183, 185, 189, 212

[LP19] Roman Langrehr and Jiaxin Pan. Tightly secure hierarchical identity-based encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 436–465. Springer, Heidelberg, April 2019. 175, 207, 208

[LP20] Roman Langrehr and Jiaxin Pan. Unbounded HIBE with tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 129–159. Springer, Heidelberg, December 2020. 175, 229

[LS17] Yong Li and Sven Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1343–1360. ACM Press, October / November 2017. 177, 185, 188

[MPS20] Andrew Morgan, Rafael Pass, and Elaine Shi. On the adaptive security of MACs and PRFs. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 724–753. Springer, Heidelberg, December 2020. 176

[MRV16] Paz Morillo, Carla Ràfols, and Jorge Luis Villar. The kernel matrix Diffie-Hellman assumption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 729–758. Springer, Heidelberg, December 2016. 206

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002. 174

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997. 183

A Proof of Theorem 3

Let us first define message-consistency for the 2-move protocol $\mathsf{AKE}_{\mathsf{2msg}}$ in Figure 6.

Message Consistency. We say that an oracle π_i^s is *message-consistent* with another oracle π_j^t, denoted by $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$, if $\mathsf{Pid}_i^s := j$ and $\mathsf{Pid}_j^t := i$ and either

(1) π_i^s has sent the first message, the same ephemeral public key $\hat{pk}$ is contained in Sent_i^s and Recv_j^t and the same ciphertext c is contained in Recv_i^s and Sent_j^t, or
(2) π_i^s has received the first message and the same ephemeral public key $\hat{pk}$ is contained in Recv_i^s and Sent_j^t.

We write $\mathsf{MsgCon}(\pi_i^s \leftrightarrow \pi_j^t)$ if $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ and $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s)$.

We now define a sequence of games G_0-G_2. Let Win_i denote the probability that G_i returns 1.

Game G_0: G_0 is the original experiment $\mathsf{Exp}_{\mathsf{AKE}_{\mathsf{2msg}},\mu,\ell,\mathcal{A}}$. In addition to the original game, we add the sets Sent_i^s and Recv_i^s which is only a conceptual change. We have

$$\Pr[\mathsf{Exp}_{\mathsf{AKE}_{\mathsf{2msg}},\mu,\ell,\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{Win}_0] \ .$$

Game G_1: In G_1, we define the event $\mathsf{NoMsgCon}$ which happens for (i,s) if π_i^s accepts, the intended partner $j := \mathsf{Pid}_i^s$ is uncorrupted when π_i^s accepts and there does not exist

$t \in [\ell]$ such that π_i^s is message-consistent with π_j^t. If event $\mathsf{NoMsgCon}$ happens, the game will abort. Due to the difference lemma,

$$|\Pr[\mathsf{Win}_0] - \Pr[\mathsf{Win}_1]| \leq \Pr[\mathsf{NoMsgCon}] \ .$$

We will prove the following lemma.

Lemma 9. *There exists an adversary $\mathcal{B}_{\mathsf{SIG}}$ against SIG such that*

$$\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] \leq \Pr[\mathsf{NoMsgCon}] \leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}).$$

Proof. If there exists an oracle π_j^t such that π_i^s is message-consistent with π_j^t and $\mathsf{Pid}_j^t = i$, then due to correctness of KEM, π_i^s is also partnered to π_j^t. It follows that $\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] \leq \Pr[\mathsf{NoMsgCon}]$.

To prove that $\Pr[\mathsf{NoMsgCon}] \leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}})$, we construct adversary $\mathcal{B}_{\mathsf{SIG}}$ against MU-EUF-CMA$^{\mathsf{corr}}$ security of SIG. $\mathcal{B}_{\mathsf{SIG}}$ inputs the public parameter $\mathsf{pp}_{\mathsf{SIG}}$ and a list of verification keys $\{vk_i\}_{i\in[\mu]}$ and has access to a signing oracle $\mathcal{O}_{\text{SIGN}}(\cdot,\cdot)$ and a corrupt oracle $\mathcal{O}_{\text{CORR}}(\cdot)$. $\mathcal{B}_{\mathsf{SIG}}$ then runs $\mathsf{pp}_{\mathsf{KEM}} \leftarrow\!\!\$\ \mathsf{KEM.Setup}$ and sets $\mathsf{pp}_{\mathsf{AKE}} := (\mathsf{pp}_{\mathsf{SIG}}, \mathsf{pp}_{\mathsf{KEM}})$ and $\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$. It initializes all variables and then runs $\mathcal{A}$ on $\mathsf{pp}_{\mathsf{AKE}}$ and PKList. If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{AKE}}$, $\mathcal{B}_{\mathsf{SIG}}$ responds as follows.

- $\mathsf{Send}(i, s, j, \mathsf{msg} = \top)$: In order to get σ_1, $\mathcal{B}_{\mathsf{SIG}}$ queries its signing oracle $\mathcal{O}_{\text{SIGN}}(i, (P_i, P_j, \hat{pk}))$.
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (\hat{pk}, \sigma_1))$: In order to get σ_2, $\mathcal{B}_{\mathsf{SIG}}$ queries its signing oracle $\mathcal{O}_{\text{SIGN}}(i, (P_j, P_i, \hat{pk}, \sigma_1, c))$.
- $\mathsf{Corrupt}(i)$: $\mathcal{B}_{\mathsf{SIG}}$ queries its own oracle $\mathcal{O}_{\text{CORR}}(i)$ to obtain the signing key ssk_i and returns ssk_i to $\mathcal{A}$.
- Queries $\mathsf{Send}(i, s, j, (c, \sigma_2))$, $\mathsf{RegisterCorrupt}$, $\mathsf{SessionKeyReveal}$ and Test can be simulated as in G_0.

During the simulation, $\mathcal{B}_{\mathsf{SIG}}$ checks if $\mathsf{NoMsgCon}$ happens. If this is the case, there exists an oracle π_i^s such that π_i^s has accepted and $j := \mathsf{Pid}_i^s$ is uncorrupted at that point in time.

Now we show that then there is a valid message-signature pair (m^*, σ^*) in Sent_i^s and Recv_i^s such that $\mathsf{Ver}(vk_j, m^*, \sigma^*) = 1$ and m^* is different from any message m signed by π_j^t for all $t \in [\ell]$. Since π_i^s is accepted, $\mathsf{Sent}_i^s \neq \varnothing$ and $\mathsf{Recv}_i^s \neq \varnothing$.

Case 1: π_i^s sent the first message. Let $\mathsf{Sent}_i^s = \{(\hat{pk}, \sigma_1)\}$ and $\mathsf{Recv}_i^s = \{(c, \sigma_2)\}$. We have $\mathsf{Ver}(vk_j, (P_i, P_j, \hat{pk}, \sigma_1, c), \sigma_2) = 1$, since $(c, \sigma_2) \in \mathsf{Recv}_i^s$. For any oracle π_j^t with $\mathsf{Recv}_j^t = \{(\hat{pk}', \sigma_1')\} \neq \varnothing$ and $\mathsf{Sent}_j^t = \{(c', \sigma_2')\} \neq \varnothing$, $\mathsf{NoMsgCon}$ implies that $(\hat{pk}, c) \neq (\hat{pk}', c')$. In this case, $\mathcal{B}_{\mathsf{SIG}}$ sets $(m^*, \sigma^*) := ((P_i, P_j, \hat{pk}, \sigma_1, c), \sigma_2)$.

Case 2: π_i^s received the first message. Let $\mathsf{Recv}_i^s = \{(\hat{pk}, \sigma_1)\}$ and $\mathsf{Sent}_i^s = \{(c, \sigma_2)\}$. We have $\mathsf{Ver}(vk_j, (P_j, P_i, \hat{pk}), \sigma_1) = 1$, since $\mathsf{Recv}_i^s \neq \varnothing$. For any oracle π_j^t with $\mathsf{Sent}_j^t = \{(\hat{pk}', \sigma_1')\} \neq \varnothing$, $\mathsf{NoMsgCon}$ implies that $\hat{pk} \neq \hat{pk}'$. In this case, $\mathcal{B}_{\mathsf{SIG}}$ sets $(m^*, \sigma^*) := ((P_j, P_i, \hat{pk}), \sigma_1)$.

As soon as event $\mathsf{NoMsgCon}$ happens, $\mathcal{B}_{\mathsf{SIG}}$ retrieves the message-signature (m^*, σ^*) pair as just described and outputs (j, m^*, σ^*). As P_j is uncorrupted, $\mathcal{B}_{\mathsf{SIG}}$ has not queried

$\mathcal{O}_{\text{CORR}}(j)$ and m^* is different from all signing queries for j, which concludes the proof of Lemma 9. □

Before moving to G_2, let us bound $(1) \wedge (2) \wedge (3.2)$.

Multiple Partners. Event $(1) \wedge (2) \wedge (3.2)$ happens if there exists any oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$ and has more than one partner oracle. We can show that

$$\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] \leq (\mu\ell)^2 \cdot 2^{-\gamma} \ .$$

The session key only depends on the ephemeral public key $\hat{pk}$ and the ciphertext c. In the following, we assume that there are two oracles π_j^t and $\pi_{j'}^{t'}$ such that π_i^s is partnered to both π_j^t and $\pi_{j'}^{t'}$. We distinguish two cases:

Case 1: π_i^s sent the first message. Let $\hat{pk}$ be the ephemeral public key determined by the internal randomness of π_i^s. Let $(c, K) \leftarrow \mathsf{Encap}(\hat{pk}; r)$ and $(c', K') \leftarrow \mathsf{Encap}(\hat{pk}; r')$, where r, r' is the internal randomness of π_j^t and $\pi_{j'}^{t'}$, respectively. As π_i^s is partnered to both oracles, this implies that $k_i^s = \mathsf{Decap}(\hat{sk}, c) = \mathsf{Decap}(\hat{sk}, c')$. By the correctness and γ-diversity of KEM, we have $k_i^s = K = K'$ which will happen with probability at most $2^{-\gamma}$.

Case 2: π_i^s received the first message. Let $\hat{pk}$ and $\hat{pk'}$ be the public keys determined by the internal randomness of π_j^t and $\pi_{j'}^{t'}$, respectively. Let r be the internal randomness of π_i^s which is used by Encap. The original keys are derived from $(c, K) \leftarrow \mathsf{Encap}(\hat{pk}; r)$ and $(c', K') \leftarrow \mathsf{Encap}(\hat{pk'}; r)$. As π_i^s is partnered to both oracles, $k_i^s = K = K'$. Due to γ-diversity of KEM, this will happen only with probability at most $2^{-\gamma}$.

As there are $\mu\ell$ oracles, we can upper bound the probability for event $(1) \wedge (2) \wedge (3.2)$ by $(\mu\ell)^2 \cdot 2^{-\gamma}$.

At this point note that

$$\begin{aligned}
\Pr[\mathsf{Win}_{\mathsf{Auth}}] &= \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge ((3.1) \vee (3.2))] \\
&\leq \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.1)] + \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] \\
&\leq \mathsf{Adv}^{\mathsf{mu\text{-}corr}}_{\mathsf{SIG},\mu}(\mathcal{B}_{\mathsf{SIG}}) + (\mu\ell)^2 \cdot 2^{-\gamma} \ .
\end{aligned}$$

Game G_2: In G_2, we check the partnership $\mathsf{Partner}(\pi_i^s \leftarrow \pi_j^t)$ by message-consistency $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$ if $\Psi_i^s = \textbf{accept}$ and $\mathsf{Aflag}_i^s = \textbf{false}$. We claim that

$$|\Pr[\mathsf{Win}_1] - \Pr[\mathsf{Win}_2]| \leq \Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)] \leq (\mu\ell)^2 \cdot 2^{-\gamma} \ .$$

Recall that if $\mathsf{NoMsgCon}$ does not happen, we know that each oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$ is partnered to and message-consistent with an oracle π_j^t. If any such oracle π_i^s has a unique partner, then G_1 is identical to G_2. On the other hand, the probability that there exists an oracle π_i^s that has accepted with $\mathsf{Aflag}_i^s = \textbf{false}$ and has multiple partners is $\Pr_{\exists(i,s)}[(1) \wedge (2) \wedge (3.2)]$, which is bounded by $(\mu\ell)^2 \cdot 2^{-\gamma}$.

$\mathcal{B}_{\mathsf{KEM}}^{\mathcal{O}_{\text{ENCAP}}^{\beta}(\cdot),\mathcal{O}_{\text{DECAP}}(\cdot,\cdot)}(\mathsf{pp}_{\mathsf{KEM}}, pk_1, ..., pk_{\mu\ell})$:
$\mathsf{pp}_{\mathsf{SIG}} \leftarrow\!\!\$\ \mathsf{SIG.Setup}$
For $i \in [\mu]$:
  $(vk_i, ssk_i) \leftarrow\!\!\$\ \mathsf{SIG.Gen}(\mathsf{pp}_{\mathsf{SIG}})$
  $crp_i := \mathbf{false}$
$\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$; $b \leftarrow\!\!\$\ \{0,1\}$
For $(i,s) \in [\mu] \times [\ell]$:
  $\mathsf{var}_i^s := (\mathsf{Pid}_i^s, k_i^s, \Psi_i^s) := (\emptyset, \emptyset, \emptyset)$
  $(\mathsf{Sent}_i^s, \mathsf{Recv}_i^s) := (\varnothing, \varnothing)$
  $\mathsf{Aflag}_i^s := \mathbf{false}$; $T_i^s := \mathbf{false}$; $kRev_i^s := \mathbf{false}$
$\mathsf{NoMsgCon} := \mathbf{false}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AKE}}(\cdot)}(\mathsf{pp}_{\mathsf{AKE}}, \mathsf{PKList})$
If $b^* = b$: Return $\beta^* := 0$
Else: Return $\beta^* := 1$

// During the execution $\mathcal{B}_{\mathsf{KEM}}$ checks if the following
// flag is set to true and if so, it aborts immediately:
$\mathsf{NoMsgCon} := \mathbf{true}$, If $\exists i, s \in [\mu] \times [\ell]$ s.t. $(1') \wedge (2') \wedge (3')$.
  Let $j := \mathsf{Pid}_i^s$.
  $(1')$ $\Psi_i^s = \mathbf{accept}$
  $(2')$ $\mathsf{Aflag}_i^s = \mathbf{false}$
  $(3')$ $\nexists t \in [\ell]$ s.t. $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t)$

$\mathcal{O}_{\mathsf{AKE}}(\mathsf{query})$:
If query=$\mathsf{Test}(i,s)$:
  If $\Psi_i^s \neq \mathbf{accept} \vee \mathsf{Aflag}_i^s = \mathbf{true} \vee kRev_i^s = \mathbf{true}$
    $\vee\ T_i^s = \mathbf{true}$:
      Return $\perp$
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [\ell]$ s.t. $\mathsf{MsgCon}(\pi_i^s \leftarrow \pi_j^t) \wedge k_j^t = k_i^s$:
    If $kRev_j^t = \mathbf{true} \vee T_j^t = \mathbf{true}$: Return $\perp$
  $T_i^s := \mathbf{true}$
  $k_0 := k_i^s$; $k_1 \leftarrow\!\!\$\ \mathcal{K}$
  Return k_b

If query=$\mathsf{SessionKeyReveal}(i,s)$:
  If $\Psi_i^s \neq \mathbf{accept}$: Return $\perp$
  If $T_i^s = \mathbf{true}$: Return $\perp$
  Let $j := \mathsf{Pid}_i^s$
  If $\exists t \in [\ell]$ s.t. $T_j^t = \mathbf{true}$:
    If $\mathsf{MsgCon}(\pi_j^t \leftarrow \pi_i^s) \wedge k_j^t = k_i^s$: Return $\perp$
  $kRev_i^s := \mathbf{true}$; Return k_i^s

$\mathcal{O}_{\mathsf{AKE}}(\mathsf{query})$:
If query=$\mathsf{Send}(i,s,j,\mathsf{msg})$:
  If $\Psi_i^s = \mathbf{accept}$: Return $\perp$
  If $\mathsf{msg} = \top$: //session is initiated
    $\mathsf{Pid}_i^s := j$
    Let $n := (i-1)\mu + s$; $\hat{pk} := pk_n$
    $\sigma_1 \leftarrow\!\!\$\ \mathsf{Sign}(ssk_i, (P_i, P_j, \hat{pk}))$
    $\mathsf{msg}' := (\hat{pk}, \sigma_1)$
  If $\mathsf{msg} = (\hat{pk}, \sigma_1)$: //first message
    $\mathsf{Pid}_i^s := j$
    If $\mathsf{Ver}(vk_j, (P_j, P_i, \hat{pk}), \sigma_1) \neq 1$:
      $\Psi_i^s := \mathbf{reject}$; Return $\perp$
    If $crp_j = \mathbf{false}$:
      Then $\exists$ unique t s.t. $\hat{pk}$ output by π_j^t
      Let $n := (j-1)\mu + t$
      $(c, K_\beta) \leftarrow \mathcal{O}_{\text{ENCAP}}^{\beta}(n)$; $k_i^s := K_\beta$
    Else:
      $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(\hat{pk})$; $k_i^s := K$
    $\sigma_2 \leftarrow\!\!\$\ \mathsf{Sign}(ssk_i, (P_j, P_i, \hat{pk}, \sigma_1, c))$
    $\Psi_i^s := \mathbf{accept}$
    $\mathsf{msg}' := (c, \sigma_2)$
  If $\mathsf{msg} = (c, \sigma_2)$: //second message
    Choose $(\hat{pk}, \sigma_1) \in \mathsf{Sent}_i^s$
    If $\mathsf{Pid} \neq j$ or $\mathsf{Ver}(vk_j, (P_i, P_j, \hat{pk}, \sigma_1, c), \sigma_2) \neq 1$:
      $\Psi_i^s := \mathbf{reject}$; Return $\perp$
    Let $n := (i-1)\mu + s$ and $j := \mathsf{Pid}_i^s$
    If $\exists t$ s.t. $\mathsf{Recv}_j^t = \{(\hat{pk}, \cdot)\} \wedge \mathsf{Sent}_j^t = \{(c, \cdot)\}$:
      $k_i^s := k_j^t$
    Else:
      $K \leftarrow \mathcal{O}_{\text{DECAP}}(n, c)$; $k_i^s := K$
    $\Psi_i^s := \mathbf{accept}$
    $\mathsf{msg}' := \emptyset$
  $\mathsf{Recv}_i^s := \mathsf{Recv}_i^s \cup \{\mathsf{msg}\}$; $\mathsf{Sent}_i^s := \mathsf{Sent}_i^s \cup \{\mathsf{msg}'\}$
  If $\Psi_i^s = \mathbf{accept}$:
    If $crp_j = \mathbf{true}$: $\mathsf{Aflag}_i^s := \mathbf{true}$
  Return msg'

Figure 13: Adversary $\mathcal{B}_{\mathsf{KEM}}$ against MUC-otCCA security of KEM for the proof of Theorem 3. Queries to $\mathcal{O}_{\mathsf{AKE}}$ where query$\in \{\mathsf{Corrupt}, \mathsf{RegisterCorrupt}\}$ are defined as in the original game $\mathsf{Exp}_{\mathsf{AKE},\mu,\ell,\mathcal{A}}$ in Figure 5.

Thus, the claims follows by the difference lemma.

In order to bound $\Pr[\mathsf{Win}_2]$, we construct an adversary $\mathcal{B}_{\mathsf{KEM}}$ against MUC-otCCA security of KEM (see Figure 13). We will show that

$$\left|\Pr[\mathsf{Win}_2] - \tfrac{1}{2}\right| \leq 2 \cdot \mathsf{Adv}^{\mathsf{muc\text{-}otcca}}_{\mathsf{KEM},\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) \ .$$

As in the proof of Theorem 2, we make use of the fact that we can not only replace the session key of an oracle when it is tested but of all oracles that are possibly tested. The

difference to $\mathsf{AKE}_{\mathsf{3msg}}$ is that now the adversary can replay the ephemeral public key $\hat{pk}$ to other oracles, which is why we need multiple $\mathcal{O}^{\beta}_{\text{ENCAP}}$ queries.

Let β be the random bit of $\mathcal{B}_{\mathsf{KEM}}$'s challenger. $\mathcal{B}_{\mathsf{KEM}}$ inputs the public parameter $\mathsf{pp}_{\mathsf{KEM}}$ and $\{pk_n\}_{n\in[\mu\ell]}$. $\mathcal{B}_{\mathsf{KEM}}$ generates the public parameter for SIG and signature key pairs (vk_i, ssk_i) for $i \in [\mu]$ and sets $\mathsf{PKList} := \{vk_i\}_{i\in[\mu]}$. It initializes all variables, chooses a random challenge bit $b \leftarrow_{\$} \{0,1\}$ and runs $\mathcal{A}$. If $\mathcal{A}$ makes a query to $\mathcal{O}_{\mathsf{AKE}}$, $\mathcal{B}_{\mathsf{KEM}}$ simulates the response as follows:

- $\mathsf{Send}(i, s, j, \mathsf{msg} = \top)$: $\mathcal{B}_{\mathsf{KEM}}$ uses the public key with index $(i-1)\mu+s$ as ephemeral public key, i.e. $\hat{pk} := pk_{(i-1)\mu+s}$.
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (\hat{pk}, \sigma_1))$: If P_j is uncorrupted, then due to the fact that $\mathsf{NoMsgCon}$ does not happen, there exists a unique oracle π^t_j such that $\hat{pk}$ was output by π^t_j. Furthermore, $n = (j-1)\mu + t$ is the index of that public key. Then $\mathcal{B}_{\mathsf{KEM}}$ queries $\mathcal{O}^{\beta}_{\text{ENCAP}}(n)$, receives a ciphertext and key (c, K_β) and sets $k^s_i := K_\beta$. If P_j is corrupted, $\mathcal{B}_{\mathsf{KEM}}$ runs $\mathsf{Encap}(\hat{pk})$ itself to compute (c, K). It also computes a signature σ_2 as the protocol specifies and outputs (c, σ_2).
- $\mathsf{Send}(i, s, j, \mathsf{msg} = (c, \sigma_2))$: Let $n = (i-1)\mu+s$. Then, π^s_i sent $\hat{pk}_n$. If there exists an oracle π^t_j that has received $\hat{pk}_n$ and has sent c, then $\mathcal{B}_{\mathsf{KEM}}$ sets $k^s_i := k^t_j$. Otherwise, $\mathcal{B}_{\mathsf{KEM}}$ queries $\mathcal{O}_{\text{DECAP}}(n, c)$, receives K and sets $k^s_i := K$.
- $\mathsf{Test}(i, s)$: After ruling out trivial attacks **TA1**, **TA2** and **TA3**, $\mathcal{B}_{\mathsf{KEM}}$ checks for trivial attacks **TA4** and **TA5** using message-consistency check $\mathsf{MsgCon}(\pi^s_i \leftarrow \pi^t_j)$ and tests if $k^t_j = k^s_i$. If it does not output $\perp$, $\mathcal{B}_{\mathsf{KEM}}$ sets $k_0 = k^s_i$ and $k_1 \leftarrow_{\$} \mathcal{K}$ and outputs k_b.
- $\mathsf{SessionKeyReveal}(i, s)$: After ruling out trivial attack **TA2**, $\mathcal{B}_{\mathsf{KEM}}$ checks for trivial attack **TA4** by checking if there exists an oracle π^t_j such that π^t_j is tested and $\mathsf{MsgCon}(\pi^t_j \leftarrow \pi^s_i)$. If further $k^t_j = k^s_i$, $\mathcal{B}_{\mathsf{KEM}}$ returns $\perp$. Otherwise, it outputs k^s_i.
- Queries $\mathsf{Corrupt}$ and $\mathsf{RegisterCorrupt}$ and can be simulated as in G_2.

Finally, $\mathcal{A}$ outputs b^* and $\mathcal{B}_{\mathsf{KEM}}$ outputs $\beta^* := 0$ if $b^* = b$ and $\beta^* := 1$ otherwise.

$\mathcal{B}_{\mathsf{KEM}}$ may query $\mathcal{O}^{\beta}_{\text{ENCAP}}$ multiple times on the same ephemeral public key $\hat{pk}$ as we cannot prevent replay attacks. However, each query to $\mathsf{Send}(i, s, j, \mathsf{msg} = \top)$ chooses a different ephemeral public key. When the oracle receives a ciphertext, it accepts and thus $\mathcal{O}_{\text{DECAP}}$ is called at most once. In the following, we will argue that $\mathcal{B}_{\mathsf{KEM}}$ perfectly simulates G_2 if $\beta = 0$, and that $\mathcal{A}$'s view is independent of b if $\beta = 1$. The analysis is the same as in the proof of Theorem 2. We have

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{muc\text{-}otcca}}_{\mathsf{KEM},\mu\ell}(\mathcal{B}_{\mathsf{KEM}}) &= \left|\Pr[\beta^* = \beta] - \tfrac{1}{2}\right| \\
&= \left|\tfrac{1}{2}\cdot\Pr[\beta^* = \beta \mid \beta = 0] + \tfrac{1}{2}\cdot\Pr[\beta^* = \beta \mid \beta = 1] - \tfrac{1}{2}\right| \\
&= \left|\tfrac{1}{2}\cdot\Pr[\mathsf{Win}_2] + \tfrac{1}{2}\cdot\tfrac{1}{2} - \tfrac{1}{2}\right| = \tfrac{1}{2}\left|\Pr[\mathsf{Win}_2] - \tfrac{1}{2}\right| \ .
\end{aligned}$$

Collecting the probabilities yields the bound in Theorem 3. □

B Proof of Lemma 4

We recall random self-reducibility of the MDDH assumption.

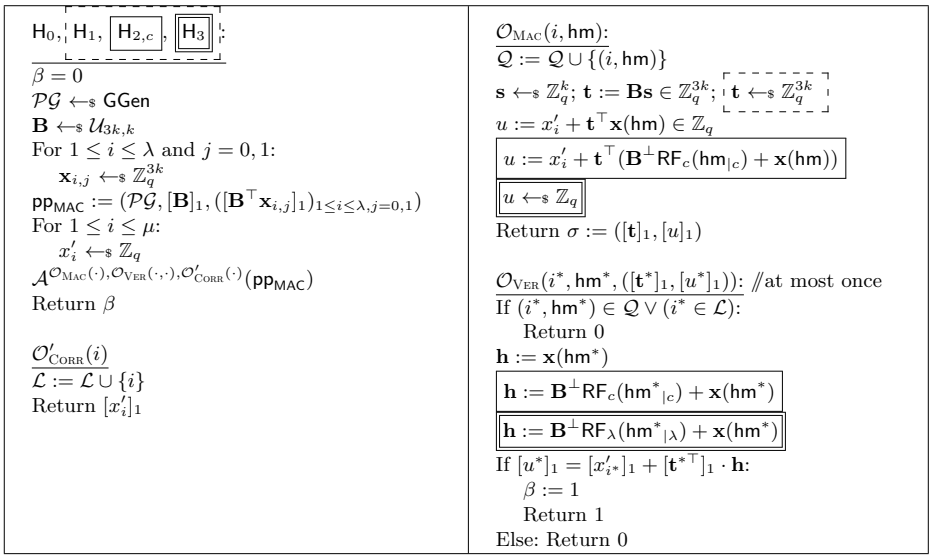

Figure 14: Games $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_{2,c}, \mathsf{H}_3, \mathsf{H}_4$ for proving Lemma 4 where $0 \le c \le \lambda$ and $\mathsf{RF}_c : \{0,1\}^c \to \mathbb{Z}_q^{2k}$ is a random function.

For $Q \in \mathbb{N}$, $\mathbf{W} \leftarrow_\$ \mathbb{Z}_q^{k \times Q}$, $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{\ell \times Q}$, we consider the Q-fold $\mathcal{D}_{\ell,k}$-MDDH problem which is to distinguish the distributions $([\mathbf{A}], [\mathbf{AW}])$ and $([\mathbf{A}], [\mathbf{U}])$. Essentially, the Q-fold $\mathcal{D}_{\ell,k}$-MDDH problem contains Q independent instances of the $\mathcal{D}_{\ell,k}$-MDDH problem (with the same $\mathbf{A}$ but different $\mathbf{w}_i$). The following lemma gives a tight reduction.

Lemma 10 (Random self-reducibility [EHK+13])**.** *For $\ell > k$ and any matrix distribution $\mathcal{D}_{\ell,k}$, the $\mathcal{D}_{\ell,k}$-*MDDH *assumption is random self-reducible. In particular, for any $Q \ge 1$ and any adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$ with*

$$\mathsf{Adv}^{Q\text{-MDDH}}_{\mathsf{GGen},\mathcal{D}_{\ell,k},\mathbb{G}_s}(\mathcal{B}) := \Pr\left[\mathcal{B}\left(\mathcal{PG}, [\mathbf{A}]_s, [\mathbf{AW}]_s\right) \Rightarrow 1\right] - \Pr\left[\mathcal{B}\left(\mathcal{PG}, [\mathbf{A}]_s, [\mathbf{U}]_s\right) \Rightarrow 1\right]$$
$$\le (\ell - k)\, \mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{D}_{\ell,k},\mathbb{G}_s}(\mathcal{A}) + \frac{1}{q-1},$$

where $\mathcal{PG} \leftarrow_\$ \mathsf{GGen}$, $\mathbf{A} \leftarrow_\$ \mathcal{D}_{\ell,k}$, $\mathbf{W} \leftarrow_\$ \mathbb{Z}_q^{k \times Q}$, $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{(k+1) \times Q}$, and $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$.

Lemma 4. We prove Lemma 4 by the sequence of games defined in Figure 14. Let $\mathcal{A}$ be an adversary against the security game $\mathsf{UF\text{-}CMA}^{\mathsf{corr}}$, and let Win_i denote the probability that H_i returns 1.

Game H_0: This is the same game as $\mathsf{UF\text{-}CMA}^{\mathsf{corr}}$. Thus,

$$\Pr[\mathsf{UF\text{-}CMA}^{\mathsf{corr}}_{\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{Win}_0].$$

Game H_1: In H_1, we switch $\mathbf{t}$ in $\mathcal{O}_{\mathrm{MAC}}$ from $\mathsf{Span}(\mathbf{B})$ to random over $\mathbb{Z}_q^{3k}$. By using the Q_e-fold $\mathcal{U}_{3k,k}$-MDDH assumption and Lemma 10, we have

$$|\Pr[\mathsf{Win}_0] - \Pr[\mathsf{Win}_1]| \le 2k \mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{U}_{3k,k},\mathbb{G}_1}(\mathcal{B}) + \frac{1}{q-1}.$$

Game $\mathsf{H}_{2,c}$ ($0 \le c \le \lambda$): In $\mathsf{H}_{2,c}$, we use a random function $\mathsf{RF}_c : \{0,1\}^c \to \mathbb{Z}_q^{2k}$ to randomize $\mathcal{O}_{\mathrm{MAC}}$ and $\mathcal{O}_{\mathrm{VER}}$ queries.

Before we justify the difference, we recall some useful linear algebra facts for the following proofs. For a random matrix $\mathbf{B} \in \mathbb{Z}_q^{3k\times k}$, there is a non-zero kernel matrix $\mathbf{B}^\perp \in \mathbb{Z}_q^{3k\times 2k}$ such that $\mathbf{B}^\top\mathbf{B}^\perp = \mathbf{0}$. It is efficient to generate random $\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_0^*, \mathbf{B}_1^* \in \mathbb{Z}_q^{3k\times k}$ such that: $(\mathbf{B} \parallel \mathbf{B}_0 \parallel \mathbf{B}_1)$ is a basis for $\mathbb{Z}_q^{3k}$; $(\mathbf{B}_0^* \parallel \mathbf{B}_1^*)$ is a basis for $\mathsf{Span}(\mathbf{B}^\perp)$; and $\mathbf{B}_0^\top\mathbf{B}_1^* = \mathbf{B}_1^\top\mathbf{B}_0^* = \mathbf{0}$. Figure 15 visualizes these properties.

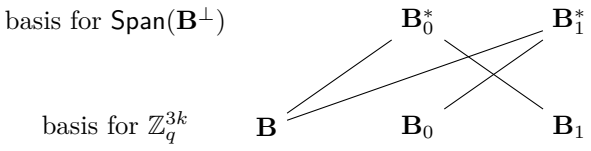

Figure 15: Solid lines mean orthogonal: $\mathbf{B}^\top\mathbf{B}_0^* = \mathbf{B}_1^\top\mathbf{B}_0^* = \mathbf{0} = \mathbf{B}^\top\mathbf{B}_1^* = \mathbf{B}_0^\top\mathbf{B}_1^* \in \mathbb{Z}_q^{k\times k}$.

We show that H_1 and $\mathsf{H}_{2,0}$ are identical by viewing $\mathbf{x}_{1,b}$ as $\mathbf{x}_{1,b} + \mathbf{B}^\perp\mathsf{RF}_0(\varepsilon)$ (for both $b = 0, 1$) where $\mathsf{RF}_0(\varepsilon)$ is a fixed random value. Thus,

$$\Pr[\mathsf{Win}_1] = \Pr[\mathsf{Win}_{2,0}].$$

To bound the difference between $\mathsf{H}_{2,c}$ and $\mathsf{H}_{2,c+1}$ (for $0 \le c \le \lambda - 1$), we define a sequence of games in Figure 16.

Game $\mathsf{H}_{2,c,1}$: In $\mathsf{H}_{2,c,1}$, instead of generating a random $\mathbf{t}$ in $\mathcal{O}_{\mathrm{MAC}}$, we choose $\mathbf{t}$ from $\mathsf{Span}(\mathbf{B} \parallel \mathbf{B}_0)$ (if $\mathsf{hm}_{j+1} = 0$) or $\mathsf{Span}(\mathbf{B} \parallel \mathbf{B}_1)$ (if $\mathsf{hm}_{j+1} = 1$). This is the same step as Lemma 17 of [LP20]. By using the Q_e-fold $\mathcal{U}_{3k,k}$-MDDH assumption in $\mathbb{G}_1$ twice (one with $[\mathbf{B}_0]_1$ and the other with $[\mathbf{B}_1]_1$) and Lemma 10, we have

$$|\Pr[\mathsf{Win}_{2,c}] - \Pr[\mathsf{Win}_{2,c,1}]| \le 4k\mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{U}_{3k,k},\mathbb{G}_1}(\mathcal{B}) + \frac{2}{q-1}$$

Game $\mathsf{H}_{2,c,2}$: Let $\mathsf{ZF}_c, \mathsf{OF}_c : \{0,1\}^c \to \mathbb{Z}_q^k$ be random functions. In $\mathsf{H}_{2,c,2}$, we decompose the terms $\mathbf{B}^\perp\mathsf{RF}_c(\mathsf{hm}_{|c})$ in $\mathcal{O}_{\mathrm{MAC}}$ as $\mathbf{B}_0^*\mathsf{ZF}_c(\mathsf{hm}_{|c}) + \mathbf{B}_1^*\mathsf{OF}_c(\mathsf{hm}_{|c})$. Similarly, we also decompose the terms $\mathbf{B}^\perp\mathsf{RF}_c(\mathsf{hm}^*{}_{|c})$ in $\mathcal{O}_{\mathrm{VER}}$ as $\mathbf{B}_0^*\mathsf{ZF}_c(\mathsf{hm}^*{}_{|c}) + \mathbf{B}_1^*\mathsf{OF}_c(\mathsf{hm}^*{}_{|c})$. Since $(\mathbf{B}_0^* \parallel \mathbf{B}_1^*)$ is a basis for $\mathsf{Span}(\mathbf{B}^\perp)$, these changes will not modify the distribution, and $\mathsf{H}_{2,c,1}$ and $\mathsf{H}_{2,c,2}$ are identical. Thus, we have

$$\Pr[\mathsf{Win}_{2,c,1}] = \Pr[\mathsf{Win}_{2,c,2}].$$

Game $\mathsf{H}_{2,c,3}$: We define

$$\mathsf{ZF}_{c+1}(\mathsf{hm}_{|c+1}) = \begin{cases} \mathsf{ZF}_c(\mathsf{hm}_{|c}) & (\text{if } \mathsf{hm}_{c+1} = 0) \\ \mathsf{ZF}_c(\mathsf{hm}_{|c}) + \mathsf{ZF}'_c(\mathsf{hm}_{|c}) & (\text{if } \mathsf{hm}_{c+1} = 1) \end{cases}, \tag{10}$$

where $\mathsf{ZF}'_c : \{0,1\}^c \to \mathbb{Z}_q^k$ is another independent random function. Note that $\mathsf{ZF}_{c+1} : \{0,1\}^{c+1} \to \mathbb{Z}_q^k$ is a random function.

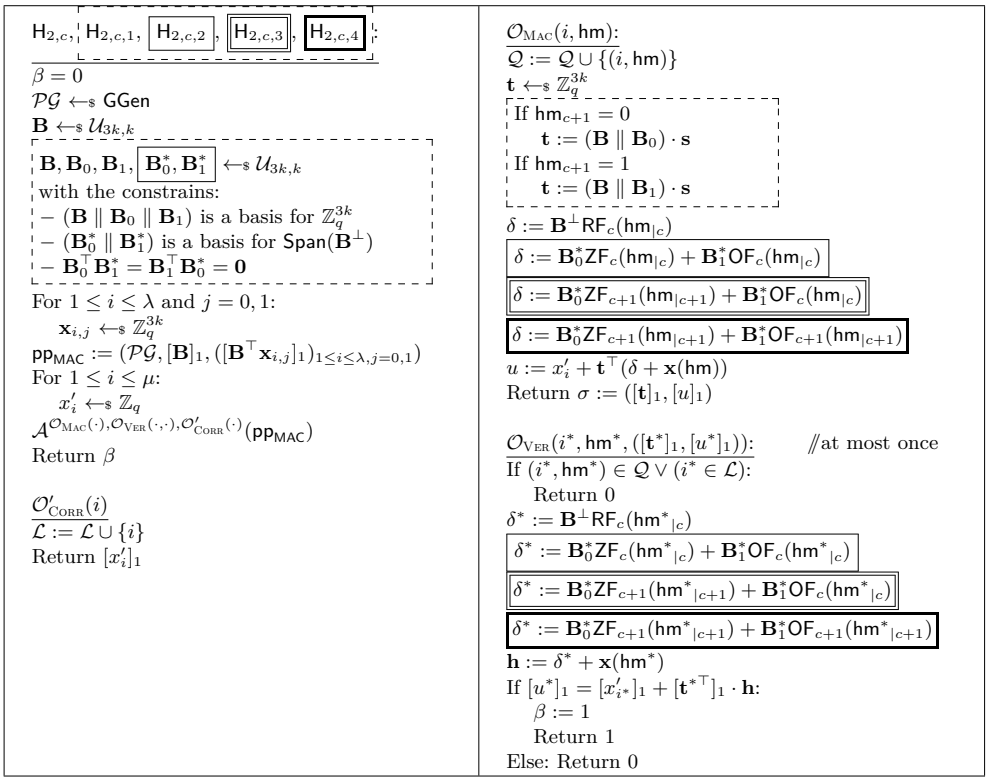

Figure 16: Games for bounding the difference between $\mathsf{H}_{2,c}$ and $\mathsf{H}_{2,c+1}$ ($1 \le c \le \lambda - 1$).

In $\mathsf{H}_{2,c,3}$, we use this new random function ZF_{c+1} to simulate our security game. We observe that:

- In a $\mathcal{O}_{\text{MAC}}(i, \mathsf{hm})$ query, if $\mathsf{hm}_{c+1} = 1$, then $\mathbf{t} \in \mathsf{Span}(\mathbf{B} \parallel \mathbf{B}_1)$, and the answer to the query is distributed identically in both $\mathsf{H}_{2,c,3}$ and $\mathsf{H}_{2,c,2}$; if $\mathsf{hm}_{c+1} = 0$, then $\mathsf{ZF}_{c+1}(\mathsf{hm}_{|c+1}) = \mathsf{ZF}_c(\mathsf{hm}_{|c})$ and its answer is distributed identically in both games.
- In a $\mathcal{O}_{\text{VER}}(i^*, \mathsf{hm}^*)$ query, if $\mathsf{hm}^*_{c+1} = 0$, then the answer is distributed identically in both games. If $\mathsf{hm}^*_{c+1} = 1$, we can view $\mathbf{x}_{c+1,1}$ as $\mathbf{x}_{c+1,1} + \mathbf{B}_0^* \mathbf{w}$ for $\mathbf{w} \leftarrow_\$ \mathbb{Z}_q^k$. Since $\mathbf{B}^\top \mathbf{B}_0^* = \mathbf{0}$, $\mathbf{w}$ is perfectly hidden from the term $[\mathbf{B}^\top \mathbf{x}_{c+1,1}]_1$. Moreover, in a $\mathcal{O}_{\text{MAC}}(i, \mathsf{hm})$ query, $\mathbf{x}_{c+1,1}$ appears if $\mathsf{hm}_{c+1} = 1$. But, since $\mathbf{B}_1^\top \cdot \mathbf{B}_0^* = \mathbf{0}$, $\mathbf{w}$ has never been leaked from $\mathcal{O}_{\text{MAC}}$ queries. By viewing $\mathbf{w}$ as $\mathsf{ZF}_{c+1}(\mathsf{hm}^*{}_{|c+1})$ (for $\mathsf{hm}^*_{c+1} = 1$), we have the one-time $\mathcal{O}_{\text{VER}}(i^*, \mathsf{hm}^*)$ query is distributed the same as in both games.

As a result of the above arguments, we have

$$\Pr[\mathsf{Win}_{2,c,2}] = \Pr[\mathsf{Win}_{2,c,3}].$$

Game $\mathsf{H}_{2,c,4}$: We define

$$\mathsf{OF}_{c+1}(\mathsf{hm}_{|c+1}) = \begin{cases} \mathsf{OF}_c(\mathsf{hm}_{|c}) + \mathsf{OF}'_c(\mathsf{hm}_{|c}) & (\text{if } \mathsf{hm}_{c+1} = 0) \\ \mathsf{OF}_c(\mathsf{hm}_{|c}) & (\text{if } \mathsf{hm}_{c+1} = 1) \end{cases}, \tag{11}$$

where $\mathsf{OF}'_c : \{0,1\}^c \to \mathbb{Z}_q^k$ is another independent random function. Note again that $\mathsf{OF}_{c+1} : \{0,1\}^{c+1} \to \mathbb{Z}_q^k$ is a random function.

By a similar argument as in $\mathsf{H}_{2,c,3}$ (but in a symmetric manner), we can show that

$$\Pr[\mathsf{Win}_{2,c,3}] = \Pr[\mathsf{Win}_{2,c,4}].$$

To bound the difference between $\mathsf{H}_{2,c,4}$ and $\mathsf{H}_{2,c+1}$, we will do the same argument as in $\mathsf{H}_{2,c,1}$ and $\mathsf{H}_{2,c,2}$ but in a reverse order. Namely, we first compose $\mathbf{B}_0^*\mathsf{ZF}_{c+1}(\mathsf{hm}_{|c+1}) + \mathbf{B}_1^*\mathsf{OF}_{c+1}(\mathsf{hm}_{|c+1})$ to $\mathbf{B}^\perp\mathsf{RF}_{c+1}(\mathsf{hm}_{|c+1})$ for both $\mathcal{O}_{\mathrm{MAC}}$ and $\mathcal{O}_{\mathrm{VER}}$ (which is only information-theoretic), and switch $\mathbf{t}$ in $\mathcal{O}_{\mathrm{MAC}}$ back to random (which is bounded by using the MDDH assumption). Then we have

$$|\Pr[\mathsf{Win}_{2,c,4}] - \Pr[\mathsf{Win}_{2,c+1}]| \le 4k\mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{U}_{3k,k},\mathbb{G}_1}(\mathcal{B}) + \frac{2}{q-1}.$$

Thus, the difference between $\mathsf{H}_{2,c}$ and $\mathsf{H}_{2,c+1}$ is bounded by

$$|\Pr[\mathsf{Win}_{2,c}] - \Pr[\mathsf{Win}_{2,c+1}]| \le 8k\mathsf{Adv}^{\mathrm{MDDH}}_{\mathsf{GGen},\mathcal{U}_{3k,k},\mathbb{G}_1}(\mathcal{B}) + \frac{4}{q-1}.$$

Game H_3: Compared to $\mathsf{H}_{2,\lambda}$, the only change H_3 is to choose u uniformly at random from $\mathbb{Z}_q$. We show that, even with adaptive corruption $\mathcal{O}'_{\mathrm{CORR}}$ queries, this change does not affect the view of adversary $\mathcal{A}$, and $\mathsf{H}_{2,\lambda}$ is identical to H_3.

Our argument is as follows: For a $\mathcal{O}_{\mathrm{MAC}}(i, \mathsf{hm})$ query in $\mathsf{H}_{2,\lambda}$, $\mathbf{t}$ is chosen uniformly in $\mathbb{Z}_q^{3k}$ and thus $\mathbf{t}^\top\mathbf{B}^\perp\mathsf{RF}_\lambda(\mathsf{hm})$ is a random value in $\mathbb{Z}_q$ with overwhelming probability $(1 - 2k/q)$, even if an (unbounded) adversary corrupts the corresponding x'_i. Thus, we have

$$|\Pr[\mathsf{Win}_{2,\lambda}] - \Pr[\mathsf{Win}_3]| \le \frac{2k}{q}.$$

Moreover, in H_3, the information about x'_{i^*} is perfectly hidden from $\mathcal{A}$, and thus $\mathcal{A}$ can compute a $([\mathbf{t}^*]_1, [u^*]_1)$ such that $[u^*]_1 = [x'_{i^*}]_1 + [\mathbf{t}^{*\top}]_1 \cdot \mathbf{h}$ with probability $1/q$, and we have

$$\Pr[\mathsf{Win}_3] \le \frac{1}{q}.$$

This completes the proof. □

Appendix D

Password-Authenticated Key Exchange from Group Actions

An extended abstract of this article appears in the proceedings of CRYPTO 2022. The following version is the full version of this article which is also available in the IACR ePrint archive, ia.cr/2022/770.

Original Publication

M. Abdalla, T. Eisenhofer, E. Kiltz, S. Kunzweiler, and D. Riepel. Password-Authenticated Key Exchange from Group Actions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 699–728. Springer, Heidelberg, August 2022. https://doi.org/10.1007/978-3-031-15979-4_24

Password-Authenticated Key Exchange from Group Actions

Michel Abdalla[1,2], Thorsten Eisenhofer[3], Eike Kiltz[3], Sabrina Kunzweiler[3], Doreen Riepel[3]

[1] DFINITY, Zürich, Switzerland
[2] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@ens.fr
[3] Ruhr-Universität Bochum, Bochum, Germany
{thorsten.eisenhofer,eike.kiltz,sabrina.kunzweiler,doreen.riepel}@rub.de

Abstract. We present two provably secure password-authenticated key exchange (PAKE) protocols based on a commutative group action. To date the most important instantiation of isogeny-based group actions is given by CSIDH. To model the properties more accurately, we extend the framework of cryptographic group actions (Alamati et al., ASIACRYPT 2020) by the ability of computing the quadratic twist of an elliptic curve. This property is always present in the CSIDH setting and turns out to be crucial in the security analysis of our PAKE protocols.
Despite the resemblance, the translation of Diffie-Hellman based PAKE protocols to group actions either does not work with known techniques or is insecure ("How not to create an isogeny-based PAKE", Azarderakhsh et al., ACNS 2020). We overcome the difficulties mentioned in previous work by using a "bit-by-bit" approach, where each password bit is considered separately.
Our first protocol $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ can be executed in a single round. Both parties need to send two set elements for each password bit in order to prevent offline dictionary attacks. The second protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ requires only one set element per password bit, but one party has to send a commitment on its message first. We also discuss different optimizations that can be used to reduce the computational cost. We provide comprehensive security proofs for our base protocols and deduce security for the optimized versions.

Keywords: Password-authenticated key exchange, group actions, CSIDH

1 Introduction

Password-authenticated key exchange (PAKE) enables two parties to securely establish a joint session key assuming that they only share a low-entropy secret known as the password. This reflects that passwords are often represented in short human-readable formats and are chosen from a small set of possible values, often referred to as dictionary.

Since the introduction of PAKE by Bellovin and Merritt [BM92], many PAKE protocols have been proposed, including SPEKE [Jab96], SPAKE2 [AP05], J-PAKE [HR10] and CPace [HL19]. In particular over the last few years, the design and construction

of PAKE protocols has attracted increasing attention, as the Crypto Forum Research Group (CFRG) which is part of the Internet Research Task Force (IETF) started a selection process to decide which PAKE protocols should be used in IETF protocols. Recently, CPace was selected as the recommended protocol for symmetric PAKE, where both parties share the same password.

Different models have been used to formally prove security of PAKE protocols, like indistinguishability-based models or the universal composability framework. In general, a PAKE protocol should resist offline and online dictionary attacks. On the one hand an adversary should not be able to perform an exhaustive search of the password offline. On the other hand, an active adversary should only be able to try a small number of passwords in one protocol execution. Furthermore, forward security ensures that session keys are still secure, even if the password is leaked at a later point in time. The same should hold if session keys are disclosed, which should not affect security of other session keys.

CSIDH and Group Actions. The PAKE protocols mentioned above are mostly based on a Diffie-Hellman key exchange in a prime order group. A promising post-quantum replacement is isogeny-based key exchange. The different isogeny-based protocols can be divided into two groups. On the one hand there are constructions based on commutative group actions on a set of elliptic curves. The first proposals by Couveignes [Cou06], and Stolbunov and Rostovtsev [RS06] suggested to use the action of the class group $cl(\mathcal{O})$ on the set of $\mathbb{F}_q$-isomorphism classes of *ordinary* elliptic curves with endomorphism ring $\mathcal{O}$. In 2018, Castryck et al. showed that this idea can also be adapted to the class group action on the set of $\mathbb{F}_p$-isomorphism classes of *supersingular* elliptic curves [CLM+18]. The resulting scheme is called CSIDH and constitutes the first practical key exchange scheme based on class group actions.

In [Cou06], Couveignes introduces *hard homogeneous spaces* - an abstract framework for group actions that models isogeny-based assumptions. This framework has been further refined by Alamati et al. in [ADMP20]. Using the abstract setting of *cryptographic group actions* the authors develop several new cryptographic primitives that can be instantiated with CSIDH. On the other hand there is the Supersingular Isogeny Diffie-Hellman (SIDH) protocol suggested by Jao and De Feo in 2011 [JD11]. Here, the set of $\mathbb{F}_{p^2}$-isomorphism classes of *supersingular* elliptic curves is considered. The endomorphism ring of a supersingular elliptic curve over $\mathbb{F}_{p^2}$ is non-commutative, hence protocols based on SIDH do not fall into the group action framework.

We now recall the framework of (restricted) effective group actions introduced in [ADMP20]. Throughout, $\mathcal{G}$ denotes a finite commutative group and $\mathcal{X}$ a set. We assume that $\mathcal{G}$ acts regularly on $\mathcal{X}$ via the operator $\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$. Regularity guarantees that for any $x, y \in \mathcal{X}$ there exists precisely one group element $g \in \mathcal{G}$ satisfying $y = g \star x$. Broadly speaking, we are interested in group actions, where evaluation is easy, but the "discrete logarithm problem" is hard. Expressed differently:

- Given $x \in \mathcal{X}$ and $g \in \mathcal{G}$, one can efficiently compute the set element $y = g \star x$.
- Given $x, y \in \mathcal{X}$, it is hard to find the element $g \in \mathcal{G}$ satisfying $y = g \star x$.

These properties facilitate the definition of a Diffie-Hellman key exchange. Let x be some fixed set element. Alice chooses a secret $g_A \in \mathcal{G}$ and publishes $y_A = g_A \star x$.

Similarly Bob chooses $g_B \in \mathcal{G}$ and publishes $y_B = g_B \star x$. They can both compute the shared secret $y_{AB} = g_A \star y_B = g_B \star y_A$. The group action computational Diffie-Hellman problem (GA-CDH) then states that given y_A and y_B, it is hard to compute y_{AB}. We refer to Section 3 for more precise definitions.

Contributions and Technical Details. Our main contributions are the two PAKE protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ based on commutative group actions. These are the first two provably secure PAKE protocols that are directly constructed from isogenies.

Group Actions with Twists. To date the most important instantiation of isogeny-based group actions is given by CSIDH. To model this situation more accurately, we suggest an enhancement of the framework which includes the ability of computing the quadratic twist of an elliptic curve efficiently. This property is inherent to CSIDH (cf. [CLM+18]) and it turns out to be crucial in the security analysis of our PAKE protocols. On the one hand, twisting allows us to construct an offline dictionary attack against our first natural PAKE attempt GA-PAKE$_\ell$. Notably, this first protocol is secure for group actions where twisting is not possible efficiently. On the other hand, twists play an important role in various security reductions applied to prove the security of our new protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$. Interestingly, this is also the case when twists are not part of any of the two problems involved in the reduction.

First attempt: GA-PAKE$_\ell$. Our two secure PAKE protocols are modifications of GA-PAKE$_\ell$. In order to illustrate the main idea behind the protocols, we describe GA-PAKE$_\ell$ in more detail here. The protocol (Figure 1) can be seen as an adaption of the simple password exponential key exchange protocol SPEKE [Jab96] to the group action setting. In SPEKE the password is used to hash to a generator of the group. Then the user and the server establish a session key following the Diffie-Hellman key exchange. Directly translating this protocol to the group action setting requires to hash the password to a random set element $x \in \mathcal{X}$. For isogeny-based group actions, this is still an open problem, hence (at the moment) a straightforward translation of SPEKE is not possible (see also [AJK+20, §4.1]). In GA-PAKE$_\ell$ we map the password to an ℓ-tuple of elements in $\mathcal{X}$ instead of hashing to one element. More precisely, two elements $\mathsf{crs} = (x_0, x_1) \in \mathcal{X}^2$ are fixed by a trusted party and a password $\mathsf{pw} = (b_1, \ldots, b_\ell) \in \{0,1\}^\ell$ is mapped to the tuple $(x_{b_1}, \cdots, x_{b_\ell}) \in \mathcal{X}^\ell$. Then a Diffie-Hellman key exchange is performed with basis x_{b_i} for each $i \in [\ell]$. This means the user generates ℓ random group elements $u_1, \ldots, u_\ell$ and computes the elements $x_1^{\mathsf{U}} = u_1 \star x_{b_1}, \ldots, x_\ell^{\mathsf{U}} = u_\ell \star x_{b_\ell}$ which it sends to the server. Similarly, the server generates ℓ random group elements $s_1, \ldots, s_\ell$ and computes $x_1^{\mathsf{S}} = s_1 \star x_{b_1}, \ldots, x_\ell^{\mathsf{S}} = s_\ell \star x_{b_\ell}$ which it sends to the user. Note that the messages may be sent simultaneously in one round. Then both parties compute $z_i = u_i \star x_i^{\mathsf{S}} = s_i \star x_i^{\mathsf{U}}$ for each $i \in [\ell]$. Finally the session key K is computed as $K = \mathsf{H}(\mathsf{U}, \mathsf{S}, x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}, x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}, \mathsf{pw}, z_1, ..., z_\ell)$, where $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ is a hash function into the key space $\mathcal{K}$.

In Section 5, we present an offline dictionary attack against GA-PAKE$_\ell$ for group actions with twists. This attack is not captured by the abstract group action framework defined in [ADMP20] which underlines the necessity of our suggested enhancement of the framework. Roughly speaking, the attack uses the fact that an attacker can

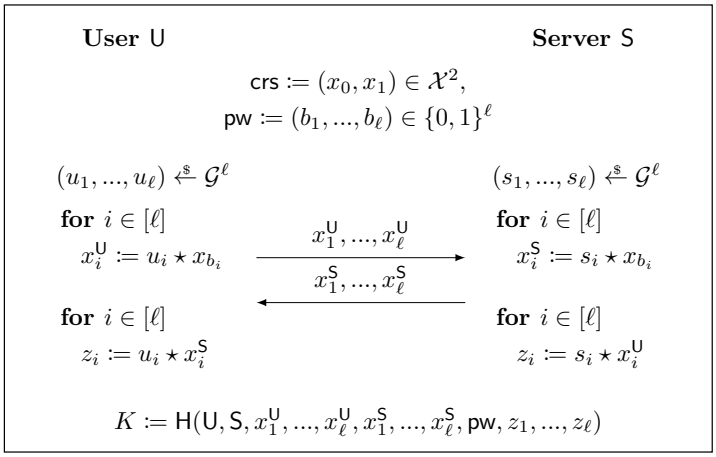

Figure 1: First Attempt: Protocol GA-PAKE$_\ell$.

choose its message in dependence on the other party's message. Using twists, it can then achieve that certain terms in the key derivation cancel out and the session key no longer depends on the other party's input.

Secure PAKE: X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$. The protocol X-GA-PAKE$_\ell$ is a modified version of GA-PAKE$_\ell$. Here security is achieved by doubling the message length in the first round of the protocol and tripling it in the key derivation. Intuitively the additional parts of the message can be viewed as an additional challenge for the key derivation that inhibits an attacker from choosing its message depending on the other party's message. The security of the protocol relies on a new computational assumption, SqInv-GA-StCDH, in which the adversary needs to compute the square and the inverse of its input at the same time (cf. Definition 7, Theorem 1).

The protocol Com-GA-PAKE$_\ell$ is a modification of GA-PAKE$_\ell$ as well. In order to achieve security against offline dictionary attacks, the protocol requires that the server sends a commitment before receiving the first message from the user. This prevents that any party chooses its message depending on the other party's message. We reduce the security of the protocol to the hardness of standard security assumptions in the isogeny-based setting (Theorem 2). An overview of our results is provided in Figure 2.

Optimizations. Both X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ require to compute multiple group action evaluations. In the last section, we discuss two optimizations that can be used to reduce the number of evaluations and show that these do not affect the security of the protocols. The first makes a tradeoff between the size of the public parameters (the common reference string crs) and the number of elements that have to be sent as well as the group actions that have to be performed. The second optimization relies on the possibility to compute twists efficiently, which is yet another advantage of adding this property to the framework and which allows to decrease the size of the public parameters by a factor of 2. We denote the final optimizations by Com-GA-PAKE$^{\mathsf{t}}_{\ell,N}$ and X-GA-PAKE$^{\mathsf{t}}_{\ell,N}$, where N is a parameter for the crs size. If N equals 1, we omit it. An overview and example of the parameter choice is provided in Table 1.

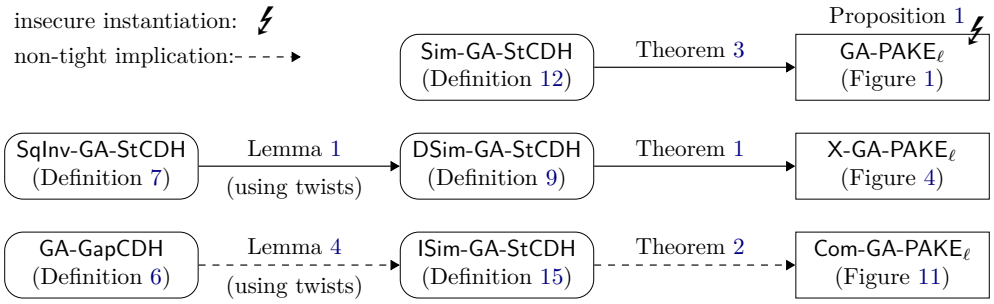

Figure 2: Overview of our security implications between assumptions (round boxes) and schemes (square boxes). Note that there exists an attack against protocol GA-PAKE$_\ell$ using twists which makes it insecure for CSIDH. Our two main protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ are proven secure under protocol-specific assumptions, but we also give reductions to simpler assumptions making use of the twisting property. Solid arrows denote tight reductions, dashed arrows non-tight reductions.

Difficulties in constructing PAKE from Isogenies. Terada and Yoneyama [TY19] proposed isogeny-based PAKE based on the EKE approach. The basic idea is that the parties perform an SIDH or CSIDH key exchange where the messages are encrypted with the password. However, as shown in [AJK+20], these protocols are not only vulnerable to offline dictionary attacks, but a modified version is even vulnerable to man-in-the-middle attacks. The main reason for the insecurity is that the elliptic curves used in the key exchange and encrypted with the password are distinguishable from random bitstrings. An exhaustive search over all passwords just requires to check if the decrypted message is a valid curve.

Another proposal based on SIDH was made by Taraskin et al. [TSJL18]. In this protocol the password is used to obfuscate the auxiliary points that are exchanged during an SIDH key exchange. While their obfuscation method prevents a certain type of offline dictionary attack, the authors were not able to provide a security proof for their protocol. The same is true for a symmetric variant of the protocol proposed by Soukharev and Hess [SH19]. Until now these are the only PAKE protocol based on isogenies which are not broken.

As noted in [AJK+20], other popular Diffie-Hellman constructions may also not be directly translated into the isogeny setting. The main reason is that hashing into the set of supersingular elliptic curves is still an open problem. This approach is for example used in SPEKE. (However, we show how to non-trivially translate the idea.) Also the approach of J-PAKE seems difficult as in this scheme different public keys are combined to obtain certain "mixed" public keys. In isogeny-based protocols, the public keys are elliptic curves and there is no natural ring structure on the set of elliptic curves that would allow to combine two elliptic curves.

In the following, we elaborate known generic constructions of PAKE from hash proof systems (HPS) and oblivious transfer (OT). We explain that the only known isogeny-based HPS is not suitable for generic constructions. On the other hand, the isogeny-based OT protocols from the literature are suited for generic constructions.

Protocol	$\lvert\mathsf{crs}\rvert$	Elements	Evaluations	Rounds	Assumption	Rew.	ROM
$\mathsf{X\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$	2^{N-1}	$2\ell/N$	$5\ell/N$	1	SqInv-GA-StCDH	no	yes
$\hookrightarrow (\ell, N) = (128, 8)$	128	32	80				
$\mathsf{Com\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$	2^{N-1}	ℓ/N (+1)	$2\ell/N$	3	Sq-GA-GapCDH	yes	yes
$\hookrightarrow (\ell, N) = (128, 8)$	128	16 (+1)	32				
OT-based$_\ell$ [LGd21]	1	3ℓ (+6ℓ)	11ℓ	4	GA-CDH	yes	yes
$\hookrightarrow \ell = 128$	1	384 (+768)	1408				
OT-based$_\ell$ [ADMP20, PVW08]	4	$> \ell^2$	$> \ell^2$	3	GA-DDH + CCA PKE	no	no
$\hookrightarrow \ell = 128$	4	$> 16,000$	$> 16,000$				

Table 1: Overview of our optimized protocols $\mathsf{Com\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$ and $\mathsf{X\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$ and comparison to the only other CSIDH-based constructions. All protocols use a bit-wise approach, i.e., passwords are treated as bitstrings of length ℓ. Sample values for $\ell = 128$ are marked in gray. "Elements" refers to the number of set elements (+ strings or symmetric ciphertexts) that each party has to send. "Evaluations" refers to the number of group action evaluations that each party has to perform. "Rew." indicates that rewinding is used to reduce to the assumption indicated in the table and GA-DDH refers to the group action decisional Diffie-Hellman problem. We apply the compiler of [CDVW12] to obtain OT-based constructions.

However, we show that the resulting PAKE protocols are less efficient than our new proposals.

Using the framework of cryptographic group actions, Alamati et al. construct a universal hash proof system [ADMP20, §4.1]. Their HPS is defined for the subset membership problem based on the DDH assumption for group actions. However, we need a different type of subset membership problem in order to construct PAKE. In particular, the framework introduced by Gennaro and Lindell [GL03] and that of follow-up works [GK10, KV11] uses an HPS for the language of ciphertexts of a public-key encryption scheme. More concretely, given a public key of the encryption scheme, a pair of message and ciphertext (m, c) is in the language of the HPS if c is a valid decryption of m (under the given public key). Note that the public evaluation of the HPS can use the encryption randomness as a witness. These kinds of HPS have been constructed for ElGamal and Cramer-Shoup encryption in the prime-order group setting (e.g., [BBC+13]), however it is less clear how this will work for group actions. We illustrate this for the simpler example of the "group action ElGamal" encryption scheme. The main obstacle here is that due to the limited structure we cannot simply encrypt the message by a one-time pad like operation (see for example [MOT20]). Instead, one can additionally hash the set element that serves as the ElGamal KEM key and then encrypt the message via XOR. However, this destroys all structure and makes it hard to build a hash proof system for ciphertexts of this form. Therefore, we leave it as an interesting open problem to build such an HPS from group actions which can then be used to construct PAKE.

It is well known that PAKE can also be generically constructed from OT. In [CDVW12], Canetti et al. describe two different constructions: the first builds upon a UC-secure OT protocol to construct a UC-secure PAKE and the second uses a statistically receiver-private OT protocol to construct PAKE in a game-based security model. In both constructions, the password is interpreted as a bit string. In particular, for each individual password bit, the PAKE user and server run the OT protocol twice: once taking the role of the OT sender for randomly chosen messages and once taking the role of the OT receiver using the password bit to recover one of the messages chosen by the other party. Together with some additional overhead consisting of nonces and/or ciphertexts that need to be sent to compute the shared session key, this results in a PAKE protocol of at least three rounds. That means, even for round-optimal and efficient OT protocols, this approach makes the final construction quite inefficient. To compare against our protocols, we apply the compiler of [CDVW12] to the following two OT protocols.

- Alamati et al. propose a two-message statistically *sender*-private OT, however we can construct a similar receiver-private OT protocol based on their dual-mode public-key encryption scheme and the transformation given in [PVW08]. The resulting OT protocol already uses a "bit-by-bit" approach, hence the resulting PAKE will have communication and computation complexity quadratic in the parameter ℓ.
- Recently, Lai et al. proposed a new very efficient CSIDH-based OT protocol using twists and the random oracle model [LGd21]. However, in order to achieve active security the protocol needs four rounds.[1] Additionally applying the generic PAKE compiler results in a protocol with complexity linear in ℓ.

The efficiency of the generic constructions compared to our new protocols is given in Table 1. Note that the computational complexity of the second OT-based protocol as well as the complexity of our protocols is linear in the password length ℓ. However the constants are important for concrete instantiations. For $\ell = 128$ and $N = 8$, our optimized versions of $\mathsf{X\text{-}GA\text{-}PAKE}_{\ell,N}$ (resp. $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell,N}$) perform considerably better. In particular, each party then has to send 32 (resp. 16) set elements and perform 80 (resp. 32) group action evaluations. Whereas each party would have to send 384 set elements and perform 1408 group action evaluations in the OT-based protocol. Additionally, $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell}$ is the only one-round protocol, where both parties send simultaneous flows, which plays an important role for practical applications.

Open Problems and Future Work. Until now, protocols based on CSIDH or group actions that use search problems together with the random oracle model do not consider quantum access to the ROM [Yon19, FTY19, KTAT20, dKGV21, LGd21]. Since PAKE proofs are already complex, we also did not prove security in the QROM. Although no reprogramming of the random oracle is necessary, the main difficulty in the QROM is to simulate the real session keys using the decision oracle. We leave this as future work. We believe that we can easily allow quantum access to the additional random oracle

[1] The original (three-round) version of this protocol was later found to have a (fixable) bug, cf. https://iacr.org/submit/files/slides/2021/eurocrypt/eurocrypt2021/20/slides.pdf.

that is used in Com-GA-PAKE$_\ell$ to commit to the message. In this case, the output is transferred classically in the first message flow such that extraction is possible using recently developed techniques [DFMS21].

As [LGd21], we use rewinding to reduce the interactive assumption underlying Com-GA-PAKE$_\ell$ to a standard assumption. An interesting open question is whether current techniques enabling quantum rewinding are applicable here.

Outline. Section 3 sets the framework for our paper. We introduce (restricted) effective group actions with twists and define the computational assumptions underlying the security of our protocols. In Section 4, we give some background on the security model that is used in the subsequent sections. In Section 5 we present our first attempt for a PAKE protocol, GA-PAKE$_\ell$, and explain its security gap. Section 6 contains a thorough analysis of our new secure protocol X-GA-PAKE$_\ell$. In Section 7 we present the protocol Com-GA-PAKE$_\ell$ and sketch the security proof. A full proof is provided in Appendix E. Finally, we discuss possible optimizations of the protocols in Section 8.

2 Preliminaries

For integers m, n where $m < n$, $[m, n]$ denotes the set $\{m, m+1, ..., n\}$. For $m = 1$, we simply write $[n]$. For a set S, $s \xleftarrow{\$} S$ denotes that s is sampled uniformly and independently at random from S. $y \leftarrow \mathcal{A}(x_1, x_2, ...)$ denotes that on input $x_1, x_2, ...$ the probabilistic algorithm $\mathcal{A}$ returns y. $\mathcal{A}^{\mathrm{O}}$ denotes that algorithm $\mathcal{A}$ has access to oracle O. An adversary is a probabilistic algorithm. We will use code-based games, where $\Pr[\mathsf{G} \Rightarrow 1]$ denotes the probability that the final output of game G is 1.

3 (Restricted) Effective Group Actions (with Twists)

In this section we recall the definition of (restricted) effective group actions from [ADMP20], which provides an abstract framework to build cryptographic primitives relying on isogeny-based assumptions such as CSIDH. Moreover, we suggest an enhancement of this framework, by introducing (restricted) effective group actions with twists. This addition is essential for the security analysis of our new PAKE protocols.

Definition 1 (Group Action)**.** *Let $(\mathcal{G}, \cdot)$ be a group with identity element $id \in \mathcal{G}$, and $\mathcal{X}$ a set. A map*

$$\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$$

is a group action if it satisfies the following properties:

1. *Identity: $id \star x = x$ for all $x \in \mathcal{X}$.*
2. *Compatibility: $(g \cdot h) \star x = g \star (h \star x)$ for all $g, h \in \mathcal{G}$ and $x \in \mathcal{X}$.*

Remark 9. Throughout this paper, we only consider group actions, where $\mathcal{G}$ is commutative. Moreover we assume that the group action is regular. This means that for any $x, y \in \mathcal{X}$ there exists precisely one $g \in \mathcal{G}$ satisfying $y = g \star x$.

Definition 2 (Effective Group Action)**.** *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action satisfying the following properties:*

1. *The group $\mathcal{G}$ is finite and there exist efficient (PPT) algorithms for membership and equality testing, (random) sampling, group operation and inversion.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm to evaluate the group action, i.e. to compute $g \star x$ given g and x.*

Then we call $\tilde{x} \in \mathcal{X}$ the origin and $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ an effective group action (EGA).

In practice, the requirements from the definition of EGA are often too strong. Therefore we will consider the weaker notion of restricted effective group actions.

Definition 3 (Restricted Effective Group Action)**.** *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action and let $\boldsymbol{g} = (g_1, ..., g_n)$ be a generating set for $\mathcal{G}$. Assume that the following properties are satisfied:*

1. *The group $\mathcal{G}$ is finite and $n = poly(\log(\#\mathcal{G}))$.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm that given $g_i \in \boldsymbol{g}$ and $x \in \mathcal{X}$, outputs $g_i \star x$ and $g_i^{-1} \star x$.*

Then we call $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ a restricted effective group action (REGA).

3.1 Isogeny-based REGAs

An important instantiation of REGAs is provided by isogeny-based group actions. We will focus on the CSIDH setting and present a refined definition of REGAs tailored to this situation.

Let p be a large prime of the form $p = 4 \cdot \ell_1 \cdots \ell_n - 1$, where the ℓ_i are small distinct odd primes. Fix the elliptic curve $E_0 : y^2 = x^3 + x$ over $\mathbb{F}_p$. The curve E_0 is supersingular and its $\mathbb{F}_p$-rational endomorphism ring is $\mathcal{O} = \mathbb{Z}[\pi]$, where π is the Frobenius endomorphism. Let $\mathcal{E}\ell\ell_p(\mathcal{O})$ be the set of elliptic curves defined over $\mathbb{F}_p$, with endomorphism ring $\mathcal{O}$. The ideal class group $cl(\mathcal{O})$ acts on the set $\mathcal{E}\ell\ell_p(\mathcal{O})$, i.e., there is a map

$$\begin{aligned} \star : cl(\mathcal{O}) \times \mathcal{E}\ell\ell_p(\mathcal{O}) &\to \mathcal{E}\ell\ell_p(\mathcal{O}) \\ ([\mathfrak{a}], E) &\mapsto [\mathfrak{a}] \star E, \end{aligned}$$

satisfying the properties from Definition 1 [CLM+18, Theorem 7]. Moreover the analysis in [CLM+18] readily shows that $(cl(\mathcal{O}), \mathcal{E}\ell\ell_p(\mathcal{O}), \star, E_0)$ is indeed a REGA.

Elliptic curves in $\mathcal{E}\ell\ell_p(\mathcal{O})$ admit equations of the form $E_A : y^2 = x^3 + Ax^2 + x$, which allows to represent them by their Montgomery coefficient $A \in \mathbb{F}_p$. An intrinsic property of the CSIDH group action which is not covered by Definition 3, is the following. For any curve $E_A = [\mathfrak{a}] \star E_0 \in \mathcal{E}\ell\ell_p(\mathcal{O})$, its quadratic twist is easily computed as $(E_A)^t = E_{-A}$ and satisfies the property $(E_A)^t = [\mathfrak{a}]^{-1} \star E_0$.

Definition 4 ((Restricted) Effective Group Action with Twists)**.** *We say that a* (R)EGA $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ *is a (Restricted) Effective Group Action with Twists* ((R)EGAT) *if there exists an efficient algorithm that given* $x = g \star \tilde{x} \in \mathcal{X}$ *computes* $x^t = g^{-1} \star \tilde{x}$*.*

As noted in [CLM+18, §10], this property contrasts with the classical group-based setting. It has already been used for the design of new cryptographic primitives based on CSIDH such as the signature scheme CSIFiSh [BKV19] and the OT protocol in [LGd21]. Moreover, it is important to consider twists in the security analysis of schemes based on group actions. In Section 5 we use twists to construct an attack on the protocol GA-PAKE$_\ell$ showing that it cannot be securely instantiated with the CSIDH group action. On the other hand, we prove that GA-PAKE$_\ell$ is secure when instantiated with a group action without efficient twisting (Theorem 3).

3.2 Computational Assumptions

For cryptographic applications, we are interested in (restricted) effective group actions that are equipped with the following hardness properties:

- Given $(x, y) \in \mathcal{X}^2$, it is hard to find $g \in \mathcal{G}$ such that $y = g \star x$.
- Given $(x, y_0, y_1) \in \mathcal{X}^3$, it is hard to find $z = (g_0 \cdot g_1) \star x$, where $g_0, g_1 \in \mathcal{G}$ are such that $y_0 = g_0 \star x$ and $y_1 = g_1 \star x$.

In [ADMP20] such group actions are called cryptographic group actions, and in [Cou06] they are called hard homogeneous spaces.

The two hardness assumptions are the natural generalizations of the discrete logarithm assumption and the Diffie-Hellman assumption in the traditional group based setting. In analogy to this setting, we introduce the notation

$$\text{GA-CDH}_x(y_0, y_1) = g_0 \star y_1, \qquad \text{where } g_0 \in \mathcal{G} \text{ such that } y_0 = g_0 \star x$$

and define the decision oracle

$$\text{GA-DDH}_x(y_0, y_1, z) = \begin{cases} 1 & \text{if GA-CDH}_x(y_0, y_1) = z, \\ 0 & \text{otherwise.} \end{cases}$$

For both, GA-CDH and GA-DDH, we omit the index x if $x = \tilde{x}$, i. e., we set $\text{GA-CDH}_{\tilde{x}}(y_0, y_1) = \text{GA-CDH}(y_0, y_1)$ and $\text{GA-DDH}_{\tilde{x}}(y_0, y_1, z) = \text{GA-DDH}(y_0, y_1, z)$.

We now introduce three computational problems GA-StCDH, GA-GapCDH, SqInv-GA-StCDH (Definitions 5 to 7). The security of our PAKE protocols relies on the hardness of these problems.

The first two problems are variants of the standard Diffie-Hellman problem, where an adversary is either given access to some fixed-basis decision oracles (indicated by the prefix *strong*) or to a general decision oracle (indicated by the prefix *gap*). Note that these problems were already defined and used in previous work [Yon19, FTY19, KTAT20, dKGV21]. In contrast the problem from Definition 7 has not been studied in any previous work. Therefore, we provide evidence for its hardness in Remark 11.

Definition 5 (Group Action Strong Computational Diffie-Hellman Problem (GA-StCDH))**.** *On input* $(g \star \tilde{x}, h \star \tilde{x}) \in \mathcal{X}^2$*, the* GA-StCDH *problem requires to compute the*

set element $(g \cdot h) \star \tilde{x}$*. To an effective group action* $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we associate the advantage function of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr[\mathcal{A}^{\text{GA-DDH}(g\star\tilde{x},\cdot,\cdot)}(g \star \tilde{x}, h \star \tilde{x}) \Rightarrow (g \cdot h) \star \tilde{x}] \ ,$$

where $(g, h) \xleftarrow{\$} \mathcal{G}^2$ *and* $\mathcal{A}$ *has access to decision oracle* $\text{GA-DDH}(g \star \tilde{x}, \cdot, \cdot)$*.*

Definition 6 (Group Action Gap Computational Diffie-Hellman Problem (GA-GapCDH))**.** *On input* $(g\star\tilde{x}, h\star\tilde{x}) \in \mathcal{X}^2$*, the* GA-GapCDH *problem requires to compute the set element* $(g\cdot h)\star\tilde{x}$*. To an effective group action* $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we associate the advantage function of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr[\mathcal{A}^{\text{GA-DDH}_*}(g \star \tilde{x}, h \star \tilde{x}) \Rightarrow (g \cdot h) \star \tilde{x}] \ ,$$

where $(g, h) \xleftarrow{\$} \mathcal{G}^2$ *and* $\mathcal{A}$ *has access to a general decision oracle* GA-DDH_**.*

Remark 10. A group action where the group action computational Diffie-Hellman problem (without any decision oracle) is hard, is the same as a weak unpredictable group action as defined by Alamati et al. [ADMP20]. Further details are given in Appendix A. Also note that the ability to compute the twist of a set element does not help in solving these problems. Hence, all results based on these problems remain true for (R)EGAT.

Definition 7 (Square-Inverse GA-StCDH (SqInv-GA-StCDH))**.** *On input* $x = g \star \tilde{x}$*, the* SqInv-GA-StCDH *problem requires to find a tuple* $(y, y_0, y_1) \in \mathcal{X}^3$ *such that* $y_0 = g^2 \star y$ *and* $y_1 = g^{-1} \star y$*. For a group action* $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{matrix} y_0 = \mathsf{GA\text{-}CDH}_{x^t}(x, y) \\ y_1 = \mathsf{GA\text{-}CDH}(x^t, y) \end{matrix} \;\middle|\; \begin{matrix} g \xleftarrow{\$} \mathcal{G} \\ x = g \star \tilde{x} \\ (y, y_0, y_1) \leftarrow \mathcal{A}^{\mathrm{O}}(x) \end{matrix}\right] \ ,$$

where $\mathrm{O} = \{\text{GA-DDH}_{x^t}(x, \cdot, \cdot), \text{GA-DDH}(x, \cdot, \cdot)\}$*.*

Remark 11. Intuitively SqInv-GA-StCDH is hard if we assume that the adversary can only use the group and twist operation. To go into more detail, $\mathcal{A}$ can choose y only based on known elements, that is either based on $\tilde{x}$, its input x or x^t.

If $\mathcal{A}$ chooses $y = \alpha \star \tilde{x}$ for some $\alpha \in \mathcal{G}$, then it can easily compute $y_1 = \alpha \star x^t$, but not $y_0 = \alpha g^2 \star \tilde{x}$. If $\mathcal{A}$ chooses $y = \alpha \star x$, then computing $y_1 = \alpha \star \tilde{x}$ is trivial, but computing $y_0 = \alpha g^3 \star \tilde{x}$ is hard. If $\mathcal{A}$ chooses $y = \alpha \star x^t$, then computing $y_0 = \alpha \star x$ is trivial, but computing $y_1 = \alpha g^{-2} \star \tilde{x}$ is hard.

4 Password Authenticated Key Exchange

Password-authenticated key exchange (PAKE) allows two parties, typically referred to as the user and the server, to establish a shared session key with the help of a short secret, known as a password, which can be drawn from a small set of possible values. To prove security of a PAKE protocol, we use the indistinguishability-based security model by Bellare, Pointcheval and Rogaway [BPR00] and its extension to multiple test

queries by Abdalla, Fouque and Pointcheval [AFP05]. In our proofs, we further adapt the game-based pseudocode used in [AB19].

The name spaces for users $\mathcal{U}$ and servers $\mathcal{S}$ are assumed to be disjoint. Each pair of user and server $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$ holds a shared password $\mathsf{pw}_{\mathsf{US}}$. A party P denotes either a user or server. Each party P has multiple instances π^i_{P} and each instance has its own state. We denote the session key space by $\mathcal{K}$. Passwords are bit strings of length ℓ and we define the password space as $\mathcal{PW} \subsetneq \{0,1\}^\ell$.

Instance State. The state of an instance π^i_{P} is a tuple $(\mathsf{e}, \mathsf{tr}, K, \mathrm{acc})$ where
- e stores the (secret) ephemeral values chosen by the party in that instance (in our case group elements).
- tr stores the trace of that instance, i.e., the user and server name involved in the protocol execution and the messages sent and received by that instance.
- K is the accepted session key.
- acc is a Boolean flag that indicates whether the instance has accepted the session key. As long as the instance did not receive the last message, $\mathrm{acc} = \bot$.

To access individual components of the state, we write $\pi^t_{\mathsf{P}}.\{\mathsf{e}, \mathsf{tr}, K, \mathrm{acc}\}$.

Partnering. Partnering is defined via matching conversations. In particular, a user instance $\pi^{t_0}_{\mathsf{U}}$ and a server instance $\pi^{t_1}_{\mathsf{S}}$ are partnered iff

$$\pi^{t_0}_{\mathsf{U}}.\mathrm{acc} = \pi^{t_1}_{\mathsf{S}}.\mathrm{acc} = \textbf{true} \quad \textbf{and} \quad \pi^{t_0}_{\mathsf{U}}.\mathsf{tr} = \pi^{t_1}_{\mathsf{S}}.\mathsf{tr} \ .$$

Two user instances are never partnered, neither are two server instances. We define a partner predicate $\mathsf{Partner}(\pi^{t_0}_{\mathsf{P}_0}, \pi^{t_1}_{\mathsf{P}_1})$ which outputs 1 if the two instances $\pi^{t_0}_{\mathsf{P}_0}$ and $\pi^{t_1}_{\mathsf{P}_1}$ are partnered and 0 otherwise.

Security Experiment. The security experiment is played between a challenger and an adversary $\mathcal{A}$. The challenger draws a random challenge bit β and creates the public parameters. Then it outputs the public parameters to $\mathcal{A}$. Now $\mathcal{A}$ has access to the following oracles:
- $\textsc{Execute}(\mathsf{U}, t_0, \mathsf{S}, t_1)$: one complete protocol execution between user instance $\pi^{t_0}_{\mathsf{U}}$ and server instance $\pi^{t_1}_{\mathsf{S}}$. This query models security against passive adversaries.
- $\textsc{SendInit}$, $\textsc{SendResp}$, $\textsc{SendTermInit}$, $\textsc{SendTermResp}$: send oracles to model security against active adversaries. $\textsc{SendTermResp}$ is only available for three-message protocols.
- $\textsc{Corrupt}(\mathsf{U}, \mathsf{S})$: outputs the shared password $\mathsf{pw}_{\mathsf{US}}$ of U and S.
- $\textsc{Reveal}(\mathsf{P}, t)$: outputs the session key of instance π^t_{P}.
- $\textsc{Test}(\mathsf{P}, t)$: challenge query. Depending on the challenge bit β, the experiment outputs either the session key of instance π^t_{P} or a uniformly random key. By $\pi^t_{\mathsf{P}}.\mathrm{test} = \textbf{true}$, we mark an instance as tested.

We denote the experiment by $\mathsf{Exp}_{\mathsf{PAKE}}$. The pseudocode is given in G_0 in Figure 5, instantiated with our first PAKE protocol.

Freshness. During the game, we register if a query is allowed to prevent trivial wins. Therefore, we define a freshness predicate $\mathsf{Fresh}(\mathsf{P}, i)$. An instance π^t_{P} is fresh iff

1. π_{P}^t accepted.
2. π_{P}^t was not queried to Test or Reveal before.
3. At least one of the following conditions holds:
   - 3.1 π_{P}^t accepted during a query to Execute.
   - 3.2 There exists more than one partner instance.
   - 3.3 A unique fresh partner instance exists.
   - 3.4 No partner exists and Corrupt was not queried.

Definition 8 (Security of PAKE)**.** *We define the security experiment, partnering and freshness conditions as above. The advantage of an adversary $\mathcal{A}$ against a password authenticated key exchange protocol* PAKE *in* $\mathsf{Exp}_{\mathsf{PAKE}}$ *is defined as*

$$\mathsf{Adv}_{\mathsf{PAKE}}(\mathcal{A}) \coloneqq \left|\Pr[\mathsf{Exp}_{\mathsf{PAKE}} \Rightarrow 1] - \frac{1}{2}\right| \ .$$

A PAKE is considered secure if the best the adversary can do is to perform an online dictionary attack. More concretely, this means that the advantage of the adversary should be negligibly close to $q_s/|\mathcal{PW}|$ when passwords are drawn uniformly and independently from $\mathcal{PW}$, where q_s is the number of send queries made by the adversary.

Note that this definition captures weak forward secrecy. We will give an extended security definition capturing also perfect forward secrecy in Appendix F, as well as proofs for our protocols.

5 First Attempt: Protocol GA-PAKE$_\ell$

The GA-PAKE$_\ell$ protocol was already introduced in the introduction (Section 1). We refer to Figure 1 for a description of the protocol. In contrast to the two PAKE protocols from Sections 7 and 6, GA-PAKE$_\ell$ is not secure for EGATs, i.e., if it is possible to compute twists of set elements efficiently. In particular it should not be instantiated with the CSIDH-group action. However, it is instructive to examine its security and it serves as a good motivation for the design of the two secure PAKE protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$.

In this section we present an offline dictionary attack against GA-PAKE$_\ell$ for (R)EGAT. However, if twisting is hard, then we can prove security of GA-PAKE$_\ell$ based on a hardness assumption that is similar to the simultaneous Diffie-Hellman problem which was introduced to prove the security of TBPEKE and CPace [PW17, AHH21]. Our proof for GA-PAKE$_\ell$ is given in Appendix C.

Proposition 1. *For* EGAT*s, the protocol* GA-PAKE$_\ell$ *is vulnerable to offline dictionary attacks.*

Proof. We construct an adversary $\mathcal{A}$ that takes the role of the server. The attack is summarized in Figure 3. After receiving x^{U}, the adversary computes

$$x_i^{\mathsf{S}} = \tilde{s}_i \star (x_i^{\mathsf{U}})^t = \tilde{s}_i \star (u_i \star x_{b_i})^t = (\tilde{s}_i \cdot u_i^{-1}) \star x_{b_i}^t = (\tilde{s}_i \cdot u_i^{-1} \cdot g_{b_i}^{-1}) \star \tilde{x}$$

for each $i \in [\ell]$ and sends $x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}$ to the user. Then the user computes $z_i = u_i \star x_i^{\mathsf{S}} = (\tilde{s}_i \cdot g_{b_i}^{-1}) \star \tilde{x} = \tilde{s}_i \star x_{b_i}^t$. For each $i \in [\ell]$, the adversary $\mathcal{A}$ can now compute z_i for both

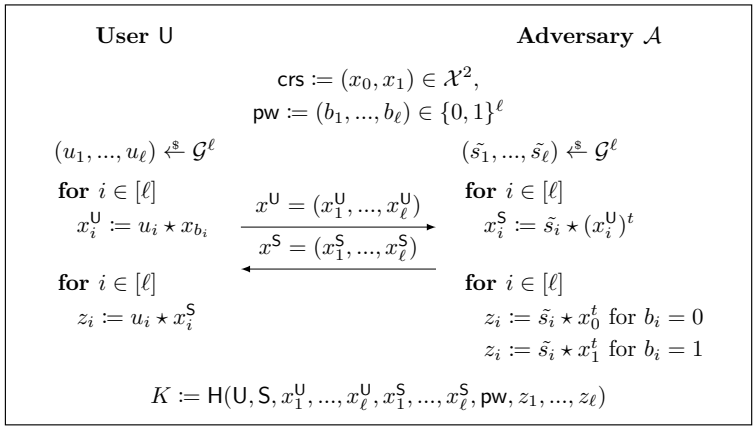

Figure 3: Attack against $\mathsf{GA\text{-}PAKE}_\ell$ using twists.

possibilities $b_i = 0$ and $b_i = 1$. This allows him to compute K for all possible passwords $\mathsf{pw} \in \mathcal{PW} \subsetneq \{0,1\}^\ell$ (being offline). □

This offline attack can easily be used to win the security experiment with high probability. $\mathcal{A}$ only needs to issue two send queries. It chooses any user U, initiates a session and computes its message $x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}$ as described in Figure 3. It reveals the corresponding session key and starts its offline attack by brute forcing all $\mathsf{pw} \in \mathcal{PW}$ until it finds a match for a candidate pw^*. Now $\mathcal{A}$ issues its second send query. This time it computes the message following the protocol using pw^* and derives a key K^*. It issues a test query and gets K_β. If $K^* = K_\beta$, then it outputs 0, otherwise it outputs 1. In case there is more than one password candidate, i.e., two inputs to H lead to the same K^*, then $\mathcal{A}$ can issue another send and reveal query to rule out false positives. In the end, it can still happen that $\beta = 1$ and $K^* = K$, but this event only occurs with probability $1/|\mathcal{K}|$.

Corollary 5. *For any adversary $\mathcal{A}$ against $\mathsf{GA\text{-}PAKE}_\ell$ instantiated with an EGAT, we have* $\Pr[\mathsf{Exp}_{\mathsf{GA\text{-}PAKE}_\ell} \Rightarrow 1] = 1 - \frac{1}{|\mathcal{K}|}$.

6 X-GA-PAKE$_\ell$: One-Round PAKE from Group Actions

In the previous section we showed that $\mathsf{GA\text{-}PAKE}_\ell$ is insecure when instantiated with an EGAT. Here, we present the modification $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$, which impedes the offline dictionary attack presented in that section. Broadly speaking, the idea is to double the message size of both parties in the first flow. In the second flow it is then necessary to compute certain "cross products" which is only possible if the previous message has been honestly generated. The letter X in $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ stands for cross product.

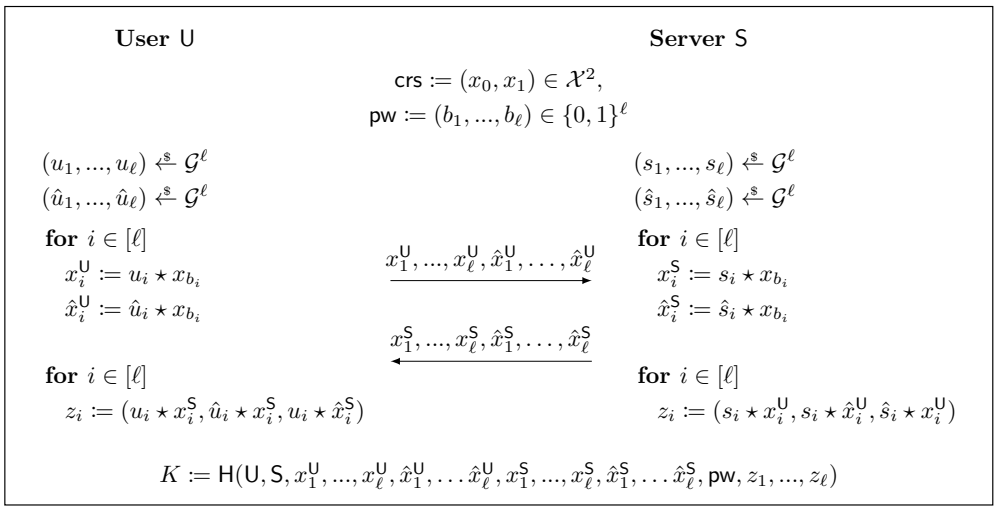

Figure 4: PAKE protocol X-GA-PAKE$_\ell$ from group actions.

By means of these modifications, the protocol X-GA-PAKE$_\ell$ is provably secure for EGATs. We show that its security can be reduced to the hardness of the computational problems GA-StCDH and SqInv-GA-StCDH (Theorem 1).

6.1 Description of the Protocol

The setup for X-GA-PAKE$_\ell$ is the same as for GA-PAKE$_\ell$. The $\mathsf{crs} = (x_0, x_1)$ comprises two elements of the set $\mathcal{X}$, and the shared password is a bit string $(b_1, \ldots, b_\ell)$ of length ℓ.

In the first flow of the protocol the user generates $2 \cdot \ell$ random group elements, $u_1, \ldots, u_\ell$ and $\hat{u}_1, \ldots, \hat{u}_\ell$. Using these elements it computes the set elements $x_i^{\mathsf{U}} = u_i \star x_{b_i}$ and $\hat{x}_i^{\mathsf{U}} = \hat{u}_i \star x_{b_i}$ for each $i \in [\ell]$ and sends these to the server.

Simultaneously, the server generates the random group elements $s_1, \ldots, s_\ell$ and $\hat{s}_1, \ldots, \hat{s}_\ell$, which it uses to compute the set elements $x_i^{\mathsf{S}} = s_i \star x_{b_i}$ and $\hat{x}_i^{\mathsf{S}} = \hat{s}_i \star x_{b_i}$ for each $i \in [\ell]$ and sends these to the user.

Upon receiving the set elements from the other party, both the server and the user compute the following three elements

$$z_{i,1} = u_i \star x_i^{\mathsf{S}} = s_i \star x_i^{\mathsf{U}}, \quad z_{i,2} = \hat{u}_i \star x_i^{\mathsf{S}} = s_i \star \hat{x}_i^{\mathsf{U}}, \quad z_{i,3} = u_i \star \hat{x}_i^{\mathsf{S}} = \hat{s}_i \star x_i^{\mathsf{U}},$$

for each $i \in [\ell]$. Finally, these elements are used to compute the session key K. The protocol is sketched in Figure 4.

6.2 Security of X-GA-PAKE$_\ell$

We now prove the security of X-GA-PAKE$_\ell$ for EGATs.

Theorem 1 (Security of X-GA-PAKE$_\ell$)**.** *For any adversary $\mathcal{A}$ against* X-GA-PAKE$_\ell$ *that issues at most q_e execute queries and q_s send queries and where* H *is modeled as*

a random oracle, there exist an adversary $\mathcal{B}_1$ *against* GA-StCDH *and an adversary* $\mathcal{B}_2$ *against* SqInv-GA-StCDH *such that*

$$\mathsf{Adv}_{\mathsf{X\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2) + \frac{q_s}{|\mathcal{PW}|} + \frac{(q_s+q_e)^2}{|\mathcal{G}|^{2\ell}} \ .$$

Before proving Theorem 1, we will introduce a new computational assumption which is tailored to the protocol.

Definition 9 (Double Simultaneous GA-StCDH (DSim-GA-StCDH))**.** *On input* $(x_0, x_1, w_0, w_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x}) \in \mathcal{X}^4$*, the* DSim-GA-StCDH *problem requires to find a tuple* $(y, y_0, y_1, y_2, y_3) \in \mathcal{X}^5$ *such that*

$$(y_0, y_1, y_2, y_3) = (g_0^{-1} \cdot h_0 \star y,\ g_0^{-1} \cdot h_1 \star y,\ g_1^{-1} \cdot h_0 \star y,\ g_1^{-1} \cdot h_1 \star y).$$

For a group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{matrix} y_0 = \mathsf{GA\text{-}CDH}_{x_0}(w_0, y) \\ y_1 = \mathsf{GA\text{-}CDH}_{x_0}(w_1, y) \\ y_2 = \mathsf{GA\text{-}CDH}_{x_1}(w_0, y) \\ y_3 = \mathsf{GA\text{-}CDH}_{x_1}(w_1, y) \end{matrix} \middle| \begin{matrix} (g_0, g_1, h_0, h_1) \xleftarrow{\$} \mathcal{G}^4 \\ (x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ (w_0, w_1) = (h_0 \star \tilde{x}, h_1 \star \tilde{x}) \\ (y, y_0, y_1, y_2, y_3) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1, w_0, w_1) \end{matrix} \right],$$

where $\mathrm{O} = \{\text{GA-DDH}_{x_j}(w_i, \cdot, \cdot)\}_{i,j \in \{0,1\}}$.

Remark 12. Note that DSim-GA-StCDH may be viewed as the doubled version of the Sim-GA-StCDH problem defined in the appendix (cf. Definition 12). The latter is an assumption underlying the security of $\mathsf{GA\text{-}PAKE}_\ell$ and (in the notation of the above problem) it only requires to find the tuple (y, y_0, y_2). For a group action with twists, this admits the trivial solution $(y, y_0, y_2) = (w_0^t, x_0^t, x_1^t)$. Such a trivial solution is inhibited by requiring to find y_1 and y_3 as well.

Lemma 1. *In the* EGAT *setting, the square-inverse* GA-StCDH (SqInv-GA-StCDH) *implies the double simultaneous* GA-StCDH (DSim-GA-StCDH)*. In particular,*

$$\mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}) \ .$$

Proof. Given a challenge $x = g \star \tilde{x} \in \mathcal{X}$ and oracles $\text{GA-DDH}(x, \cdot, \cdot)$, $\text{GA-DDH}_{x^t}(x, \cdot, \cdot)$ for the SqInv-GA-StCDH problem, we choose three group elements $\alpha, \beta, \gamma \xleftarrow{\$} \mathcal{G}$ and call the adversary for the DSim-GA-StCDH problem on input

$$(x_0, x_1, w_0, w_1) = (x^t, \alpha \star x, \beta \star x, \gamma \star \tilde{x}).$$

The corresponding decision oracles can be simulated using the oracles provided by SqInv-GA-StCDH. More precisely, for any $z_1, z_2 \in \mathcal{X}$:

$$\begin{aligned}
&\text{GA-DDH}_{x_0}(w_0, z_1, z_2) = \text{GA-DDH}_{x_t}(x, z_1, \beta^{-1} \star z_2), \\
&\text{GA-DDH}_{x_0}(w_1, z_1, z_2) = \text{GA-DDH}(x, z_1, \gamma^{-1} \star z_2), \\
&\text{GA-DDH}_{x_1}(w_0, z_1, z_2) = \text{GA-DDH}(\tilde{x}, z_1, (\alpha \cdot \beta^{-1}) \star z_2), \\
&\text{GA-DDH}_{x_0}(w_1, z_1, z_2) = \text{GA-DDH}(x, z_1^t, (\alpha^{-1} \cdot \gamma) \star z_2^t).
\end{aligned}$$

For the third oracle note that GA-DDH$(\tilde{x}, z_1, (\alpha \cdot \beta^{-1}) \star z_2) = 1$ precisely when $z_1 = (\alpha \cdot \beta^{-1}) \star z_2$. For the forth oracle, note that GA-DDH$_{x_0}(w_1, z_1, z_2) = 1$ iff $z_2 = (\alpha^{-1} \cdot \gamma \cdot g^{-1}) \star z_1$. This implies

$$z_2^t = (\alpha \cdot \gamma^{-1} \cdot g) \star z_1^t = (\alpha \cdot \gamma^{-1}) \cdot \mathsf{GA\text{-}CDH}(x, z_1^t).$$

If the adversary is successful, it returns a tuple (y, y_0, y_1, y_2, y_3), where

$$y_0 = g(\beta g) \star y,\ y_1 = g \cdot \gamma \star y,\ y_2 = \alpha^{-1} \cdot \beta \star y,\ y_3 = (\alpha g)^{-1}\gamma \star y.$$

Consequently, the tuple $(y, y_0', y_1') = (y, \beta^{-1} \star y_0, \alpha \star y_3)$ solves the SqInv-GA-StCDH problem. □

In the following, we give the full proof of Theorem 1.

Proof of Theorem 1. Let $\mathcal{A}$ be an adversary against X-GA-PAKE$_\ell$. Consider the games in Figures 5, 8, 9.

Game G_0. This is the original game, hence

$$\mathsf{Adv}_{\mathsf{X\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq |\Pr[\mathsf{G}_0 \Rightarrow 1] - 1/2| \ .$$

Game G_1. In game G_1, we raise flag $\mathbf{bad}_{\text{coll}}$ whenever a server instance computes the same trace as any other accepted instance (line 60) or a user instance computes the same trace as any other accepted user instance (line 75). In this case, SendResp or SendTermInit return $\perp$. We do the same if a trace that is computed in an Execute query collides with one of a previously accepted instance (line 28). Due to the difference lemma,

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{coll}}] \ .$$

Note that when $\mathbf{bad}_{\text{coll}}$ is not raised, each instance is unique and has at most one partner. In order to bound $\mathbf{bad}_{\text{coll}}$, recall that the trace of an oracle π_{P}^t consists of $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}} = (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}), \hat{x}^{\mathsf{U}} = (\hat{x}_1^{\mathsf{U}}, ...\hat{x}_\ell^{\mathsf{U}}), x^{\mathsf{S}} = (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}), \hat{x}^{\mathsf{S}} = (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}))$, where at least one of the message pairs $(x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$ or $(x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ was chosen by the game. Thus, $\mathbf{bad}_{\text{coll}}$ can only happen if all those $2 \cdot \ell$ set elements collide with all $2 \cdot \ell$ set elements of another instance. The probability that this happens for two (fixed) sessions is $|\mathcal{G}|^{-2\ell}$, hence the union bound over q_e and q_s sessions yields

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{coll}}] \leq \binom{q_e + q_s}{2} \cdot \frac{1}{|\mathcal{G}|^{2\ell}} \leq \frac{(q_e + q_s)^2}{|\mathcal{G}|^{2\ell}} \ .$$

Game G_2. In game G_2, we make the freshness explicit. To each oracle π_{P}^t, we assign an additional variable $\pi_{\mathsf{P}}^t.\mathsf{fr}$ which is updated during the game. In particular, all instances used in execute queries are marked as fresh (line 34).

An instance is fresh if the password was not corrupted yet (lines 63, 80). Otherwise, it is not fresh (lines 65, 82). For user instances we also check if there exists a fresh partner (line 78). If $\mathcal{A}$ issues a Corrupt query later, the freshness variable will also be

GAMES G_0-G_4

00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T) \coloneqq (\emptyset, \emptyset)$
03 $\mathbf{bad}_{\text{coll}} \coloneqq \mathbf{false}$
04 $\beta \xleftarrow{\$} \{0,1\}$
05 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
06 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
07 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
08 **return** $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$

09 **if** $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
10 $\quad$ **return** $\bot$
11 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$ // G_0-G_3
12 $u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
13 $\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
14 $s \coloneqq (s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
15 $\hat{s} \coloneqq (\hat{s}_1, ..., \hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
16 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$ // G_0-G_3
17 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star x_{b_1}, ..., \hat{u}_\ell \star x_{b_\ell})$ // G_0-G_3
18 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$ // G_0-G_3
19 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star x_{b_1}, ..., \hat{s}_\ell \star x_{b_\ell})$ // G_0-G_3
20 **for** $i \in [\ell]$: // G_0-G_3
21 $\quad z_i \coloneqq (z_{i,1}, z_{i,2}, z_{i,3})$
$\quad\quad \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$ // G_0-G_3
22 $z \coloneqq (z_1, \ldots, z_\ell)$ // G_0-G_3
23 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$ // G_4
24 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star \tilde{x}, ..., \hat{u}_\ell \star \tilde{x})$ // G_4
25 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$ // G_4
26 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star \tilde{x}, ..., \hat{s}_\ell \star \tilde{x})$ // G_4
27 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t.
$\quad \pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ // G_1-G_4
28 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_4
29 $\quad$ **return** $\bot$ // G_1-G_4
30 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$ // G_0-G_2
31 $K \xleftarrow{\$} \mathcal{K}$ // G_3-G_4
32 $\pi_{\mathsf{U}}^{t_0} \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
33 $\pi_{\mathsf{S}}^{t_1} \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
34 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$ // G_2-G_4
35 **return** $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

REVEAL(P, t)

36 **if** $\pi_{\mathsf{P}}^t.\mathrm{acc} \neq \mathbf{true}$ **or** $\pi_{\mathsf{P}}^t.\mathrm{test} = \mathbf{true}$
37 $\quad$ **return** $\bot$
38 **if** $\exists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
$\quad$ **and** $\pi_{\mathsf{P}'}^{t'}.\mathrm{test} = \mathbf{true}$
39 $\quad$ **return** $\bot$
40 $\forall (\mathsf{P}', t')$ s. t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = \pi_{\mathsf{P}}^t.\mathsf{tr}$ // G_2-G_4
41 $\quad \pi_{\mathsf{P}'}^{t'}.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
42 **return** $\pi_{\mathsf{P}}^t.K$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$

43 **if** $\pi_{\mathsf{U}}^t \neq \bot$
44 $\quad$ **return** $\bot$
45 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
46 $u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
47 $\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
48 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
49 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star x_{b_1}, ..., \hat{u}_\ell \star x_{b_\ell})$
50 $\pi_{\mathsf{U}}^t \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
51 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
52 **return** $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$

53 **if** $\pi_{\mathsf{S}}^t \neq \bot$
54 $\quad$ **return** $\bot$
55 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
56 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
57 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$
58 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star x_{b_1}, ..., \hat{s}_\ell \star x_{b_\ell})$
59 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t.
$\quad \pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ // G_1-G_4
60 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_4
61 $\quad$ **return** $\bot$ // G_1-G_4
62 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ // G_2-G_4
63 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_2-G_4
64 **else** // G_2-G_4
65 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
66 **for** $i \in [\ell]$:
67 $\quad z_i \coloneqq (z_{i,1}, z_{i,2}, z_{i,3})$
$\quad\quad \coloneqq (s_i \star x_i^{\mathsf{U}}, s \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \star x_i^{\mathsf{U}})$
68 $z \coloneqq (z_1, ..., z_\ell)$
69 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
70 $\pi_{\mathsf{S}}^t \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
71 **return** $(\mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

72 **if** $\pi_{\mathsf{U}}^t \neq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
73 $\quad$ **return** $\bot$
74 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ // G_1-G_4
75 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_4
76 $\quad$ **return** $\bot$ // G_1-G_4
77 **if** $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
$\quad$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$ // G_2-G_4
78 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_2-G_4
79 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ // G_2-G_4
80 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_2-G_4
81 **else** // G_2-G_4
82 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
83 **for** $i \in [\ell]$:
84 $\quad z_i \coloneqq (z_{i,1}, z_{i,2}, z_{i,3})$
85 $\quad\quad \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$
86 $z \coloneqq (z_1, ..., z_\ell)$
87 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
88 $\pi_{\mathsf{U}}^t \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}} x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
89 **return true**

Figure 5: Games G_0-G_4 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{H}\}$, where oracles TEST, CORRUPT and H are defined in Figure 6.

$\underline{\textsc{Test}(\mathsf{P}, t)}$
00 **if** $\mathsf{Fresh}(\pi_{\mathsf{P}}^t) = \textbf{false}$ **return** $\bot$ // G_0-G_1
01 **if** $\pi_{\mathsf{P}}^t.\mathsf{fr} = \textbf{false}$ **return** $\bot$ // G_2-G_4
02 $K_0^* \coloneqq \textsc{Reveal}(\mathsf{P}, t)$
03 **if** $K_0^* = \bot$ **return** $\bot$
04 $K_1^* \xleftarrow{\$} \mathcal{K}$
05 $\pi_{\mathsf{P}}^t.\mathrm{test} \coloneqq \textbf{true}$
06 **return** K_β^*

$\underline{\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z)}$
07 **if** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$
08 **return** K
09 $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, Z] \xleftarrow{\$} \mathcal{K}$
10 **return** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z]$

$\underline{\textsc{Corrupt}(\mathsf{U}, \mathsf{S})}$
11 **if** $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
12 **for** $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
13 **if** $\exists t$ s. t. $\pi_{\mathsf{P}}^t.\mathrm{test} = \textbf{true}$
and $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
14 **return** $\bot$
15 $\forall \pi_{\mathsf{P}}^t$: **if** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$ // G_2-G_4
16 $\pi_{\mathsf{P}}^t.\mathsf{fr} = \textbf{false}$ // G_2-G_4
17 $\mathcal{C} \coloneqq \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
18 **return** $\mathsf{pw}_{\mathsf{US}}$

Figure 6: Oracles Test, Corrupt and H for games G_0-G_4 in Figure 5.

updated (line 16). When the session key of an instance is revealed, this instance and its potential partner instance are marked as not fresh (line 41). On a query to test, the game then only checks the freshness variable (line 01). These are only a conceptual changes, hence

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1] \ .$$

Game G_3. In game G_3, we choose random keys for instances queried to Execute. We construct adversary $\mathcal{B}_1$ against GA-StCDH in Figure 7 and show that

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}_{\mathsf{EGAT}}^{\mathsf{GA\text{-}StCDH}}(\mathcal{B}_1) \ .$$

Adversary $\mathcal{B}_1$ inputs a GA-StCDH challenge $(x, y) = (g \star \tilde{x}, h \star \tilde{x})$ and has access to a decision oracle GA-DDH$(x, \cdot, \cdot)$. First, it generates the crs elements (x_0, x_1) as in game G_3 and then runs adversary $\mathcal{A}$. Queries to Execute are simulated as follows: It chooses random group elements $u_i, \hat{u}_i$ and $s_i, \hat{s}_i$ for user and server instances and $i \in [\ell]$, but instead of using (x_0, x_1) to compute the set elements, $\mathcal{B}_1$ uses x for the user instance and y for the server instance, independent of the password bits b_i (lines 30-33). We can rewrite this as

$$x_i^{\mathsf{U}} = u_i \star x = (u_i \cdot g) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i} \cdot g_{b_i}^{-1}) \star \tilde{x} = \underbrace{(u_i \cdot g \cdot g_{b_i}^{-1})}_{u_i'} \star x_{b_i} \ ,$$

where u_i' is the group element that the user actually needs in order to compute the session key. In the same way, $\hat{u}_i' = \hat{u}_i \cdot g \cdot g_{b_i}^{-1}$, $s_i' = s_i \cdot h \cdot g_{b_i}^{-1}$ and $\hat{s}_i' = \hat{s}_i \cdot h \cdot g_{b_i}^{-1}$. Note that $z_i = (z_{i,1}, z_{i,2}, z_{i,3})$ is implicitly set to

$$\begin{aligned} z_{i,1} &= (u_i' \cdot s_i') \star x_{b_i} = u_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ , \\ z_{i,2} &= (\hat{u}_i' \cdot s_i') \star x_{b_i} = \hat{u}_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ , \\ z_{i,3} &= (u_i' \cdot \hat{s}_i') \star x_{b_i} = u_i \cdot g \cdot \hat{s}_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ . \end{aligned}$$

Before choosing a random session key, we check if there has been a query to the random oracle H that matches the session key (lines 37-45). We iterate over the entries in T,

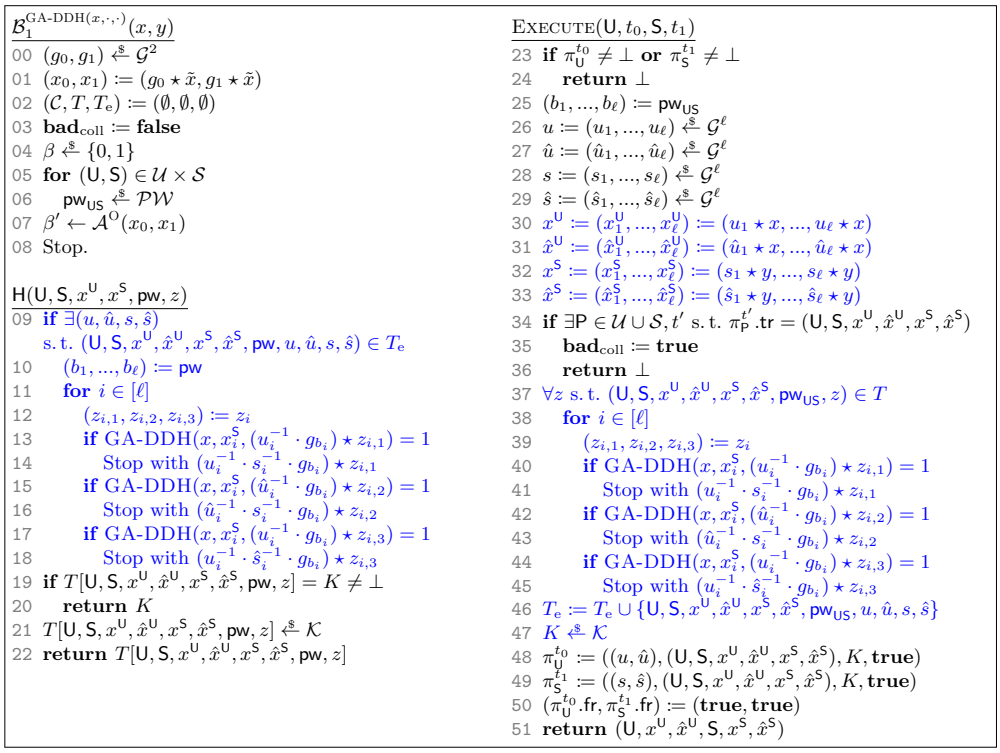

Figure 7: Adversary $\mathcal{B}_1$ against GA-StCDH for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, H$\}$. Oracles SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT and TEST are defined as in G_2. Lines written in blue show how $\mathcal{B}_1$ simulates the game.

where U, S, x^{U}, $\hat{x}^{\mathsf{U}}$, x^{S}, $\hat{x}^{\mathsf{S}}$ and $\mathsf{pw}_{\mathsf{US}}$ match, and check if one of the entries in z is correct. Note that we can use the following equivalences:

$$\begin{aligned}
\mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}},x_i^{\mathsf{S}}) = z_{i,1} \quad &\Leftrightarrow \quad \mathsf{GA\text{-}CDH}(x,x_i^{\mathsf{S}}) = (u_i^{-1}\cdot g_{b_i})\star z_{i,1},\\
\mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{U}},x_i^{\mathsf{S}}) = z_{i,2} \quad &\Leftrightarrow \quad \mathsf{GA\text{-}CDH}(x,x_i^{\mathsf{S}}) = (\hat{u}_i^{-1}\cdot g_{b_i})\star z_{i,2},\\
\mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}},\hat{x}_i^{\mathsf{S}}) = z_{i,3} \quad &\Leftrightarrow \quad \mathsf{GA\text{-}CDH}(x,\hat{x}_i^{\mathsf{S}}) = (u_i^{-1}\cdot g_{b_i})\star z_{i,3},
\end{aligned}$$

which allows us to use the restricted decision oracle GA-DDH$(x,\cdot,\cdot)$. If one of $z_{i,1},z_{i,2},z_{i,3}$ is correct, $\mathcal{B}_1$ aborts and outputs the solution $(g\cdot h)\star\tilde{x}$ which is respectively given by $(u_i^{-1}\cdot s_i^{-1}\cdot g_{b_i})\star z_{i,1}$, $(\hat{u}_i^{-1}\cdot s_i^{-1}\cdot g_{b_i})\star z_{i,2}$ or $(u_i^{-1}\cdot \hat{s}_i^{-1}\cdot g_{b_i})\star z_{i,3}$.

Otherwise, we store the values $u_i,\hat{u}_i$ and $s_i,\hat{s}_i$ in list T_e together with the trace and the password (line 46) and choose a session key uniformly at random. We need list T_e to identify relevant queries to H. In particular, if the trace and password appear in a query, we retrieve the values $u_i,\hat{u}_i$ and $s_i,\hat{s}_i$ to check whether the provided z_i are correct. We do this in the same way as described above using the decision oracle (lines 09-18). If the oracle returns 1 for any $z_{i,j}$, $\mathcal{B}_1$ aborts and outputs the solution for $(g\cdot h)\star\tilde{x}$ which is respectively given by $(u_i^{-1}\cdot s_i^{-1}\cdot g_{b_i})\star z_{i,1}$, $(\hat{u}_i^{-1}\cdot s_i^{-1}\cdot g_{b_i})\star z_{i,2}$ or $(u_i^{-1}\cdot \hat{s}_i^{-1}\cdot g_{b_i})\star z_{i,3}$.

GAME G_4. In game G_4, we remove the password from execute queries. In particular, we do not compute $x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}$ to the basis x_{b_i}, but simply use $\tilde{x}$. Note that the values have the same distribution as in the previous game. Also, the group elements $u, \hat{u}, s$ and $\hat{s}$ are not used to to derive the key. Hence, this change is not observable by $\mathcal{A}$ and

$$\Pr[\mathsf{G}_4 \Rightarrow 1] = \Pr[\mathsf{G}_3 \Rightarrow 1] \ .$$

GAME G_5. G_5 is given in Figure 8. In this game we want to replace the session keys by random for all fresh instances in oracles SENDRESP and SENDTERMINIT (lines 61, 82). Therefore, we introduce an additional independent random oracle T_{s} which maps only the trace of an instance to a key (lines 62, 83). We keep partner instances consistent, i.e., in case the adversary queries SENDTERMINIT for a user instance and there exists a fresh partner instance, then we retrieve the corresponding key from T_{s} and also assign it to this instance (line 77). For all instances that are not fresh, we simply compute the correct key using random oracle H (lines 65-68, 86-89). If a session is fresh and there is an inconsistency between T and T_{s}, we raise flag **bad**. This happens in the following cases:

- a server instance is about to compute the session key, the password was not corrupted, but there already exists an entry in T with the correct password and z (lines 59-60).
- a user instance is about to compute the session key, there exists no partner instance and the password was not corrupted, but there already exists an entry in T with the correct password and z (lines 80-81).
- the random oracle is queried on some trace that appears in T_{s} together with the correct password and z (lines 35-46). At this point, we also check if the password was corrupted in the meantime and if this is the case and the adversary issues the correct query, we simply output the key stored in T_{s} (line 45) as this instance cannot be tested. This case corresponds to perfect forward secrecy which we cover in Appendix F.2.

When **bad** is not raised, there is no difference between G_4 and G_5. Hence,

$$|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \ .$$

GAME G_6. G_6 is given in Figure 9. In this game we remove the password from send queries and generate passwords as late as possible, that is either when the adversary issues a corrupt query (line 21) or after it has stopped with output β' (line 07). In SENDINIT and SENDRESP we still choose group elements $u_i, \hat{u}_i, s_i$ and $\hat{s}_i$ uniformly at random, but now compute $x_i^{\mathsf{U}}, \hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}$ and $\hat{x}_i^{\mathsf{S}}$ using the origin element (lines 26-27 and 50-51). Thus, depending on which password is chosen afterwards, we implicitly set

$$x_i^{\mathsf{U}} = u_i \cdot \tilde{x} = (u_i \cdot g_0^{-1}) \star x_0 = (u_i \cdot g_1^{-1}) \star x_1$$

and analogously for $\hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}$ and $\hat{x}_i^{\mathsf{S}}$. For all instances that are not fresh, we have to compute the real session key using $z_i = (s_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}})$ (line 69) or $z_i = (u_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, \hat{u}_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_i^{\mathsf{S}})$ (line 95). Note that the password is already defined for these instances.

GAME G_5

$(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
$(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(\mathcal{C}, T, T_s) \coloneqq (\emptyset, \emptyset, \emptyset)$
bad $\coloneqq$ **false**
$\beta \xleftarrow{\$} \{0,1\}$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
  $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
$\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
return $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$

if $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$ **return** $\bot$
$u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$s \coloneqq (s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{s} \coloneqq (\hat{s}_1, ..., \hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$
$\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star \tilde{x}, ..., \hat{u}_\ell \star \tilde{x})$
$x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
$\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star \tilde{x}, ..., \hat{s}_\ell \star \tilde{x})$
if $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
  return $\bot$
$K \xleftarrow{\$} \mathcal{K}$
$\pi_{\mathsf{U}}^{t_0} \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
$\pi_{\mathsf{S}}^{t_1} \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
$(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$
return $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$

if $\pi_{\mathsf{U}}^{t} \neq \bot$ **return** $\bot$
$(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
$u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
$\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star x_{b_1}, ..., \hat{u}_\ell \star x_{b_\ell})$
$\pi_{\mathsf{U}}^{t} \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
$\pi_{\mathsf{U}}^{t}.\mathsf{fr} \coloneqq$ **false**
return $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z)$

if $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$ **return** K
if $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}) \in T_s$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$
  if $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{U}, (u, \hat{u}), K)$
    for $i \in [\ell]$
      $z_i' \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$
    $z' \coloneqq (z_1', ..., z_\ell')$
  if $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{S}, (s, \hat{s}), K)$
    for $i \in [\ell]$
      $z_i' \coloneqq (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \star x_i^{\mathsf{U}})$
    $z' \coloneqq (z_1', ..., z_\ell')$
  if $z = z'$
    if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$: **return** K
    if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$: **bad** $\coloneqq$ **true**
$T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z] \xleftarrow{\$} \mathcal{K}$
return $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z]$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$

if $\pi_{\mathsf{S}}^{t} \neq \bot$ **return** $\bot$
$(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
$s \coloneqq (s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{s} \coloneqq (\hat{s}_1, ..., \hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$
$\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star x_{b_1}, ..., \hat{s}_\ell \star x_{b_\ell})$
if $\exists P \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
  return $\bot$
if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{S}}^{t}.\mathsf{fr} \coloneqq$ **true**
  if $\exists z$ s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$
  and $z_i \coloneqq (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \star x_i^{\mathsf{U}})\ \forall i \in [\ell]$
    bad $\coloneqq$ **true**
  $K \xleftarrow{\$} \mathcal{K}$
  $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] \coloneqq (\mathsf{S}, (s, \hat{s}), K)$
else
  $\pi_{\mathsf{S}}^{t}.\mathsf{fr} \coloneqq$ **false**
  for $i \in [\ell]$
    $z_i \coloneqq (s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \star x_i^{\mathsf{U}})$
  $z \coloneqq (z_1, ..., z_\ell)$
  $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_{\mathsf{S}}^{t} \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
return $(\mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

if $\pi_{\mathsf{U}}^{t} \neq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
  return $\bot$
if $\exists \mathsf{P} \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
  return $\bot$
if $\exists t'$ s.t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
and $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
  $\pi_{\mathsf{U}}^{t}.\mathsf{fr} \coloneqq$ **true**
  $(\mathsf{S}, (s, \hat{s}), K) \coloneqq T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}]$
else if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{U}}^{t}.\mathsf{fr} \coloneqq$ **true**
  if $\exists z$ s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$
  and $z_i \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})\ \forall i \in [\ell]$
    bad $\coloneqq$ **true**
  $K \xleftarrow{\$} \mathcal{K}$
  $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] \coloneqq (\mathsf{U}, (u, \hat{u}), K)$
else
  $\pi_{\mathsf{U}}^{t}.\mathsf{fr} \coloneqq$ **false**
  for $i \in [\ell]$
    $z_i \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$
  $z \coloneqq (z_1, ..., z_\ell)$
  $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_{\mathsf{U}}^{t} \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
return true

Figure 8: Game G_5 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{H}\}$. REVEAL, TEST and CORRUPT are defined as in Figure 5. Differences to G_4 are highlighted in blue.

GAME G_6

$(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
$(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(\mathcal{C}, T, T_\mathsf{s}, T_\mathrm{bad}) \coloneqq (\emptyset, \emptyset, \emptyset, \emptyset)$
$(\mathbf{bad}_\mathrm{guess}, \mathbf{bad}_\mathsf{pw}) \coloneqq (\mathbf{false}, \mathbf{false})$
$\beta \xleftarrow{\$} \{0,1\}$
$\beta' \leftarrow \mathcal{A}^\mathrm{O}(x_0, x_1)$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S} \setminus \mathcal{C}$
  $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
if $\exists \mathsf{pw}, \mathsf{pw}', (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, z, z')$ s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z) \in T_\mathrm{bad}$ **and** $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}', z') \in T_\mathrm{bad}$
  $\mathbf{bad}_\mathsf{pw} \coloneqq \mathbf{true}$
else
  if $\exists \mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, z$ s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}_{\mathsf{US}}, z) \in T_\mathrm{bad}$
    $\mathbf{bad}_\mathrm{guess} \coloneqq \mathbf{true}$
return $[\![\beta = \beta']\!]$

CORRUPT(U, S)

if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
for $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
  if $\exists t$ s.t. $\pi_\mathsf{P}^t.\mathsf{test} = \mathbf{true}$ **and** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_\mathsf{P}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
    return $\bot$
  $\forall \pi_\mathsf{P}^t$: **if** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_\mathsf{P}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
    $\pi_\mathsf{P}^t.\mathsf{fr} = \mathbf{false}$
$\mathcal{C} \coloneqq \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
$\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
return $\mathsf{pw}_{\mathsf{US}}$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$

if $\pi_\mathsf{U}^t \neq \bot$ **return** $\bot$
$u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^\mathsf{U} \coloneqq (x_1^\mathsf{U}, ..., x_\ell^\mathsf{U}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$
$\hat{x}^\mathsf{U} \coloneqq (\hat{x}_1^\mathsf{U}, ..., \hat{x}_\ell^\mathsf{U}) \coloneqq (\hat{u}_1 \star \tilde{x}, ..., \hat{u}_\ell \star \tilde{x})$
$\pi_\mathsf{U}^t \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, \bot, \bot), \bot, \bot)$
$\pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \bot$
return $(\mathsf{U}, x^\mathsf{U}, \hat{x}^\mathsf{U})$

H$(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z)$

if $T[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z] = K \neq \bot$ **return** K
if $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}) \in T_\mathsf{s}$
  $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
  if $T_\mathsf{s}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] = (\mathsf{U}, (u, \hat{u}), K)$
    for $i \in [\ell]$
      $z_i' \coloneqq (u_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{S}, \hat{u}_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{S}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_i^\mathsf{S})$
    $z' \coloneqq (z_1', ..., z_\ell')$
  if $T_\mathsf{s}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] = (\mathsf{S}, (s, \hat{s}), K)$
    for $i \in [\ell]$
      $z_i' \coloneqq (s_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{U}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_i^\mathsf{U}, \hat{s}_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{U})$
    $z' \coloneqq (z_1', ..., z_\ell')$
  if $z = z'$
    if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$: **return** K
    if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$: $T_\mathrm{bad} \coloneqq T_\mathrm{bad} \cup \{\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z\}$
$T[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z] \xleftarrow{\$} \mathcal{K}$
return $T[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z]$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^\mathsf{U}, \hat{x}^\mathsf{U})$

if $\pi_\mathsf{S}^t \neq \bot$ **return** $\bot$
$s \coloneqq (s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{s} \coloneqq (\hat{s}_1, ..., \hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^\mathsf{S} \coloneqq (x_1^\mathsf{S}, ..., x_\ell^\mathsf{S}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
$\hat{x}^\mathsf{S} \coloneqq (\hat{x}_1^\mathsf{S}, ..., \hat{x}_\ell^\mathsf{S}) \coloneqq (\hat{s}_1 \star \tilde{x}, ..., \hat{s}_\ell \star \tilde{x})$
if $\exists P \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S})$
  return $\bot$
if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_\mathsf{S}^t.\mathsf{fr} \coloneqq \mathbf{true}$
  $\forall \mathsf{pw}, z$ s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z) \in T$
    $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
    for $i \in [\ell]$
      $z_i' \coloneqq (s_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{U}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_i^\mathsf{U}, \hat{s}_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{U})$
    $z' \coloneqq (z_1', ..., z_\ell')$
    if $z = z'$
      $T_\mathrm{bad} \coloneqq T_\mathrm{bad} \cup \{(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z)\}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_\mathsf{s}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] \coloneqq (\mathsf{S}, (s, \hat{s}), K)$
else
  $\pi_\mathsf{S}^t.\mathsf{fr} \coloneqq \mathbf{false}$
  $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
  for $i \in [\ell]$
    $z_i \coloneqq (s_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{U}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_i^\mathsf{U}, \hat{s}_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{U})$
  $z \coloneqq (z_1, ..., z_\ell)$
  $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_\mathsf{S}^t \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}), K, \mathbf{true})$
return $(\mathsf{S}, x^\mathsf{S}, \hat{x}^\mathsf{S})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^\mathsf{S}, \hat{x}^\mathsf{S})$

if $\pi_\mathsf{U}^t \neq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, \bot, \bot), \bot, \bot)$ **return** $\bot$
if $\exists \mathsf{P} \in \mathcal{U}, t'$ s.t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S})$
  return $\bot$
if $\exists t'$ s.t. $\pi_\mathsf{S}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S})$ **and** $\pi_\mathsf{S}^{t'}.\mathsf{fr} = \mathbf{true}$
  $\pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{true}$
  $(\mathsf{S}, (s, \hat{s}), K) \coloneqq T_\mathsf{s}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}]$
else if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{true}$
  $\forall \mathsf{pw}, z$ s.t. $(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z) \in T$
    $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
    for $i \in [\ell]$
      $z_i' \coloneqq (u_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{S}, \hat{u}_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{S}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_i^\mathsf{S})$
    $z' \coloneqq (z_1', ..., z_\ell')$
    if $z = z'$
      $T_\mathrm{bad} \coloneqq T_\mathrm{bad} \cup \{(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}, z)\}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_\mathsf{s}[\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}] \coloneqq (\mathsf{U}, (u, \hat{u}), K)$
else
  $\pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{false}$
  $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
  for $i \in [\ell]$
    $z_i \coloneqq (u_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{S}, \hat{u}_i \cdot g_{b_i}^{-1} \star x_i^\mathsf{S}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_i^\mathsf{S})$
  $z \coloneqq (z_1, ..., z_\ell)$
  $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_\mathsf{U}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \hat{x}^\mathsf{U}, x^\mathsf{S}, \hat{x}^\mathsf{S}), K, \mathbf{true})$
return true

Figure 9: Game G_6 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\text{EXECUTE}, \text{SENDINIT}, \text{SENDRESP}, \text{SENDTERMINIT}, \text{REVEAL}, \text{CORRUPT}, \text{TEST}, \mathsf{H}\}$. Oracles REVEAL and TEST are defined as in game G_4 in Figure 5. Oracle EXECUTE is defined as in Figure 8. Differences to G_5 are highlighted in blue.

Recall that event **bad** in game G_5 is raised whenever there is an inconsistency in the random oracle queries and the keys of fresh instances. In this game, we split event **bad** into two different events:

- $\mathbf{bad}_{\mathsf{pw}}$ captures the event that there exists more than one valid entry in T for the same trace of a fresh instance, but different passwords.
- $\mathbf{bad}_{\text{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen and is raised if there exists a valid entry in T for the trace of a fresh instance and the correct password, where the password was not corrupted when the query to H was made.

To identify the different events, we introduce a new set T_{bad}. For all fresh instances in SendResp and SendTermInit, we now iterate over all entries in T that contain the corresponding trace. We check if the given password and z are valid for this trace by computing the real values z' in the same way as for non-fresh instances. If $z = z'$, we add this entry to the set T_{bad} (lines 56-62, 82-88). We essentially do the same when the random oracle H is queried on a trace that appears in T_{s}. Here, the adversary specifies the password and we check if z is valid for that password using the $u_i, \hat{u}_i$ stored in T_{s} for user instances and $s_i, \hat{s}_i$ for server instances. If z is valid and the instance is still fresh, we add the query to T_{bad} (lines 32-44). In case the password was corrupted in the meantime, we output the key stored in T_{s} as introduced in the previous game.

After the adversary terminates, we check T_{bad} whether event $\mathbf{bad}_{\mathsf{pw}}$ (line 09) or event $\mathbf{bad}_{\text{guess}}$ (line 12) occurred. We will bound these events below. First note that whenever **bad** is raised in G_5, then either flag $\mathbf{bad}_{\text{guess}}$ or $\mathbf{bad}_{\mathsf{pw}}$ is raised in G_6, thus

$$\Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] \ .$$

Finally, we bound the probabilities of the two events. We start with $\mathbf{bad}_{\mathsf{pw}}$. In Figure 10, we construct adversary $\mathcal{B}_2$ against DSim-GA-StCDH that simulates G_6.

We show that when $\mathbf{bad}_{\mathsf{pw}}$ occurs, then $\mathcal{B}_2$ can solve DSim-GA-StCDH. Hence,

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] \leq \mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_2) \ .$$

Adversary $\mathcal{B}_2$ inputs (x_0, x_1, w_0, w_1), where $x_0 = g_0 \star \tilde{x}$, $x_1 = g_1 \star \tilde{x}$, $w_0 = h_0 \star \tilde{x}$ and $w_1 = h_1 \star \tilde{x}$ for group elements $g_0, g_1, h_0, h_1 \in \mathcal{G}$ chosen uniformly at random. Adversary $\mathcal{B}_2$ also has access to decision oracles $\text{GA-DDH}_{x_j}(w_i, \cdot, \cdot)$ for $(i, j) \in \{0, 1\}^2$. It runs adversary $\mathcal{A}$ on (x_0, x_1). Queries to SendInit are simulated as follows: $\mathcal{B}_2$ chooses group elements u_i and $\hat{u}_i$ uniformly at random and sets

$$x_i^{\mathsf{U}} = u_i \star w_0 = (u_i \cdot h_0 \cdot g_0^{-1}) \star x_0 = (u_i \cdot h_0 \cdot g_1^{-1}) \star x_1 \ ,$$
$$\hat{x}_i^{\mathsf{U}} = \hat{u}_i \star w_1 = (\hat{u}_i \cdot h_1 \cdot g_0^{-1}) \star x_0 = (\hat{u}_i \cdot h_1 \cdot g_1^{-1}) \star x_1 \ .$$

The simulation of x_i^{S} and $\hat{x}_i^{\mathsf{S}}$ in SendResp is done in the same way, choosing random s_i and $\hat{s}_i$. In case the server instance is fresh, we must check if there already exists an entry in T that causes an inconsistency. As in G_6, we iterate over all pw, z, in T that contain the trace of this instance. In particular, we must check whether

$$z_{i,1} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, x_i^{\mathsf{U}}) = s_i^{-1} \star z_{i,1} \ ,$$
$$z_{i,2} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, \hat{x}_i^{\mathsf{U}}) = s_i^{-1} \star z_{i,2} \ ,$$
$$z_{i,3} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, \hat{x}_i^{\mathsf{S}}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_1, x_i^{\mathsf{U}}) = \hat{s}_i^{-1} \star z_{i,3} \ ,$$

$\underline{\mathcal{B}_2^{\{\text{GA-DDH}_{x_j}(w_i,\cdot,\cdot)\}_{i,j\in\{0,1\}}}(x_0,x_1,w_0,w_1)}$
00 $(\mathcal{C},T,T_\mathsf{s},T_\mathsf{bad}) \coloneqq (\emptyset,\emptyset,\emptyset,\emptyset)$
01 $\beta \xleftarrow{\$} \{0,1\}$
02 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0,x_1)$
03 **for** $(\mathsf{U},\mathsf{S}) \in \mathcal{U}\times\mathcal{S}\setminus\mathcal{C}$
04 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
05 **if** $\exists \mathsf{pw},\mathsf{pw}', (\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},z,z')$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\in T_\mathsf{bad}$ **and** $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}',z')\in T_\mathsf{bad}$
06 $\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}$
07 $\quad (b'_1,\ldots,b'_\ell) \coloneqq \mathsf{pw}'$
08 $\quad$ Find first index i such that $b_i \neq b'_i$
09 $\quad$ W.l.o.g. let $b_i = 0, b'_i = 1$
10 $\quad$ **if** $T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}] = (\mathsf{U},(u,\hat{u}),K)$
11 $\quad\quad$ Stop with $(x_i^{\mathsf{S}}, u_i^{-1}\star z_{i,1}, \hat{u}_i^{-1}\star z_{i,2}, u_i^{-1}\star z'_{i,1}, \hat{u}_i^{-1}\star z'_{i,2})$
12 $\quad$ **if** $T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}] = (\mathsf{S},(s,\hat{s}),K)$
13 $\quad\quad$ Stop with $(x_i^{\mathsf{U}}, s_i^{-1}\star z_{i,1}, \hat{s}_i^{-1}\star z_{i,3}, s_i^{-1}\star z'_{i,1}, \hat{s}_i^{-1}\star z'_{i,3})$

$\underline{\textsc{SendInit}(\mathsf{U},t,\mathsf{S})}$
14 **if** $\pi_\mathsf{U}^t \neq \bot$ **return** $\bot$
15 $u \coloneqq u_1,\ldots,u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
16 $\hat{u} \coloneqq (\hat{u}_1,\ldots,\hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
17 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}},\ldots,x_\ell^{\mathsf{U}}) \coloneqq (u_1\star w_0,\ldots,u_\ell\star w_0)$
18 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}},\ldots,\hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1\star w_1,\ldots,\hat{u}_\ell\star w_1)$
19 $\pi_\mathsf{U}^t \coloneqq ((u,\hat{u}),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},\bot,\bot),\bot,\bot)$
20 **return** $(\mathsf{U},x^{\mathsf{U}},\hat{x}^{\mathsf{U}})$

$\underline{\mathsf{H}(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)}$
21 **if** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z] = K \neq \bot$ **return** K
22 **if** $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}) \in T_\mathsf{s}$
23 $\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}$
24 $\quad$ **if** $T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}] = (\mathsf{U},(u,\hat{u}),K)$
25 $\quad\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{S}},u_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{S}},\hat{u}_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{S}},u_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
26 $\quad\quad\quad$ **if** $(\mathsf{U},\mathsf{S})\notin\mathcal{C}$
27 $\quad\quad\quad\quad T_\mathsf{bad} \coloneqq T_\mathsf{bad}\cup\{(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\}$
28 $\quad\quad\quad$ **if** $(\mathsf{U},\mathsf{S})\in\mathcal{C}$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$: **return** K
29 $\quad$ **if** $T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}] = (\mathsf{S},(s,\hat{s}),K)$
30 $\quad\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{U}},s_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{U}},s_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{U}},\hat{s}_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
31 $\quad\quad\quad$ **if** $(\mathsf{U},\mathsf{S})\notin\mathcal{C}$
32 $\quad\quad\quad\quad T_\mathsf{bad} \coloneqq T_\mathsf{bad}\cup\{(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\}$
33 $\quad\quad\quad$ **if** $(\mathsf{U},\mathsf{S})\in\mathcal{C}$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$: **return** K
34 **if** $\exists(u,\hat{u})$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},(u,\hat{u}))\in T$
35 $\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}$
36 $\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{S}},u_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{S}},\hat{u}_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{S}},u_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
37 $\quad\quad$ **return** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},(u,\hat{u})]$
38 **else if** $\exists(s,\hat{s})$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},(s,\hat{s}))\in T$
39 $\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}$
40 $\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{U}},s_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{U}},s_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{U}},\hat{s}_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
41 $\quad\quad$ **return** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},(s,\hat{s})]$
42 $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z] \xleftarrow{\$} \mathcal{K}$
43 **return** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z]$

$\underline{\textsc{SendResp}(\mathsf{S},t,\mathsf{U},x^{\mathsf{U}})}$
44 **if** $\pi_\mathsf{S}^t \neq \bot$ **return** $\bot$
45 $s \coloneqq (s_1,\ldots,s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
46 $\hat{s} \coloneqq (\hat{s}_1,\ldots,\hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
47 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}},\ldots,x_\ell^{\mathsf{S}}) \coloneqq (s_1\star w_0,\ldots,s_\ell\star w_0)$
48 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}},\ldots,\hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1\star w_1,\ldots,\hat{s}_\ell\star w_1)$
49 **if** $\exists P\in\mathcal{U}\cup\mathcal{S},t'$ s. t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})$
50 $\quad$ **return** $\bot$
51 **if** $(\mathsf{U},\mathsf{S})\notin\mathcal{C}$
52 $\quad \pi_\mathsf{S}^t.\mathsf{fr} \coloneqq$ **true**
53 $\quad \forall \mathsf{pw},z$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\in T$
54 $\quad\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}$
55 $\quad\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{U}},s_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{U}},s_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{U}},\hat{s}_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
56 $\quad\quad\quad T_\mathsf{bad} \coloneqq T_\mathsf{bad}\cup\{(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\}$
57 $\quad K \xleftarrow{\$} \mathcal{K}$
58 $\quad T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}] \coloneqq (S,(s,\hat{s}),K)$
59 **else**
60 $\quad \pi_\mathsf{S}^t.\mathsf{fr} \coloneqq$ **false**
61 $\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
62 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z)\in T$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{U}},s_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{U}},s_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{U}},\hat{s}_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
63 $\quad\quad K \coloneqq T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z]$
64 $\quad$ **else**
65 $\quad\quad K \xleftarrow{\$} \mathcal{K}$
66 $\quad\quad T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},(s,\hat{s})] \coloneqq K$
67 $\pi_\mathsf{S}^t \coloneqq ((s,\hat{s}),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}),K,\mathbf{true})$
68 **return** $(\mathsf{S},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})$

$\underline{\textsc{SendTermInit}(\mathsf{U},t,\mathsf{S},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})}$
69 **if** $\pi_\mathsf{U}^t \neq ((u,\hat{u}),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},\bot,\bot),\bot,\bot)$ **return** $\bot$
70 **if** $\exists\mathsf{P}\in\mathcal{U},t'$ s. t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})$ **return** $\bot$
71 **if** $\exists t'$ s. t. $\pi_\mathsf{S}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}})$ **and** $\pi_\mathsf{S}^{t'}.\mathsf{fr} =$ **true**
72 $\quad \pi_\mathsf{U}^t.\mathsf{fr} \coloneqq$ **true**
73 $\quad (\mathsf{S},(s,\hat{s}),K) \coloneqq T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}]$
74 **else if** $(\mathsf{U},\mathsf{S})\notin\mathcal{C}$
75 $\quad \pi_\mathsf{U}^t.\mathsf{fr} \coloneqq$ **true**
76 $\quad \forall \mathsf{pw},z$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\in T$
77 $\quad\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}$
78 $\quad\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{S}},u_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{S}},\hat{u}_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{S}},u_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
79 $\quad\quad\quad T_\mathsf{bad} \coloneqq T_\mathsf{bad}\cup\{(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw},z)\}$
80 $\quad K \xleftarrow{\$} \mathcal{K}$
81 $\quad T_\mathsf{s}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}] \coloneqq (\mathsf{U},(u,\hat{u}),K)$
82 **else**
83 $\quad \pi_\mathsf{S}^t.\mathsf{fr} \coloneqq$ **false**
84 $\quad (b_1,\ldots,b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
85 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z)\in T$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,x_i^{\mathsf{S}},u_i^{-1}\star z_{i,1}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_1,x_i^{\mathsf{S}},\hat{u}_i^{-1}\star z_{i,2}) = 1\ \forall i\in[\ell]$ **and** $\text{GA-DDH}_{x_{b_i}}(w_0,\hat{x}_i^{\mathsf{S}},u_i^{-1}\star z_{i,3}) = 1\ \forall i\in[\ell]$
86 $\quad\quad K \coloneqq T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z]$
87 $\quad$ **else**
88 $\quad\quad K \xleftarrow{\$} \mathcal{K}$
89 $\quad\quad T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},(u,\hat{u})] \coloneqq K$
90 $\pi_\mathsf{U}^t \coloneqq ((u,\hat{u}),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\hat{x}^{\mathsf{U}},x^{\mathsf{S}},\hat{x}^{\mathsf{S}}),K,\mathbf{true})$
91 **return true**

Figure 10: Adversary $\mathcal{B}_2$ against DSim-GA-StCDH for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTermInit}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{H}\}$. Oracles $\textsc{Execute}$, $\textsc{Reveal}$, $\textsc{Corrupt}$ and $\textsc{Test}$ are defined as in G_6. Lines written in blue show how $\mathcal{B}_2$ simulates the game.

which can be done with the decision oracles GA-DDH$_{x_{b_i}}(w_j, \cdot, \cdot)$. If all z_i are valid, then we add this entry to T_{bad} (lines 53-56).

If the instance is not fresh, then we have to compute the correct key. We check list T for a valid entry z as explained above and if it exists, we assign this value to the session key (line 63). Otherwise, we choose a random key and add a special entry to T, which instead of z contains the secret group elements s_i and $\hat{s}_i$ (line 66) so that we can patch the random oracle later. SendTermInit is simulated analogously, using the secret group elements u_i and $\hat{u}_i$.

Now we look at the random oracle queries. If the trace is contained in set T_{s} which means the corresponding instance was fresh when the send query was issued, we check if z is valid using the GA-DDH oracle. We do this as described above, depending on whether it is a user or a server instance (lines 24, 29). In case z is valid, we first check if the instance is still fresh (i.e., the password was not corrupted in the meantime) and if this is the case, we add the query to T_{bad} (lines 27, 32). Otherwise, if the password was corrupted and is specified in the query, we return the session key stored in T_{s} (lines 28, 33).

Next, we check if the query matches a special entry in T that was added in SendResp or SendTermInit for a non-fresh instance, which means we have to output the same key that was chosen before. Again, we can use the GA-DDH oracle and differentiate between user and server instances (lines 34-41).

After $\mathcal{A}$ terminates with output β', $\mathcal{B}_2$ chooses the passwords which have not been generated in a Corrupt query yet. If $\mathbf{bad}_{\mathsf{pw}}$ occurred (lines 05-13), then there must be two entries in T_{bad} for the same trace and different passwords $\mathsf{pw} \neq \mathsf{pw}'$ along with values z and z'. Let i be the first index where the two passwords differ, i.e., $b_i \neq b'_i$. Without loss of generality assume that $b_i = 0$ and $b'_i = 1$, otherwise swap pw, z and pw', z'. If the entries in T_{bad} are those of a user instance, we retrieve the secret group elements $u_{,}\hat{u}_i$ from T_{s}.

Recall that the $\mathsf{DSim\text{-}GA\text{-}StCDH}$ problem requires to compute the four values $y_0 = \mathsf{GA\text{-}CDH}_{x_0}(w_0, y)$, $y_1 = \mathsf{GA\text{-}CDH}_{x_0}(w_1, y)$, $y_2 = \mathsf{GA\text{-}CDH}_{x_1}(w_0, y)$ and $y_3 = \mathsf{GA\text{-}CDH}_{x_1}(w_1, y)$, where y can be chosen by the adversary. $\mathcal{B}_2$ sets $y = x_i^{\mathsf{S}}$, and outputs y and

$$
\begin{aligned}
y_0 &= u_i^{-1} \star z_{i,1} = \mathsf{GA\text{-}CDH}_{x_0}(u_i^{-1} \star x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_0, x_i^{\mathsf{S}}) \ , \\
y_1 &= \hat{u}_i^{-1} \star z_{i,2} = \mathsf{GA\text{-}CDH}_{x_0}(\hat{u}_i^{-1} \star \hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_1, x_i^{\mathsf{S}}) \ , \\
y_2 &= u_i^{-1} \star z'_{i,1} = \mathsf{GA\text{-}CDH}_{x_1}(u_i^{-1} \star x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_0, x_i^{\mathsf{S}}) \ , \\
y_3 &= \hat{u}_i^{-1} \star z'_{i,2} = \mathsf{GA\text{-}CDH}_{x_1}(\hat{u}_i^{-1} \star \hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_1, x_i^{\mathsf{S}}) \ .
\end{aligned}
$$

If the instance is a server instance, $\mathcal{B}_2$ outputs $(y, y_0, y_1, y_2, y_3) = (x_i^{\mathsf{U}}, s_i^{-1} \star z_{i,1}, \hat{s}_i^{-1} \star z_{i,3}, s_i^{-1} \star z'_{i,1}, \hat{s}_i^{-1} \star z'_{i,3})$. This concludes the analysis of $\mathbf{bad}_{\mathsf{pw}}$.

Next, we analyze event $\mathbf{bad}_{\text{guess}}$. Recall that $\mathbf{bad}_{\text{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen. Hence, for each instance there is at most one entry in T_{bad} and the size of T_{bad} is at most q_s. As all entries were added before the corresponding password was sampled,

the probability is bounded by

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] \leq \frac{q_s}{|\mathcal{PW}|} \ .$$

Finally, note that if none of the bad events happens in G_6, all session keys output by TEST are uniformly random and the adversary can only guess β. Hence, $\Pr[\mathsf{G}_6 \Rightarrow 1] = \frac{1}{2}$. Collecting the probabilities and using Equation Lemma 1 yields the bound in Theorem 1. □

7 Com-GA-PAKE$_\ell$: Three-Round PAKE from Group Actions

In this section we present a second modification of GA-PAKE$_\ell$, which can be securely instantiated with an EGAT. The protocol Com-GA-PAKE$_\ell$ extends GA-PAKE$_\ell$ by a commitment that has to be sent before sending the actual messages. In the first round, the server sends a commitment on those set elements it will send in the next round, thus ensuring that the server cannot choose the set elements depending on the message it receives from the user. This is the crucial step in the attack against GA-PAKE$_\ell$. In the second round, the user sends its message to the server and only after receiving that message, the servers sends its message to the user. While this protocol adds two rounds to the original protocol, the total computational cost is lower than for X-GA-PAKE$_\ell$.

7.1 Description of the Protocol

The setup for Com-GA-PAKE$_\ell$ is the same as for GA-PAKE$_\ell$, where the $\mathsf{crs} = (x_0, x_1)$ comprises two set elements, and the shared password is a bit string $(b_1, \ldots, b_\ell)$ of length ℓ.

The difference to GA-PAKE$_\ell$ is that before sending the set elements x^{U} and x^{S}, the server commits on x^{S}. More precisely, in the first round the server sends $\mathsf{com} = \mathsf{G}(x^{\mathsf{S}})$, where $\mathsf{G} : \{0,1\}^* \to \{0,1\}^\lambda$ is a hash function and λ is a parameter of the protocol. The user only accepts the session key after verifying that com corresponds to x^{S}. The session key K is derived as in GA-PAKE$_\ell$ but additionally takes the commitment com as input. The protocol is sketched in Figure 11. The security of Com-GA-PAKE$_\ell$ for EGATs is established in the following theorem.

Theorem 2 (Security of Com-GA-PAKE$_\ell$)**.** *For any adversary $\mathcal{A}$ against* Com-GA-PAKE$_\ell$ *that issues at most q_e execute queries, q_s send queries and at most q_{G} and q_{H} queries to random oracles* G *and* H*, there exist an adversary $\mathcal{B}_1$ against* GA-StCDH *and an adversary $\mathcal{B}_2$ against* GA-GapCDH *such that*

$$\mathsf{Adv}_{\mathsf{Com\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + q_s \ell \cdot \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^\ell} + \frac{q_{\mathsf{G}} q_s}{|\mathcal{G}|^\ell} + \frac{2 \cdot (q_{\mathsf{G}} + q_s + q_e)^2}{2^\lambda} + \frac{q_s}{|\mathcal{PW}|} \ ,$$

where λ is the output length of G *in bits.*

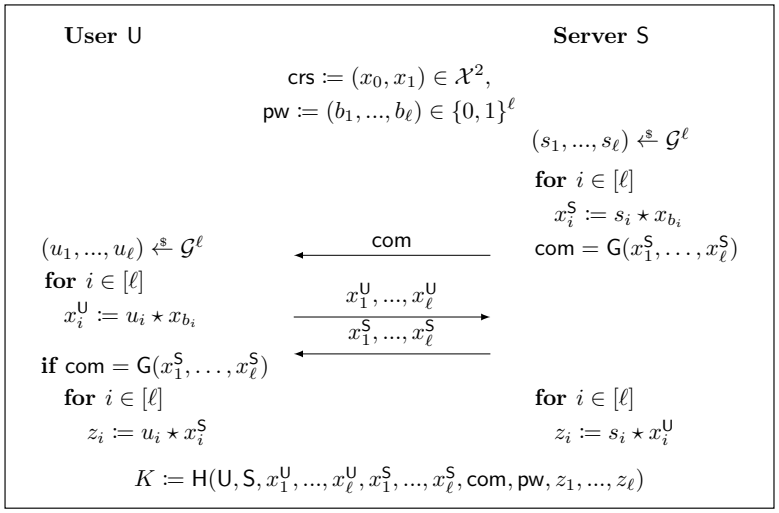

Figure 11: PAKE protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ from group actions.

The proof is similar to the one of Theorem 1 so we will only sketch it here. The full proof is given in Appendix E.

Sketch. After ensuring that all traces are unique, we need to deal with the commitment and in particular collisions. First, we require that there are never two inputs to the random oracle G that return the same commitment. This is to ensure that the adversary cannot open a commitment to a different value, which might depend on previous messages.

Second, we need to ensure that after the adversary has seen a commitment, it does not query G on the input, which is the hiding property of the commitment. What we actually do here is that we choose a random commitment in the first round. Only later we choose the input and patch the random oracle accordingly.

Now we can replace the session keys of instances which are used in execute queries. Here, the freshness condition allows the adversary to corrupt the password. However, as both x^{S} and x^{U} are generated by the experiment, the only chance to notice this change is to solve the $\mathsf{GA\text{-}StCDH}$ problem, where the decision oracle is required to simulate instances correctly.

In order to replace the session keys of fresh instances which are used in send queries, we make the key independent of the password. The session key of a fresh instance is now defined by the trace of that instance. The only issue that may arise here is an inconsistency between the session key that is derived using the trace and the session key that is derived using the random oracle H. Whenever such an inconsistency occurs, we differentiate between two cases:

- There exists more than one valid entry in T_{H} for the same trace of a fresh instance, but different passwords.
- There exists a valid entry in T_{H} for the trace of a fresh instance and the correct password, where the password was not corrupted when the query to H was made.

Finally, we bound the probabilities of the two cases. Similar to Theorem 1, we will define a new computational problem that reflects exactly the interaction in the protocol. We show that this problem is implied by GA-GapCDH using the reset lemma. The general idea is that the adversary can always compute the session key for one password guess, but not for a second one. After excluding this, we choose the actual password, which is possible because session keys are computed independently of the password. Thus, looking at one fixed instance, the probability that the adversary guessed the password correctly is $1/|\mathcal{PW}|$. □

8 Variants of the PAKE Protocols

Both protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ require that the user and the server generate multiple random group elements and evaluate their action on certain set elements. In this section we present two optimizations that allow us to reduce the number of random group elements and more importantly the number of necessary group action evaluations.

8.1 Increasing the Number of Public Parameters

In X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ the common reference string is set to $\mathsf{crs} \coloneqq (x_0, x_1) \in \mathcal{X}^2$. Increasing the number of public parameters allows to reduce the number of group action evaluations in the execution of the protocol. The idea is similar to the optimizations deployed to speed up the CSIDH-based signatures schemes SeaSign [DG19] and CSI-FiSh [BKV19]. We refer to Table 1 in the introduction for an overview and example of the parameter choice.

We explain the changes on the basis of protocol X-GA-PAKE$_\ell$. A security analysis for the variant is provided in Appendix D.1. The analysis for the variant of Com-GA-PAKE$_\ell$ is similar and is given in Appendix E.2. For some positive integer N dividing ℓ, we set

$$\mathsf{crs} \coloneqq (x_0, \ldots, x_{2^N-1}) \in \mathcal{X}^{2^N}.$$

As before, the password is a bitstring of length ℓ, but now we divide it into ℓ/N blocks of length N and write

$$\mathsf{pw} = (b_1, ..., b_{\ell/N}) \in \{0, ..., 2^N - 1\}^{\ell/N}.$$

In particular x_{b_i} refers to one of the 2^N different set elements in the crs. The general outline of the protocol does not change. The only difference is that in the first step both the server and the user only generate $2 \cdot \ell/N$ random group elements (instead of $2 \cdot \ell$). Hence they only need to perform $2 \cdot \ell/N$ group action evaluations in the first round and $3 \cdot \ell/N$ evaluations in the session key derivation. We write X-GA-PAKE$_{\ell,N}$ for this variant of the protocol.

8.2 Using Twists in the Setup

Both X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ require that some trusted party generates two random set elements $\mathsf{crs} = (x_0, x_1)$. Here, we shortly discuss the setup where x_1 is replaced by the twist of x_0, i.e. $\mathsf{crs} \coloneqq (x_0, x_0^t)$.

This simplification is particularly helpful when applied to one of the variants from the previous subsection. These modified versions require to generate 2^N random set elements for the crs. Using twists it suffices to generate 2^{N-1} random set elements. More precisely, a trusted party provides $(x_0, \ldots, x_{2^{N-1}-1}) \in \mathcal{X}^{2^{N-1}}$, then user and server set $x_{i+2^{N-1}} = x_i^t$ for each $i \in [0, 2^{N-1} - 1]$.

The security of X-GA-PAKE$_\ell^{\mathsf{t}}$ and Com-GA-PAKE$_\ell^{\mathsf{t}}$ (the twisted versions of X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$) are discussed in Appendices D.2 and E.2, respectively.

Acknowledgments

Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler and Doreen Riepel were supported by the DFG under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[AB19] Michel Abdalla and Manuel Barbosa. Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194, 2019. https://eprint.iacr.org/2019/1194. 246, 298

[ADMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 411–439. Springer, Heidelberg, December 2020. 236, 237, 240, 242, 244, 245, 268

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Heidelberg, January 2005. 246

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. Cryptology ePrint Archive, Report 2021/114, 2021. https://eprint.iacr.org/2021/114. 247, 270

[AJK+20] Reza Azarderakhsh, David Jao, Brian Koziel, Jason T. LeGrow, Vladimir Soukharev, and Oleg Taraskin. How not to create an isogeny-based PAKE. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20, Part I*, volume 12146 of *LNCS*, pages 169–186. Springer, Heidelberg, October 2020. 237, 239

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, February 2005. 235

[BBC+13] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, Heidelberg, August 2013. 240

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 227–247. Springer, Heidelberg, December 2019. 244, 263

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992. 235

[BP02] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 162–177. Springer, Heidelberg, August 2002. 269

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000. 245

[CDVW12] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Heidelberg, May 2012. 240, 241

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018. 236, 237, 243, 244

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `https://eprint.iacr.org/2006/291`. 236, 244

[DFMS21] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. Cryptology ePrint

Archive, Report 2021/280, 2021. https://eprint.iacr.org/2021/280. 242

[DG19] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 759–789. Springer, Heidelberg, May 2019. 263

[dKGV21] Bor de Kock, Kristian Gjøsteen, and Mattia Veroni. Practical isogeny-based key-exchange with optimal tightness. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography*, pages 451–479, Cham, 2021. Springer International Publishing. 241, 244

[FTY19] Atsushi Fujioka, Katsuyuki Takashima, and Kazuki Yoneyama. One-round authenticated group key exchange from isogenies. In Ron Steinfeld and Tsz Hon Yuen, editors, *ProvSec 2019*, volume 11821 of *LNCS*, pages 330–338. Springer, Heidelberg, October 2019. 241, 244

[GK10] Adam Groce and Jonathan Katz. A new framework for efficient password-based authenticated key exchange. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 516–525. ACM Press, October 2010. 240

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Heidelberg, May 2003. https://eprint.iacr.org/2003/032.ps.gz. 240

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384. 235

[HR10] Feng Hao and Peter Ryan. J-PAKE: Authenticated key exchange without PKI. Cryptology ePrint Archive, Report 2010/190, 2010. https://eprint.iacr.org/2010/190. 235

[Jab96] David P Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996. 235, 237

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011. 236

[KTAT20] Tomoki Kawashima, Katsuyuki Takashima, Yusuke Aikawa, and Tsuyoshi Takagi. An efficient authenticated key exchange from random self-reducibility on CSIDH. In Deukjo Hong, editor, *ICISC 20*, volume 12593 of *LNCS*, pages 58–84. Springer, Heidelberg, December 2020. 241, 244

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Heidelberg, March 2011. 240

[LGd21] Yi-Fu Lai, Steven D. Galbraith, and Cyprien de Saint Guilhem. Compact, efficient and UC-secure isogeny-based oblivious transfer. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 213–241. Springer, Heidelberg, October 2021. 240, 241, 242, 244, 297

[MOT20] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. SiGamal: A supersingular isogeny-based PKE and its application to a PRF. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 551–580. Springer, Heidelberg, December 2020. 240

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008. 240, 241

[PW17] David Pointcheval and Guilin Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17*, pages 301–312. ACM Press, April 2017. 247, 270

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145. 236

[SH19] Vladimir Soukharev and Basil Hess. PQDH: A quantum-safe replacement for Diffie-Hellman based on SIDH. Cryptology ePrint Archive, Report 2019/730, 2019. https://eprint.iacr.org/2019/730. 239

[TSJL18] Oleg Taraskin, Vladimir Soukharev, David Jao, and Jason LeGrow. An isogeny-based password-authenticated key establishment protocol. Cryptology ePrint Archive, Report 2018/886, 2018. https://eprint.iacr.org/2018/886. 239

[TY19] Shintaro Terada and Kazuki Yoneyama. Password-based authenticated key exchange from standard isogeny assumptions. In Ron Steinfeld and Tsz Hon Yuen, editors, *ProvSec 2019*, volume 11821 of *LNCS*, pages 41–56. Springer, Heidelberg, October 2019. 239

[Yon19] Kazuki Yoneyama. Post-quantum variants of iso/iec standards: Compact chosen ciphertext secure key encapsulation mechanism from isogeny. In *Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop*, SSR'19, page 13–21, New York, NY, USA, 2019. Association for Computing Machinery. 241, 244

Overview of Appendices

In Appendix A, we establish the relation between our assumptions and the computational assumptions in [ADMP20]. In Appendix B, we recall the Reset Lemma which is used in several security reductions. In Appendix C, we show that our first attempt $\mathsf{GA\text{-}PAKE}_\ell$ would be secure if an adversary was not able to compute twists efficiently. In Appendix D, we provide a security analysis for the two variants of $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$. Appendix E contains the security analysis of $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ as well as an analysis for different variants of the protocol. Finally, Appendix F deals with the perfect forward secrecy of $\mathsf{GA\text{-}PAKE}_\ell$, $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ and $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$.

A Relation to Assumptions from ADMP20

We first recall the definitions of a weak unpredictable permutation and weak unpredictable group action from [ADMP20] and then relate them to the group action computational Diffie-Hellman problem from Section 3.

Definition 10 (Weak Unpredictable Permutation [ADMP20])**.** *Let K, X and Y be sets indexed by λ, and let D_K and D_X be distributions on K and X respectively. Let $F_k^{\$}$ be a* randomized *oracle that when queried, samples $x \leftarrow D_X$ and outputs $(x, F(k,x))$. A (D_K, D_X)-weak UP (wUP) is a family of efficiently computable permutations $\{F(k,\cdot) : X \to X\}_{k\in K}$ such that for all PPT adversaries $\mathcal{A}$ we have*

$$\Pr[\mathcal{A}^{F_k^{\$}}(x^*) = F(k,x^*)] \leq negl(\lambda),$$

where $k \leftarrow D_K$, and $x^* \leftarrow D_X$. If D_K and D_X are uniform distributions, then we simply speak of a wUP family.

Definition 11 (Weak Unpredictable Group Action [ADMP20])**.** *A group action $(\mathcal{G}, \mathcal{X}, \star)$ is $(D_\mathcal{G}, D_\mathcal{X})$-weakly unpredictable if the family of efficiently computable permutations $\{\pi_g : X \to X\}_{g\in\mathcal{G}}$ is $(D_\mathcal{G}, D_\mathcal{X})$-weakly unpredictable, where π_g is defined as $\pi_g : x \mapsto g\star x$ and $D_\mathcal{G}$, $D_\mathcal{X}$ are distributions on $\mathcal{G}$, $\mathcal{X}$ respectively.*

Proposition 2. *If the group action computational Diffie-Hellman problem is hard for a group action, then the group action is weak unpredictable.*

Proof. Let $\mathcal{A}$ be an adversary against weak unpredictability, i.e., given access to an oracle π_g, where $g \leftarrow D_\mathcal{G}$ and $x^* \leftarrow D_\mathcal{X}$, $\mathcal{A}$ will compute $g \star x^*$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the group action computational Diffie-Hellman problem. $\mathcal{B}$ inputs $(x, y) = (g \star \tilde{x}, h \star \tilde{x})$ and has to compute $gh \star \tilde{x}$. Therefore, it runs $\mathcal{A}$ on $x^* := y$. On a query to π_g, $\mathcal{B}$ chooses $h' \leftarrow D_\mathcal{G}$ and computes $x' = h' \star \tilde{x}$ (instead of $x' \leftarrow D_\mathcal{X}$) and returns $(x', h' \star x)$ to $\mathcal{A}$. Note that $h' \star x = g \star x'$. Finally, $\mathcal{A}$ outputs $g \star x^*$, which $\mathcal{B}$ forwards as a solution to the group action computational Diffie-Hellman problem, since $g \star x^* = gh \star \tilde{x}$. □

The other direction is a bit more intricate. We can easily show that a more general definition of the group action computational Diffie-Hellman problem, namely where

the basis is not the origin element $\tilde{x}$, but a random set element, is tightly implied by the weak unpredictability property. However, we can also use the standard group action computational Diffie-Hellman problem, but with a non-tight reduction. Therefore, we use the fact that $\mathsf{GA\text{-}CDH}_x(y_0, y_1) = \mathsf{GA\text{-}CDH}(\mathsf{GA\text{-}CDH}(y_0, y_1), x^t)$.

Proposition 3. *If a group action is weak unpredictable, then the group action computational Diffie-Hellman problem is hard for the group action.*

Proof. Let $\mathcal{A}$ be an adversary against the group action computational Diffie-Hellman problem, i.e., on input $(x, y) = (g \star \tilde{x}, h \star \tilde{x})$ it computes $gh \star \tilde{x}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against weak unpredictability. $\mathcal{B}$ inputs $x^* \leftarrow D_{\mathcal{X}}$ and has access to an oracle π_g, where $g \leftarrow D_{\mathcal{G}}$. It queries π_g once to receive $(x, g \star x)$. Let $x = g' \star \tilde{x}$. $\mathcal{B}$ runs $\mathcal{A}$ on $(g \star x, x^*)$ and $\mathcal{A}$ outputs $gg' \star x^*$. Now $\mathcal{B}$ runs $\mathcal{A}$ a second time, this time on input $(gg' \star x^*, x^t)$. Note that $\mathsf{GA\text{-}CDH}(gg' \star x^*, x^t) = gg'(g')^{-1} \star x^*$, which is the solution for the weak unpredictability experiment. □

B Reset Lemma

We recall the reset lemma given by Bellare and Palacio [BP02, Lemma 3.1], which we will need to relate some of our new assumptions.

Lemma 2 (Reset Lemma [BP02])**.** *Fix a non-empty set H. Let $\mathcal{B}$ be an adversary that on input (I, h) returns a pair, where the first element is a bit b and the second element σ is some side output. Let IG be a randomized algorithm that we call instance generator. The accepting probability of $\mathcal{B}$ is defined as*

$$\mathrm{acc} := \Pr[b = 1 \mid I \xleftarrow{\$} \mathsf{IG}; h \xleftarrow{\$} H; (b, \sigma) \xleftarrow{\$} \mathcal{B}(I, h)] \ .$$

The reset algorithm $\mathcal{R}_{\mathcal{B}}$ associated to $\mathcal{B}$ is defined as in Figure 12. Let $\mathrm{res} = \Pr[b^* = 1 : I \xleftarrow{\$} \mathsf{IG}; (b^*, \sigma, \sigma') \xleftarrow{\$} \mathcal{R}_{\mathcal{B}}(I)]$*. Then*

$$\mathrm{acc} \leq \sqrt{\mathrm{res}} + \frac{1}{|H|} \ .$$

$\underline{\mathcal{R}_{\mathcal{B}}(I)}$
00 Pick random coins ρ for $\mathcal{B}$
01 $h \xleftarrow{\$} H$; $(b, \sigma) \leftarrow \mathcal{B}(I, h; \rho)$
02 $h' \xleftarrow{\$} H$; $(b', \sigma') \leftarrow \mathcal{B}(I, h'; \rho)$
03 **if** $b = b' = 1$ **and** $h \neq h'$
04 **return** $(1, \sigma, \sigma')$
05 **return** $(0, \epsilon, \epsilon)$

Figure 12: Reset algorithm $\mathcal{R}_{\mathcal{B}}$ associated to adversary $\mathcal{B}$.

C Security of GA-PAKE$_\ell$ in the EGA Setting

We introduce a new security assumption for EGA and REGA, namely the simultaneous GA-StCDH, which is used in the traditional Diffie-Hellman setting to prove security of several PAKE protocols [PW17, AHH21].

Definition 12 (Simultaneous GA-StCDH (Sim-GA-StCDH)). *On input* $(x, x_0, x_1) = (g \star \tilde{x}, g_0 \star \tilde{x}, g_1 \star \tilde{x})$*, the* Sim-GA-StCDH *problem requires to compute the set elements* $y_0 = ({g_0}^{-1} \cdot g) \star y$*,* $y_1 = ({g_1}^{-1} \cdot g) \star y$*, where* $y \in \mathcal{X}$ *can be chosen by the adversary* $\mathcal{A}$*. To an effective group action* XXX $\in \{\mathsf{EGA}, \mathsf{REGA}\}$*, we associate the advantage function of* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{Sim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{matrix} y_0 = \mathsf{GA\text{-}CDH}_{x_0}(x, y), \\ y_1 = \mathsf{GA\text{-}CDH}_{x_1}(x, y) \end{matrix} \;\middle|\; \begin{matrix} (g, g_0, g_1) \xleftarrow{\$} \mathcal{G}^3 \\ (x, x_0, x_1) \coloneqq (g \star \tilde{x}, g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ (y, y_0, y_1) \leftarrow \mathcal{A}^{\mathrm{O}}(x, x_0, x_1) \end{matrix}\right],$$

where $\mathrm{O} = \{\text{GA-DDH}_{x_0}(x, \cdot, \cdot), \text{GA-DDH}_{x_1}(x, \cdot, \cdot)\}$.

Note that the Sim-GA-StCDH problem is easy in the EGAT and REGAT setting, where the group action allows to twist elements efficiently (see Definition 4). The attack works exactly as the attack against GA-PAKE$_\ell$ given in Section 5. An adversary can solve the Sim-GA-StCDH problem by choosing $(y, y_0, y_1) = (x^t, x_0^t, x_1^t)$.

On the other hand, if a group action does not allow for efficient twisting, the Sim-GA-StCDH problem is believed to be hard. Pointcheval and Wand [PW17] analyzed the generic hardness of the assumption in the traditional Diffie-Hellman setting, where $\mathcal{G} = \mathcal{X} = \mathbb{F}_p^*$.

Theorem 3 (Security of GA-PAKE$_\ell$). *For any adversary* $\mathcal{A}$ *against* GA-PAKE$_\ell$ *that issues at most* q_e *execute queries and* q_s *send queries and where* H *is modeled as a random oracle, there exist adversary* $\mathcal{B}_1$ *against* GA-StCDH *and adversary* $\mathcal{B}_2$ *against* Sim-GA-StCDH *such that*

$$\mathsf{Adv}_{\mathsf{GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{Sim\text{-}GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_2) + \frac{q_s}{|\mathcal{PW}|} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^\ell} \quad .$$

Proof. The proof follows the one of Theorem 1 very closely, so we will give the full games and adversaries in pseudocode, but leave descriptions short.

Let $\mathcal{A}$ be an adversary against GA-PAKE$_\ell$. Consider the games in Figure 13.

Game G_0. This is the original game, hence

$$\mathsf{Adv}_{\mathsf{GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq |\Pr[\mathsf{G}_0 \Rightarrow 1] - 1/2| \quad .$$

Game G_1. In game G_1, we raise flag $\mathbf{bad}_{\text{coll}}$ and output $\perp$ whenever a server instance computes the same trace as any other accepted instance or a user instance computes the same trace as any other accepted user instance. As user and server messages consist of ℓ group elements each, we have

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{coll}}] \leq \frac{(q_e + q_s)^2}{|\mathcal{G}|^\ell} \quad .$$

GAMES G_0-G_4

$(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
$(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(\mathcal{C}, T) \coloneqq (\emptyset, \emptyset)$
$\mathbf{bad}_{\text{coll}} \coloneqq \mathbf{false}$
$\beta \xleftarrow{\$} \{0,1\}$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
  $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
$\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
return $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$

if $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
  return $\bot$
$(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$ //G_0-G_3
$(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$ //G_0-G_3
$x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$ //G_0-G_3
$z \coloneqq (z_1, ..., z_\ell) \coloneqq (u_1 \star x_1^{\mathsf{S}}, ..., u_\ell \star x_\ell^{\mathsf{S}})$ //G_0-G_3
$x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$ //G_4
$x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$ //G_4
if $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$ //G_1-G_4
  $\mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ //G_1-G_4
  return $\bot$ //G_1-G_4
$K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$ //G_0-G_2
$K \xleftarrow{\$} \mathcal{K}$ //G_3-G_4
$\pi_{\mathsf{U}}^{t_0} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
$\pi_{\mathsf{S}}^{t_1} \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
$(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$ //G_2-G_4
return $(\mathsf{U}, x^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}})$

REVEAL(P, t)

if $\pi_{\mathsf{P}}^t.\mathsf{acc} \neq \mathbf{true}$ **or** $\pi_{\mathsf{P}}^t.\mathsf{test} = \mathbf{true}$
  return $\bot$
if $\exists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
  and $\pi_{\mathsf{P}'}^{t'}.\mathsf{test} = \mathbf{true}$
  return $\bot$
$\forall (\mathsf{P}', t')$ s. t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = \pi_{\mathsf{P}}^t.\mathsf{tr}$ //G_2-G_4
  $\pi_{\mathsf{P}'}^{t'}.\mathsf{fr} \coloneqq \mathbf{false}$ //G_2-G_4
return $\pi_{\mathsf{P}}^t.K$

TEST$(\mathsf{P}, t))$

if $\mathsf{Fresh}(\pi_{\mathsf{P}}^t) = \mathbf{false}$ **return** $\bot$ //G_0-G_1
if $\pi_{\mathsf{P}}^t.\mathsf{fr} = \mathbf{false}$ **return** $\bot$ //G_2-G_4
$K_0^* \coloneqq$ REVEAL(P, t)
if $K_0^* = \bot$ **return** $\bot$
$K_1^* \xleftarrow{\$} \mathcal{K}$
$\pi_{\mathsf{P}}^t.\mathsf{test} \coloneqq \mathbf{true}$
return K_β^*

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)$

if $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$
  return K
$T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, Z] \xleftarrow{\$} \mathcal{K}$
return $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z]$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$

if $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
$(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
$(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
$\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot), \bot, \bot)$
$\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ //G_2-G_4
return $(\mathsf{U}, x^{\mathsf{U}})$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$

if $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
$(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
$(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$
if $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$ //G_1-G_4
  $\mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ //G_1-G_4
  return $\bot$ //G_1-G_4
if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ //G_2-G_4
  $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ //G_2-G_4
else //G_2-G_4
  $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ //G_2-G_4
$z \coloneqq (z_1, ..., z_\ell) \coloneqq (s_1 \star x_1^{\mathsf{U}}, ..., s_\ell \star x_\ell^{\mathsf{U}})$
$K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_{\mathsf{S}}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
return $(\mathsf{S}, x^{\mathsf{S}})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$

if $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot), \bot, \bot)$
  return $\bot$
if $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$ //G_1-G_4
  $\mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ //G_1-G_4
  return $\bot$ //G_1-G_4
if $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
  and $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$ //G_2-G_4
  $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ //G_2-G_4
else if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ //G_2-G_4
  $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ //G_2-G_4
else //G_2-G_4
  $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ //G_2-G_4
$z \coloneqq (z_1, ..., z_\ell) \coloneqq (u_1 \star x_1^{\mathsf{S}}, ..., u_\ell \star x_\ell^{\mathsf{S}})$
$K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
return true

CORRUPT(U, S)

if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
for $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
  if $\exists t$ s. t. $\pi_{\mathsf{P}}^t.\mathsf{test} = \mathbf{true}$
  and $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
    return $\bot$
  $\forall \pi_{\mathsf{P}}^t$: **if** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$ //G_2-G_4
    $\pi_{\mathsf{P}}^t.\mathsf{fr} = \mathbf{false}$ //G_2-G_4
$\mathcal{C} \coloneqq \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
return $\mathsf{pw}_{\mathsf{US}}$

Figure 13: Games G_0-G_4 for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{H}\}$.

GAME G_2. In game G_2, we make the freshness explicit. To each oracle π_{P}^t, we assign an additional variable $\pi_{\mathsf{P}}^t.\mathsf{fr}$ which is updated during the game. These are only a conceptual changes, hence

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1] \ .$$

GAME G_3. In game G_3, we choose random keys for all instances queried to EXECUTE. We construct adversary $\mathcal{B}_1$ against GA-StCDH in Figure 14 and show that

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}_{\mathsf{EGA}}^{\mathsf{GA\text{-}StCDH}}(\mathcal{B}_1) \ . \tag{1}$$

Adversary $\mathcal{B}_1$ inputs a GA-StCDH challenge $(x, y) = (g \star \tilde{x}, h \star \tilde{x})$ and has access to a decision oracle GA-DDH$(x, \cdot, \cdot)$. First, it generates the crs elements (x_0, x_1) as in game G_3 and then runs adversary $\mathcal{A}$. In EXECUTE queries, $\mathcal{B}_1$ chooses random group elements u_i and s_i and computes x^{U} using x and x^{S} using y independent of the password such that

$$x_i^{\mathsf{U}} = u_i \star x = (u_i \cdot g) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i} \cdot g_{b_i}^{-1}) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i}^{-1}) \star x_{b_i}$$

and analogously for server instances. Note that the value z_i is implicitly set to

$$z_i = u_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x}$$

Before choosing a random session key now, we check if there has been a query to the random oracle H with the correct z. This can be done using the decision oracle and the following equality:

$$\begin{aligned} \mathsf{GA\text{-}CDH}(x, x_i^{\mathsf{S}}) = (u_i^{-1} \cdot g_{b_i}) \star z_i \ &\Leftrightarrow \ \mathsf{GA\text{-}CDH}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = g_{b_i} \star z_i \\ &\Leftrightarrow \ \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = z_i \ . \end{aligned}$$

If one z_i is correct, $\mathcal{B}_1$ aborts and outputs the solution $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i = (g \cdot h) \star \tilde{x}$.

Otherwise, we store the values u_i and s_i in list T_{e} together with the trace and the password and choose a session key uniformly at random. List T_{e} is used to identify relevant queries to H. In particular, if the trace and password appear in a query, we retrieve the values u_i and s_i to check whether the provided z_i are correct as described above. If the oracle returns 1 for any i, $\mathcal{B}_1$ aborts and outputs $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i$.

GAME G_4. In game G_4, we remove the password from execute queries and use $\tilde{x}$ as the basis to compute x_i^{U} and x_i^{S}. This change is not observable by $\mathcal{A}$ and

$$\Pr[\mathsf{G}_4 \Rightarrow 1] = \Pr[\mathsf{G}_3 \Rightarrow 1] \ .$$

GAME G_5. G_5 is given in Figure 15. In this game we want to replace the session keys by random for all fresh instances in oracles SENDRESP and SENDTERMINIT. Therefore, we introduce an additional independent random oracle T_{s} which maps only the trace of an instance to a key. For all instances that are not fresh, we simply compute the correct key using random oracle H. If an instance is fresh and there is an inconsistency between T and T_{s}, we raise flag **bad**. This happens in the following cases:

$\underline{\mathcal{B}_1^{\text{GA-DDH}(x,\cdot,\cdot)}(x,y)}$
00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T, T_e) \coloneqq (\emptyset, \emptyset, \emptyset)$
03 $\beta \xleftarrow{\$} \{0,1\}$
04 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
05 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
06 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
07 Stop.

$\underline{\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)}$
08 **if** $\exists (u_1, ..., u_\ell, s_1, ..., s_\ell)$
$\quad$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, u_1, ..., u_\ell, s_1, ..., s_\ell) \in T_e$
09 $\quad (b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
10 $\quad$ **for** $i \in [\ell]$
11 $\quad\quad$ **if** GA-DDH$(x, x_i^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star z_i) = 1$
12 $\quad\quad\quad$ Stop with $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i$
13 **if** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$
14 $\quad$ **return** K
15 $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] \xleftarrow{\$} \mathcal{K}$
16 **return** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z]$

$\underline{\textsc{Execute}(\mathsf{U}, t_0, \mathsf{S}, t_1)}$
17 **if** $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
18 $\quad$ **return** $\bot$
19 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
20 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
21 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
22 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x, ..., u_\ell \star x)$
23 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star y, ..., s_\ell \star y)$
24 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
25 $\quad$ **return** $\bot$
26 $\forall z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$
27 $\quad$ **for** $i \in [\ell]$
28 $\quad\quad$ **if** GA-DDH$(x, x_i^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star z_i) = 1$
29 $\quad\quad\quad$ Stop with $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i$
30 $T_e \coloneqq T_e \cup \{\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, u_1, ..., u_\ell, s_1, ..., s_\ell\}$
31 $K \xleftarrow{\$} \mathcal{K}$
32 $\pi_{\mathsf{U}}^{t_0} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
33 $\pi_{\mathsf{S}}^{t_1} \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
34 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$
35 **return** $(\mathsf{U}, x^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}})$

Figure 14: Adversary $\mathcal{B}_1$ against GA-StCDH for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTermInit}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{H}\}$. Oracles SendInit, SendResp, SendTermInit, Reveal, Corrupt and Test are defined as in G_2. Lines written in blue show how $\mathcal{B}_1$ simulates the game.

- a fresh user or server instance is about to compute the session key, but there already exists a valid entry in T.
- the random oracle is queried on some trace of a fresh instance that appears in T_s together with the correct password and z.

Note that when **bad** is not raised, there is no difference between G_4 and G_5. Hence,

$$|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \ .$$

Game G_6. G_6 is given in Figure 16. In this game we remove the password from send queries and generate passwords as late as possible. In SendInit and SendResp we then compute x^{U} and x^{S} using $\tilde{x}$ such that

$$x_i^{\mathsf{U}} = u_i \cdot \tilde{x} = (u_i \cdot g_0^{-1}) \star x_0 = (u_i \cdot g_1^{-1}) \star x_1$$

and equivalently for server instances. For all instances that are not fresh, we have to compute the real session key using $z_i = (s_i \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{U}}$ or $z_i = (u_i \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{S}}$. Now we split event **bad** into two different events:

- $\mathbf{bad}_{\mathsf{pw}}$ captures the event that there exists more than one valid entry in T for the same trace of a fresh instance, but different passwords.
- $\mathbf{bad}_{\text{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen and if there exists a valid entry in T for the trace of a fresh instance and the correct password.

To identify the different events, we introduce a new set T_{bad}. For all fresh instances in SendResp and SendTermInit, we now iterate over all entries in T that contain the

GAME G_5

00 $(g_0, g_1) \xleftarrow{s} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T, T_s) \coloneqq (\emptyset, \emptyset, \emptyset)$
03 $\mathbf{bad} \coloneqq \mathbf{false}$
04 $\beta \xleftarrow{s} \{0,1\}$
05 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
06 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{s} \mathcal{PW}$
07 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
08 **return** $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$

09 **if** $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
10 $\quad$ **return** $\bot$
11 $(u_1, ..., u_\ell) \xleftarrow{s} \mathcal{G}^\ell$
12 $(s_1, ..., s_\ell) \xleftarrow{s} \mathcal{G}^\ell$
13 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$
14 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
15 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
16 $\quad$ **return** $\bot$
17 $K \xleftarrow{s} \mathcal{K}$
18 $\pi_{\mathsf{U}}^{t_0} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
19 $\pi_{\mathsf{S}}^{t_1} \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
20 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$
21 **return** $(\mathsf{U}, x^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}})$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$

22 **if** $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
23 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
24 $(u_1, ..., u_\ell) \xleftarrow{s} \mathcal{G}^\ell$
25 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
26 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot), \bot, \bot)$
27 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
28 **return** $(\mathsf{U}, x^{\mathsf{U}})$

H$(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)$

29 **if** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$
30 $\quad$ **return** K
31 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}) \in T_s$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$
32 $\quad$ **if** $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{U}, u_1, ..., u_\ell, K)$
33 $\quad\quad z' \coloneqq (z'_1, ..., z'_\ell) \coloneqq (u_1 \star x_1^{\mathsf{S}}, ..., u_\ell \star x_\ell^{\mathsf{S}})$
34 $\quad$ **if** $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{S}, s_1, ..., s_\ell, K)$
35 $\quad\quad z' \coloneqq (z'_1, ..., z'_\ell) \coloneqq (s_1 \star x_1^{\mathsf{U}}, ..., s_\ell \star x_\ell^{\mathsf{U}})$
36 $\quad$ **if** $z = z'$
37 $\quad\quad$ **if** $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$
38 $\quad\quad\quad$ **return** K
39 $\quad\quad$ **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
40 $\quad\quad\quad \mathbf{bad} \coloneqq \mathbf{true}$
41 $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] \xleftarrow{s} \mathcal{K}$
42 **return** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z]$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$

43 **if** $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
44 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
45 $(s_1, ..., s_\ell) \xleftarrow{s} \mathcal{G}^\ell$
46 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$
47 **if** $\exists P \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
48 $\quad$ **return** $\bot$
49 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
50 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
51 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$ **and** $z_i = s_i \star x_i^{\mathsf{U}}\ \forall i \in [\ell]$
52 $\quad\quad \mathbf{bad} \coloneqq \mathbf{true}$
53 $\quad K \xleftarrow{s} \mathcal{K}$
54 $\quad T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
55 **else**
56 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
57 $\quad z \coloneqq (z_1, ..., z_\ell) \coloneqq (s_1 \star x_1^{\mathsf{U}}, ..., s_\ell \star x_\ell^{\mathsf{U}})$
58 $\quad K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
59 $\pi_{\mathsf{S}}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
60 **return** $(\mathsf{S}, x^{\mathsf{S}})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$

61 **if** $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot), \bot, \bot)$
62 $\quad$ **return** $\bot$
63 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
64 $\quad$ **return** $\bot$
65 **if** $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
66 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
67 $\quad (\mathsf{S}, (s_1, ..., s_\ell), K) \coloneqq T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}]$
68 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
69 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
70 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$ **and** $z_i = u_i \star x_i^{\mathsf{S}}\ \forall i \in [\ell]$
71 $\quad\quad \mathbf{bad} \coloneqq \mathbf{true}$
72 $\quad K \xleftarrow{s} \mathcal{K}$
73 $\quad T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] \coloneqq (\mathsf{U}, (u_1, ..., u_\ell), K)$
74 **else**
75 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
76 $\quad z \coloneqq (z_1, ..., z_\ell) \coloneqq (u_1 \star x_1^{\mathsf{S}}, ..., u_\ell \star x_\ell^{\mathsf{S}})$
77 $\quad K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
78 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
79 **return true**

Figure 15: Game G_5 for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\text{EXECUTE}, \text{SENDINIT}, \text{SENDRESP}, \text{SENDTERMINIT}, \text{REVEAL}, \text{CORRUPT}, \text{TEST}, \mathsf{H}\}$. Oracles REVEAL, CORRUPT and TEST are defined as in Figure 13. Differences to game G_4 are highlighted in blue.

corresponding trace. We check if the given password and z are valid for this trace by computing the real values z' in the same way as for non-fresh instances. If $z = z'$, we

GAME G_6

$(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
$(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
$(\mathcal{C}, T, T_s, T_{\text{bad}}) \coloneqq (\emptyset, \emptyset, \emptyset, \emptyset)$
$(\mathbf{bad}_{\text{guess}}, \mathbf{bad}_{\mathsf{pw}}, \mathbf{bad}_{\text{pfs}}) = (\mathbf{false}, \mathbf{false}, \mathbf{false})$
$\beta \xleftarrow{\$} \{0,1\}$
$\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
for $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S} \setminus \mathcal{C}$
  $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
if $\exists \mathsf{pw}, \mathsf{pw}', (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, z, z')$
  s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z) \in T_{\text{bad}}$
  and $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}', z') \in T_{\text{bad}}$
  $\mathbf{bad}_{\mathsf{pw}} \coloneqq \mathbf{true}$
else
  if $\exists \mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, z$
  s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\text{bad}}$
    $\mathbf{bad}_{\text{guess}} \coloneqq \mathbf{true}$
return $[\![\beta = \beta']\!]$

CORRUPT(U, S)

if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
for $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
  if $\exists t$ s.t. $\pi_{\mathsf{P}}^t.\mathsf{test} = \mathbf{true}$
  and $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
    return $\bot$
  $\forall \pi_{\mathsf{P}}^t$: **if** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
    $\pi_{\mathsf{P}}^t.\mathsf{fr} = \mathbf{false}$
$\mathcal{C} \coloneqq \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
$\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
return $\mathsf{pw}_{\mathsf{US}}$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$

if $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
$(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$
$\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot), \bot, \bot)$
$\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \bot$
return $(\mathsf{U}, x^{\mathsf{U}})$

H$(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)$

if $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$ **return** K
if $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}) \in T_s$
  $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
  if $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{U}, u_1, ..., u_\ell, K)$
    $z' \coloneqq ((u_1 \cdot g_{b_1}^{-1}) \star x_1^{\mathsf{S}}, ..., (u_\ell \cdot g_{b_\ell}^{-1}) \star x_\ell^{\mathsf{S}})$
  if $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{S}, s_1, ..., s_\ell, K)$
    $z' \coloneqq ((s_1 \cdot g_{b_1}^{-1}) \star x_1^{\mathsf{U}}, ..., (s_\ell \cdot g_{b_\ell}^{-1}) \star x_\ell^{\mathsf{U}})$
  if $z = z'$
    if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$
      return K
    if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
      $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z\}$
$T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] \xleftarrow{\$} \mathcal{K}$
return $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z]$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$

if $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
$(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
if $\exists P \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
  return $\bot$
if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
  $\forall \mathsf{pw}, z$ s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z) \in T$
    $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
    $z' \coloneqq ((s_1 \cdot g_{b_1}^{-1}) \star x_1^{\mathsf{U}}, ..., (s_\ell \cdot g_{b_\ell}^{-1}) \star x_\ell^{\mathsf{U}})$
    if $z = z'$
      $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)\}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
else
  $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
  $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
  $z \coloneqq ((s_1 \cdot g_{b_1}^{-1}) \star x_1^{\mathsf{U}}, ..., (s_\ell \cdot g_{b_\ell}^{-1}) \star x_\ell^{\mathsf{U}})$
  $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_{\mathsf{S}}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
return $(\mathsf{S}, x^{\mathsf{S}})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$

if $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot), \bot, \bot)$
  return $\bot$
if $\exists \mathsf{P} \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
  return $\bot$
if $\exists t'$ s.t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
  and $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
  $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
  $(\mathsf{S}, (s_1, ..., s_\ell), K) \coloneqq T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}]$
else if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
  $\forall \mathsf{pw}, z$ s.t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z) \in T$
    $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
    $z' \coloneqq ((u_1 \cdot g_{b_1}^{-1}) \star x_1^{\mathsf{S}}, ..., (u_\ell \cdot g_{b_\ell}^{-1}) \star x_\ell^{\mathsf{S}})$
    if $z = z'$
      $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)\}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_s[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] \coloneqq (\mathsf{U}, (u_1, ..., u_\ell), K)$
else
  $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
  $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
  $z \coloneqq ((u_1 \cdot g_{b_1}^{-1}) \star x_1^{\mathsf{S}}, ..., (u_\ell \cdot g_{b_\ell}^{-1}) \star x_\ell^{\mathsf{S}})$
  $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
$\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
return true

Figure 16: Game G_6 for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\text{EXECUTE}, \text{SENDINIT}, \text{SENDRESP}, \text{SENDTERMINIT}, \text{REVEAL}, \text{CORRUPT}, \text{TEST}, \mathsf{H}\}$. Oracles REVEAL and TEST are defined as in game G_4 in Figure 13. Oracle EXECUTE is defined as in Figure 15. Differences to game G_5 are highlighted in blue.

add this entry to the set T_{bad}. We essentially do the same when the random oracle H is queried on a trace that appears in T_{s}. Here, the adversary specifies the password and we check if z is valid for that password using the u_i stored in T_{s} for user instances and s_i for server instances. If z is valid and the instance is still fresh, we add the query to T_{bad}. In case the password was corrupted in the meantime, we output the key stored in T_{s} as introduced in the previous game.

After the adversary terminates, we check T_{bad} whether event $\mathbf{bad}_{\mathsf{pw}}$ or event $\mathbf{bad}_{\text{guess}}$ occurred. We will bound these events below. First note that whenever $\mathbf{bad}$ is raised in G_5, then either flag $\mathbf{bad}_{\text{guess}}$ or $\mathbf{bad}_{\mathsf{pw}}$ is raised in G_6, thus

$$\Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] \ .$$

To bound $\mathbf{bad}_{\mathsf{pw}}$, we construct adversary $\mathcal{B}_2$ against $\mathsf{Sim\text{-}GA\text{-}StCDH}$ in Figure 17. When $\mathbf{bad}_{\mathsf{pw}}$ occurs, then $\mathcal{B}_2$ can solve $\mathsf{Sim\text{-}GA\text{-}StCDH}$. Hence,

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] \leq \mathsf{Adv}_{\mathsf{EGA}}^{\mathsf{Sim\text{-}GA\text{-}StCDH}}(\mathcal{B}_2) \ .$$

Adversary $\mathcal{B}_2$ inputs (x, x_0, x_1), where $x = g \star \tilde{x}$, $x_0 = g_0 \star \tilde{x}$ and $x_1 = g_1 \star \tilde{x}$ for uniformly random group elements $g, g_0, g_1 \in \mathcal{G}$. It also has access to decision oracles $\text{GA-DDH}_{x_0}(x, \cdot, \cdot)$ and $\text{GA-DDH}_{x_1}(x, \cdot, \cdot)$. It runs adversary $\mathcal{A}$ on (x_0, x_1). On a query to SendInit, $\mathcal{B}_2$ embeds x in x^{U} such that

$$x_i^{\mathsf{U}} = u_i \star x = (u_i \cdot g_0^{-1} \cdot g) \star x_0 = (u_i \cdot g_1^{-1} \cdot g) \star x_1 \ .$$

The simulation of x_i^{S} in SendResp is done in the same way. In case the server instance is fresh, we must check if there already exists an entry in T that causes an inconsistency, iterating over all pw, z, in T and using the decision oracles to check

$$z_i = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, s_i \star x) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(x, x_i^{\mathsf{U}}) = s_i^{-1} \star z_i \ ,$$

If all z_i are valid, then we add this entry to T_{bad}.

If the instance is not fresh, then we have to compute the correct key. We check list T for a valid entry z as explained above or choose a random key and add a special entry to T, which instead of z contains the secret group elements s_i so that we can patch the random oracle later. SendTermInit is simulated analogously.

Now we look at the random oracle queries. If the trace is contained in set T_{s}, we check if z is valid using the GA-DDH oracle. In case z is valid, we first check if the instance is still fresh and we add the query to T_{bad}. Otherwise, if the password was corrupted and is specified in the query, we return the session key stored in T_{s}. Next, we check if the query matches a special entry in T that was added in SendResp or SendTermInit for a non-fresh instance to keep the output consistent.

After $\mathcal{A}$ terminates with output β', $\mathcal{B}_2$ chooses the passwords which have not been generated yet. If $\mathbf{bad}_{\mathsf{pw}}$ occurred, then there must be two entries in T_{bad} for the same trace and different passwords pw and pw' along with values z and z'. As $\mathsf{pw} \neq \mathsf{pw}'$, we look for the first index i where the two passwords differ, i.e., $b_i \neq b_i'$. Recall that the $\mathsf{Sim\text{-}GA\text{-}StCDH}$ problem requires to compute $y_0 = \mathsf{GA\text{-}CDH}_{x_0}(x, y)$, $y_1 = \mathsf{GA\text{-}CDH}_{x_1}(x, y)$,

$\mathcal{B}_2^{\text{GA-DDH}_{x_0}(x,\cdot,\cdot),\text{GA-DDH}_{x_1}(x,\cdot,\cdot)}(x, x_0, x_1)$

$(\mathcal{C}, T, T_{\mathsf{s}}) := (\emptyset, \emptyset, \emptyset)$
$\beta \xleftarrow{\$} \{0,1\}$
$\beta' \leftarrow \mathcal{A}^{\mathsf{O}}(x_0, x_1)$
for $(\mathsf{U},\mathsf{S}) \in \mathcal{U} \times \mathcal{S} \setminus \mathcal{C}$
  $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
if $\exists \mathsf{pw}, \mathsf{pw}', (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, z, z')$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z) \in T_{\text{bad}}$ **and** $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}', z') \in T_{\text{bad}}$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  $(b'_1, ..., b'_\ell) := \mathsf{pw}'$
  Find first index i such that $b_i \neq b'_i$
  W.l.o.g. let $b_i = 0$, $b'_i = 1$
  if $T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{U}, (u_1, ..., u_\ell), K)$
    Stop with $(x_i^{\mathsf{S}}, u_i^{-1} \star z_i, u_i^{-1} \star z'_i)$
  if $T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{S}, (s_1, ..., s_\ell), K)$
    Stop with $(x_i^{\mathsf{U}}, s_i^{-1} \star z_i, s_i^{-1} \star z'_i)$

SendInit$(\mathsf{U}, t, \mathsf{S})$

if $\pi_{\mathsf{U}}^t \neq \perp$ **return** $\perp$
$(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} := (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) := (u_1 \star x, ..., u_\ell \star x)$
$\pi_{\mathsf{U}}^t := ((u_1, ..., u_\ell), (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, \perp), \perp, \perp)$
return $(\mathsf{U}, x^{\mathsf{U}})$

$\mathsf{H}(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)$

if $T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \perp$ **return** K
if $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}) \in T_{\mathsf{s}}$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  if $T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{U}, (u_1, ..., u_\ell), K)$
    if GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
      if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
        $T_{\text{bad}} := T_{\text{bad}} \cup \{(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)\}$
      if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$
        return K
  if $T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] = (\mathsf{S}, (s_1, ..., s_\ell), K)$
    if GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
      if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
        $T_{\text{bad}} := T_{\text{bad}} \cup \{(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)\}$
      if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} = \mathsf{pw}_{\mathsf{US}}$
        return K
if $\exists (u_1, ..., u_\ell)$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, (u_1, ..., u_\ell)) \in T$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  if GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
    return $T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, (u_1, ..., u_\ell)]$
else if $\exists (s_1, ..., s_\ell)$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, (s_1, ..., s_\ell)) \in T$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  if GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
    return $T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, (s_1, ..., s_\ell)]$
$T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z] \xleftarrow{\$} \mathcal{K}$
return $T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z]$

SendResp$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$

if $\pi_{\mathsf{S}}^t \neq \perp$ **return** $\perp$
$(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{S}} := (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) := (s_1 \star x, ..., s_\ell \star x)$
if $\exists P \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
  return $\perp$
if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{true}$
  $\forall \mathsf{pw}, z$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z) \in T$
    $(b_1, ..., b_\ell) := \mathsf{pw}$
    if GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
      $T_{\text{bad}} := T_{\text{bad}} \cup \{(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)\}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] := (S, (s_1, ..., s_\ell), K)$
else
  $\pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{false}$
  $(b_1, ..., b_\ell) := \mathsf{pw}_{\mathsf{US}}$
  if $\exists z$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$ **and** GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
    $K := T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z]$
  else
    $K \xleftarrow{\$} \mathcal{K}$
    $T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, (s_1, ..., s_\ell)] := K$
$\pi_{\mathsf{S}}^t := ((s_1, ..., _\ell), (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
return $(\mathsf{S}, x^{\mathsf{S}})$

SendTermInit$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$

if $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, \perp), \perp, \perp)$
  return $\perp$
if $\exists \mathsf{P} \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$
  return $\perp$
if $\exists t'$ s.t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}})$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
  $\pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{true}$
  $(\mathsf{S}, (s_1, ..., s_\ell), K) := T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}]$
else if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{true}$
  $\forall \mathsf{pw}, z$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z) \in T$
    $(b_1, ..., b_\ell) := \mathsf{pw}$
    if GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
      $T_{\text{bad}} := T_{\text{bad}} \cup \{(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}, z)\}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_{\mathsf{s}}[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}] := (\mathsf{U}, (u_1, ..., u_\ell), K)$
else
  $\pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{false}$
  $(b_1, ..., b_\ell) := \mathsf{pw}_{\mathsf{US}}$
  if $\exists z$ s.t. $(\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$ **and** GA-DDH$_{x_{b_i}}(x, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
    $K := T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z]$
  else
    $K \xleftarrow{\$} \mathcal{K}$
    $T[\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, (u_1, ..., u_\ell)] := K$
$\pi_{\mathsf{U}}^t := ((u_1, ..., u_\ell), (\mathsf{U},\mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}), K, \mathbf{true})$
return true

Figure 17: Adversary $\mathcal{B}_2$ against Sim-GA-StCDH for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathsf{O} := \{$Execute, SendInit, SendResp, SendTermInit, Reveal, Corrupt, Test, $\mathsf{H}\}$. Oracles Execute, Reveal, Corrupt and Test are defined as in G_6. Lines written in blue show how $\mathcal{B}_2$ simulates the game.

where y can be chosen by the adversary. If the entries in T_{bad} belong to a user instance, $\mathcal{B}_2$ sets $y = x_i^{\mathsf{S}}$ and outputs y together with

$$y_0 = u_i^{-1} \star z_i = \mathsf{GA\text{-}CDH}_{x_0}(u_i^{-1} \star x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(x, x_i^{\mathsf{S}}) \ ,$$
$$y_1 = u_i^{-1} \star z_i' = \mathsf{GA\text{-}CDH}_{x_1}(u_i^{-1} \star x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(x, x_i^{\mathsf{S}}) \ .$$

If the instance is a server instance, $\mathcal{B}_2$ outputs $(y, y_0, y_1) = (x_i^{\mathsf{U}}, s_i^{-1} \star z_i, s_i^{-1} \star z_i')$. This concludes the analysis of $\mathbf{bad}_{\mathsf{pw}}$.

Next, we analyze event $\mathbf{bad}_{\text{guess}}$. As $\mathbf{bad}_{\text{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen, there is at most one entry for each instance in T_{bad} and the size of T_{bad} is at most q_s. As all entries were added before the corresponding password was sampled, we can bound the probability by

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] \leq \frac{q_s}{|\mathcal{PW}|} \ .$$

Finally, if none of the bad events happens in G_6, all session keys output by Test are uniformly random and the adversary can only guess β. Hence, $\Pr[\mathsf{G}_6 \Rightarrow 1] = \frac{1}{2}$ and collecting the probabilities yields the bound in Theorem 3. □

D Security of X-GA-PAKE$_{\ell,N}$ and X-GA-PAKE$^{\mathsf{t}}_{\ell}$

Protocols X-GA-PAKE$_{\ell,N}$ and X-GA-PAKE$^{\mathsf{t}}_{\ell}$ are the two variants of X-GA-PAKE$_{\ell}$ as introduced in Section 8. In Appendices D.1 and D.2, we now provide a security analysis for the two protocols.

D.1 X-GA-PAKE$_{\ell,N}$

The main result of this part is the following theorem.

Theorem 4 (Security of X-GA-PAKE$_{\ell,N}$)**.** *For any adversary $\mathcal{A}$ against* X-GA-PAKE$_{\ell,N}$ *that issues at most q_e execute queries and q_s send queries and where* H *is modeled as a random oracle, there exist adversary $\mathcal{B}_1$ against* GA-StCDH *and $\mathcal{B}_2$ against* SqInv-GA-StCDH *such that*

$$\mathsf{Adv}_{\mathsf{X\text{-}GA\text{-}PAKE}_{\ell,N}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_1) + 2 \cdot \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_2) + \frac{q_s}{|\mathcal{PW}|} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^M} \ .$$

The proof of Theorem 4 is very similar to the proof of Theorem 1. Therefore we do not give a full proof for the security of X-GA-PAKE$_{\ell,N}$, but shortly explain the difference between the two proofs.

The security assumptions underlying the proof of Theorem 1 need to be slightly adapted. In particular, the problem DSim-GA-StCDH is replaced by its variant 2^{N}DSim-GA-StCDH. However the 2^{N}DSim-GA-StCDH problem can be reduced to DSim-GA-StCDH (Lemma 3).

Definition 13 (2^{N}DSim-GA-StCDH)**.** *On input* $(x_0 = g_0 \star \tilde{x}, ..., x_{2^N-1} = g_{2^N-1} \star \tilde{x}, w_0 = h_0 \star \tilde{x}, w_1 = h_1 \star \tilde{x}) \in \mathcal{X}^{2^N+2}$*, the* 2^{N}DSim-GA-StCDH *problem requires to find* $i \neq j \in [0, 2^N - 1]$ *and a tuple* $(y, y_0, y_1, y_2, y_3) \in \mathcal{X}^5$ *such that*

$$(y_0, y_1, y_2, y_3) = (g_i^{-1} \cdot h_0 \star y,\ g_i^{-1} \cdot h_1 \star y,\ g_j^{-1} \cdot h_0 \star y,\ g_j^{-1} \cdot h_1 \star y).$$

For a group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{2^{\mathsf{N}}\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{array}{l} i \neq j \in [0, 2^N - 1] \\ y_0 = \mathsf{GA\text{-}CDH}_{x_i}(w_0, y) \\ y_1 = \mathsf{GA\text{-}CDH}_{x_i}(w_1, y) \\ y_2 = \mathsf{GA\text{-}CDH}_{x_j}(w_0, y) \\ y_3 = \mathsf{GA\text{-}CDH}_{x_j}(w_1, y) \end{array} \middle| \begin{array}{c} (g_0, ..., g_{2^N-1}, h_0, h_1) \xleftarrow{\$} \mathcal{G}^{2^N+2} \\ (x_0, ..., x_{2^N-1}) = (g_0 \star \tilde{x}, ..., g_{2^N-1} \star \tilde{x}) \\ (w_0, w_1) = (h_0 \star \tilde{x}, h_1 \star \tilde{x}) \\ (i, j, y, y_0, y_1, y_2, y_3) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, ..., x_{2^N-1}, w_0, w_1) \end{array} \right],$$

where $\mathrm{O} = \{\mathrm{GA\text{-}DDH}_{x_i}(w_j, \cdot, \cdot)\}_{i \in [0, 2^N-1], j \in \{0,1\}}$.

Lemma 3. *For any adversary* $\mathcal{A}$ *against* 2^{N}DSim-GA-StCDH*, there exists adversary* $\mathcal{B}$ *against* DSim-GA-StCDH *such that*

$$\mathsf{Adv}^{2^{\mathsf{N}}\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}) \ .$$

Proof. We construct adversary $\mathcal{B}$ as follows. On input $(x_0, x_1, w_0, w_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x})$, $\mathcal{B}$ chooses a random bit $b_i \xleftarrow{\$} \{0, 1\}$, a random group element $\alpha_i \xleftarrow{\$} \mathcal{G}$ and computes $x'_i = \alpha_i \star x_{b_i}$ for each $i \in \{0, ..., 2^N - 1\}$. Then it runs $\mathcal{A}$ on input $(x'_0, ..., x'_{2^N-1}, w_0, w_1)$. Finally, $\mathcal{A}$ outputs two indices (i, j) with $i \neq j$ and (y, y_0, y_1, y_2, y_3). If $b_i = b_j$ which happens with probability $1/2$, then $\mathcal{B}$ aborts. Otherwise, assume that $b_i = 0$ and $b_j = 1$. Then $x'_i = (\alpha_i g_0) \star \tilde{x}$ and $x'_j = (\alpha_j g_1) \star \tilde{x}$, hence

$$\begin{aligned} y_0 &= (\alpha_i^{-1} g_0^{-1} h_0) \star y, & y_2 &= (\alpha_j^{-1} g_1^{-1} h_0) \star y, \\ y_1 &= (\alpha_i^{-1} g_0^{-1} h_1) \star y, & y_3 &= (\alpha_j^{-1} g_1^{-1} h_1) \star y. \end{aligned}$$

It follows that $\mathcal{B}$ can compute the solution by simply multiplying by α_i and α_j respectively. If $b_j = 0$ and $b_i = 1$, the output of $\mathcal{B}$ must be swapped.

During the experiment, $\mathcal{A}$ also has access to decision oracles $\mathrm{GA\text{-}DDH}_{x'_i}(w_j, \cdot, \cdot)$ for $i \in \{0, ..., 2^N - 1\}$ and $j \in \{0, 1\}$. These can be easily simulated using $\mathcal{B}$'s decision oracles. On a query $\mathrm{GA\text{-}DDH}_{x'_i}(w_j, z_1, z_2)$, $\mathcal{B}$ queries its own oracle on $\mathrm{GA\text{-}DDH}_{x_{b_i}}(w_j, z_1, \alpha_i \star z_2)$ and forwards the output to $\mathcal{A}$. □

D.2 X-GA-PAKE$^{\mathsf{t}}_{\ell}$

Here we discuss the security of X-GA-PAKE$^{\mathsf{t}}_{\ell}$, the twisted version of X-GA-PAKE$_{\ell}$, in more detail. For the most part, one can just replace x_1 by x_0^t everywhere in the proof of Theorem 1. The only significant difference occurs in the analysis of the event $\mathbf{bad}_{\mathsf{pw}}$. In particular this event does not allow to construct an adversary against DSim-GA-StCDH. Instead, we need to consider the following alteration of DSim-GA-StCDH for the security analysis.

Definition 14 (Twisted Double Simultaneous GA-StCDH (TDSim-GA-StCDH))**.** *On input* $(x_0 = g_0 \star \tilde{x}, w_0 = h_0 \star \tilde{x}, w_1 = h_1 \star \tilde{x}) \in \mathcal{X}^3$*, the* TDSim-GA-StCDH *requires to find a tuple* $(y, y_0, y_1, y_2, y_3) \in \mathcal{X}^5$ *such that*

$$(y_0, y_1, y_2, y_3) = (g_0^{-1} \cdot h_0 \star y,\ g_0^{-1} \cdot h_1 \star y,\ g_0 \cdot h_0 \star y,\ g_0 \cdot h_1 \star y).$$

For a group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{TDSim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{array}{l|c} y_0 = \mathsf{GA\text{-}CDH}_{x_0}(w_0, y) & (g_0, h_0, h_1) \xleftarrow{\$} \mathcal{G}^4 \\ y_1 = \mathsf{GA\text{-}CDH}_{x_0}(w_1, y) & x_0 = g_0 \star \tilde{x} \\ y_2 = \mathsf{GA\text{-}CDH}_{x_0^t}(w_0, y) & (w_0, w_1) = (h_0 \star \tilde{x}, h_1 \star \tilde{x}) \\ y_3 = \mathsf{GA\text{-}CDH}_{x_0^t}(w_1, y) & (y, y_0, y_1, y_2, y_3) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, w_0, w_1) \end{array}\right],$$

where $\mathrm{O} = \{\mathrm{GA\text{-}DDH}_{x_0}(w_j, \cdot, \cdot)\}_{j \in \{0,1\}}$.

The proof of Lemma 1 can be adapted to the TDSim-GA-StCDH problem in a straightforward way, showing that

$$\mathsf{Adv}^{\mathsf{TDSim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}) \ .$$

Consequently the statement of Theorem 1 continues to hold true for $\mathsf{X\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell}$.

E Security of Com-GA-PAKE$_\ell$ and its Variants

The first part of this section is dedicated to the proof of Theorem 2. The second part discusses the different variants of $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ and analyzes their security.

E.1 Proof of Theorem 2

For the convenience of the reader, we repeat the statement of the theorem.

Theorem 2 (Security of $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$)**.** *For any adversary* $\mathcal{A}$ *against* $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ *that issues at most* q_e *execute queries,* q_s *send queries and at most* q_{G} *and* q_{H} *queries to random oracles* G *and* H*, there exist an adversary* $\mathcal{B}_1$ *against* GA-StCDH *and an adversary* $\mathcal{B}_2$ *against* GA-GapCDH *such that*

$$\mathsf{Adv}_{\mathsf{Com\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + q_s \ell \cdot \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^\ell} + \frac{q_{\mathsf{G}} q_s}{|\mathcal{G}|^\ell} + \frac{2 \cdot (q_{\mathsf{G}} + q_s + q_e)^2}{2^\lambda} + \frac{q_s}{|\mathcal{PW}|} \ ,$$

where λ *is the output length of* G *in bits.*

Before proving this theorem, we will introduce a new (interactive) computational assumption which is tailored to the protocol where the interactive part of the assumption reflects the commitment in that protocol. We will show that this assumption is implied by GA-GapCDH.

Definition 15 (Interactive Simultaneous GA-StCDH (ISim-GA-StCDH))**.** *On input* $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \in \mathcal{X}^2$*, the adversary first chooses and commits to some* $y \in \mathcal{X}$*. After receiving the challenge* $x = g \star \tilde{x} \in \mathcal{X}$*, the* ISim-GA-StCDH *problem requires to compute* $y_0 = g g_0^{-1} \star y, y_1 = g g_1^{-1} \star y$*. For a group action* $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{array}{l|c} & (g_0, g_1) \xleftarrow{\$} \mathcal{G}^2 \\ & (x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ y_0 = \mathsf{GA\text{-}CDH}_{x_0}(x, y) & y \leftarrow \mathcal{A}^{\mathrm{O}_1}(x_0, x_1) \\ y_1 = \mathsf{GA\text{-}CDH}_{x_1}(x, y) & g \xleftarrow{\$} \mathcal{G} \\ & x = g \star \tilde{x} \\ & (y_0, y_1) \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(x) \end{array}\right] ,$$

where $\mathrm{O}_1 = \{\text{GA-DDH}_{x_j}(\tilde{x}, \cdot, \cdot)\}_{j \in \{0,1\}}$ *and* $\mathrm{O}_2 = \{\text{GA-DDH}_{x_j}(x, \cdot, \cdot)\}_{j \in \{0,1\}}$.

Note that ISim-GA-StCDH would be easy without the commitment if a group action allows to compute twists efficiently. In this case an adversary could simply choose $(y, y_0, y_1) = (x^t, x_0^t, x_1^t)$. The commitment prevents this trivial solution and intuitively, it thwarts the offline dictionary attack that was possible on $\mathsf{GA\text{-}PAKE}_\ell$ (Proposition 1).

Lemma 4. *The Group Action Gap Computational Diffie-Hellman Problem* (GA-GapCDH) *problem implies the Interactive Simultaneous* GA-StCDH (ISim-GA-StCDH) *for* EGAT*s, in particular*

$$\mathsf{Adv}^{\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B})} \ .$$

Proof. For the proof we use the reset lemma (see Lemma 2) with $H = \mathcal{X}$. Let $\mathcal{A}$ be an adversary against ISim-GA-StCDH. Consider adversary $\mathcal{B}$ against GA-GapCDH in Figure 18 that takes input (x_0, x_1) as well as x. It also has access to a gap oracle GA-DDH_*. First, $\mathcal{B}$ runs $\mathcal{A}$ on (x_0, x_1) to receive a commitment y. Now $\mathcal{B}$ sends x to $\mathcal{A}$ and $\mathcal{A}$ will finally output (y_0, y_1). $\mathcal{B}$ checks if the solution is correct using the decision oracle and if this is the case, it outputs $b = 1$ and $\sigma = (y, y_0, y_1)$. Otherwise it outputs $(0, \epsilon)$. As $\mathcal{B}$ has access to a full gap oracle, it can forward all queries of $\mathcal{A}$.

Let IG be the algorithm that chooses $g_0, g_1 \xleftarrow{\$} \mathcal{G}$ and outputs $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$. Let acc be defined as in Lemma 2, thus

$$\text{acc} \geq \mathsf{Adv}^{\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \ .$$

Let $\mathcal{R}_\mathcal{B}$ be the reset algorithm associated to $\mathcal{B}$ as in Lemma 2 with access to the same decision oracles as $\mathcal{B}$.

We construct an adversary $\mathcal{C}$ against GA-GapCDH (Figure 18), but instead of running the reset algorithm, $\mathcal{C}$ will simulate $\mathcal{R}_\mathcal{B}$ running $\mathcal{B}$ directly.

$\mathcal{C}$ inputs (x_0, x_1) and has access to a gap oracle. First, it chooses random coins ρ for $\mathcal{B}$. It also samples a random element from H by first picking $a \xleftarrow{\$} \mathcal{G}$ and then computing $x = a \star \tilde{x}$. Then it runs $\mathcal{B}$ on $(x_0, x_1^t, x; \rho)$. Note that we use the twist of x_1. $\mathcal{B}$ outputs a bit b and side output σ. If $\mathcal{B}$ was successful, i.e., $b = 1$, then $\mathcal{C}$ parses σ as (y, y_1, y_2). Otherwise it aborts. Now it runs $\mathcal{B}$ a second time, this time on input

$\underline{\mathcal{B}^{\text{GA-DDH}_*}(x_0, x_1, x)}$
00 $y \leftarrow \mathcal{A}^{\text{O}_1}(x_0, x_1)$
01 $(y_0, y_1) \leftarrow \mathcal{A}^{\text{O}_1, \text{O}_2}(x)$
02 **if** $\text{GA-DDH}_{x_0}(x, y, y_0) = 1$
and $\text{GA-DDH}_{x_1}(x, y, y_1) = 1$
03 **return** $(1, (y, y_0, y_1))$
04 **return** $(0, \epsilon)$

$\underline{\mathcal{C}^{\text{GA-DDH}_*}(x_0, x_1)}$
05 Pick random coins ρ for $\mathcal{B}$
06 $a \xleftarrow{\$} \mathcal{G}$; $x \coloneqq a \star \tilde{x}$
07 $(b, \sigma) \leftarrow \mathcal{B}^{\text{GA-DDH}_*}(x_0, x_1^t, x; \rho)$
08 **if** $b = 0$ **return** $\perp$
09 $(y, y_0, y_1) \coloneqq \sigma$
10 $\alpha \xleftarrow{\$} \mathcal{G}$; $x' \coloneqq \alpha \star y_0^t$
11 $(b', \sigma') \leftarrow \mathcal{B}^{\text{GA-DDH}_*}(x_0, x_1^t, x'; \rho)$
12 **if** $b = 0$ **return** $\perp$
13 $(y, y_0', y_1') \coloneqq \sigma$
14 **return** $\alpha^{-1} \cdot a \star y_1'$

Figure 18: Adversaries $\mathcal{B}$ and $\mathcal{C}$ against GA-GapCDH for the proof of Lemma 4. Adversary $\mathcal{A}$ has access to decision oracles $\text{O}_1 = \{\text{GA-DDH}_{x_j}(\tilde{x}, \cdot, \cdot)\}_{j \in \{0,1\}}$ and $\text{O}_2 = \{\text{GA-DDH}_{x_j}(x, \cdot, \cdot)\}_{j \in \{0,1\}}$, which $\mathcal{B}$ simulates using the gap oracle GA-DDH_*.

(x_0, x_1^t, x'), where $x' = \alpha \star y_0^t$ for some $\alpha \xleftarrow{\$} \mathcal{G}$, and the same random coins ρ. Note that x' is also uniformly distributed over $\mathcal{X}$. If $\mathcal{B}$ is successful again, it outputs $(1, (y, y_0', y_1'))$, where y is the same as before since we run $\mathcal{B}$ on the same random coins. Now $\mathcal{C}$ can solve GA-GapCDH as follows: Let $y = h \star \tilde{x}$ for some $h \in \mathcal{G}$. Then we have

$$\alpha \star y_0^t = (\alpha \cdot a^{-1} g_0 h^{-1}) \star \tilde{x},$$

hence $y_0' = (\alpha \cdot a^{-1}) \star \tilde{x}$, $y_1' = (\alpha \cdot a^{-1} \cdot g_0 \cdot g_1 \star \tilde{x})$. Note that h cancels out. Using the knowledge of a and α, $\mathcal{C}$ can compute $\alpha^{-1} \cdot a \star y_1' = \text{GA-CDH}(x_0, x_1)$.

Note that even if $x = x'$, we can solve GA-GapCDH, so there is no additional term in the bound. □

Note that for the above proof it is indeed necessary that $\mathcal{B}$ (and $\mathcal{C}$) has access to a full gap oracle. If $\mathcal{B}$ only had access to a restricted oracle, it would not be able to simulate the oracles $\text{GA-DDH}_{x_0}(\alpha \star y_0^t, \cdot, \cdot)$ and $\text{GA-DDH}_{x_1^t}(\alpha \star y_0^t, \cdot, \cdot)$ in the second part of the proof.

Now we give the full proof of Theorem 2.

Proof (of Theorem 2). Let $\mathcal{A}$ be an adversary against Com-GA-PAKE_ℓ. Consider the games in Figures 19 and 20.

GAME G_0. This is the original game, hence

$$\text{Adv}_{\text{Com-GA-PAKE}_\ell}(\mathcal{A}) \leq |\Pr[\mathsf{G}_0 \Rightarrow 1] - 1/2| \quad .$$

GAME G_1. In game G_1, we raise flag $\mathbf{bad}_{\text{coll}}$ whenever a server instance computes the same trace as any other accepted instance (line 77) or a user instance computes the same trace as any other accepted user instance (line 33). In this case, SENDTERMINIT or SENDTERMRESP return $\perp$. We do the same if a trace that is computed in an EXECUTE query collides with one of a previously accepted instance (line 20). Due to the difference lemma,

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{coll}}] \quad .$$

GAMES G_0-G_7

00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T_\mathsf{H}, T_\mathsf{G}) \coloneqq (\emptyset, \emptyset)$
03 $(\mathbf{bad}_{\text{coll}}, \mathbf{bad}_{\text{bind}}, \mathbf{bad}_{\text{hide}}) \coloneqq (\mathbf{false}, \mathbf{false}, \mathbf{false})$
04 $\beta \xleftarrow{\$} \{0,1\}$
05 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
06 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
07 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
08 **return** $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$

09 **if** $\pi_\mathsf{U}^{t_0} \neq \bot$ **or** $\pi_\mathsf{S}^{t_1} \neq \bot$ **return** $\bot$
10 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$ // G_0-G_6
11 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
12 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
13 $x^\mathsf{U} \coloneqq (x_1^\mathsf{U}, ..., x_\ell^\mathsf{U}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$ // G_0-G_6
14 $x^\mathsf{S} \coloneqq (x_1^\mathsf{S}, ..., x_\ell^\mathsf{S}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$ // G_0-G_6
15 $z \coloneqq (z_1, ..., z_\ell) \coloneqq (u_1 \star x_1^\mathsf{S}, ..., u_\ell \star x_\ell^\mathsf{S})$ // G_0-G_6
16 $x^\mathsf{U} \coloneqq (x_1^\mathsf{U}, ..., x_\ell^\mathsf{U}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$ // G_7
17 $x^\mathsf{S} \coloneqq (x_1^\mathsf{S}, ..., x_\ell^\mathsf{S}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$ // G_7
18 $\mathsf{com} \coloneqq \mathsf{G}(x^\mathsf{S})$
19 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com})$ // G_1-G_7
20 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$; **return** $\bot$ // G_1-G_7
21 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$ // G_0-G_5
22 $K \xleftarrow{\$} \mathcal{K}$ // G_6-G_7
23 $\pi_\mathsf{U}^{t_0} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}), K, \mathbf{true})$
24 $\pi_\mathsf{S}^{t_1} \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}), K, \mathbf{true})$
25 $(\pi_\mathsf{U}^{t_0}.\mathsf{fr}, \pi_\mathsf{S}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$ // G_5-G_7
26 **return** $(\mathsf{U}, x^\mathsf{U}, \mathsf{S}, (\mathsf{com}, x^\mathsf{S}))$

SENDTERMRESP$(\mathsf{U}, t, \mathsf{S}, x^\mathsf{S})$

27 **if** $\pi_\mathsf{U}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \bot, \mathsf{com}), \bot, \bot)$
28 $\quad$ **return** $\bot$
29 **if** $\mathsf{G}(x^\mathsf{S}) \neq \mathsf{com}$
30 $\quad \pi_\mathsf{U}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}), \bot, \mathbf{false})$
31 $\quad$ **return** $\bot$
32 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com})$ // G_1-G_7
33 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$; **return** $\bot$ // G_1-G_7
34 **if** $\exists t'$ s. t. $\pi_\mathsf{S}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com})$
$\quad$ **and** $\pi_\mathsf{S}^{t'}.\mathsf{fr} = \mathbf{true}$ // G_5-G_7
35 $\quad \pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_5-G_7
36 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ // G_5-G_7
37 $\quad \pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_5-G_7
38 **else** // G_5-G_7
39 $\quad \pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_5-G_7
40 $z \coloneqq (z_1, ..., z_\ell) \coloneqq (u_1 \star x_1^\mathsf{S}, ..., u_\ell \star x_\ell^\mathsf{S})$
41 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$
42 $\pi_\mathsf{U}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}), K, \mathbf{true})$
43 **return true**

SENDINIT$(\mathsf{S}, t, \mathsf{U})$

44 **if** $\pi_\mathsf{S}^t \neq \bot$ **return** $\bot$
45 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
46 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$ // G_0-G_2
47 $x^\mathsf{S} \coloneqq (x_1^\mathsf{S}, ..., x_\ell^\mathsf{S}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$ // G_0-G_2
48 $\mathsf{com} \coloneqq \mathsf{G}(x^\mathsf{S})$ // G_0-G_2
49 $\pi_\mathsf{S}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, \bot, x^\mathsf{S}, \mathsf{com}), \bot, \bot)$ // G_0-G_2
50 $\mathsf{com} \xleftarrow{\$} \{0,1\}^\lambda$ // G_3-G_7
51 **if** $\exists x^\mathsf{S}$ s. t. $T_\mathsf{G}[x^\mathsf{S}] = \mathsf{com}$ // G_3-G_7
52 $\quad$ **return** $\bot$ // G_3-G_7
53 $T_\mathsf{G}[\diamond] \coloneqq \mathsf{com}$ // G_3-G_7
54 $\pi_\mathsf{S}^t \coloneqq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$ // G_3-G_7
55 $\pi_\mathsf{S}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_5-G_7
56 **return** $(\mathsf{S}, \mathsf{com})$

SENDRESP$(\mathsf{U}, t, \mathsf{S}, \mathsf{com})$

57 **if** $\pi_\mathsf{U}^t \neq \bot$ **return** $\bot$
58 **if** $\nexists x^\mathsf{S}$ s. t. $T_\mathsf{G}[x^\mathsf{S}] = \mathsf{com}$ // G_4-G_7
59 $\quad \pi_\mathsf{U}^t.\mathrm{acc} \coloneqq \mathbf{false}$ // G_4-G_7
60 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
61 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
62 $x^\mathsf{U} \coloneqq (x_1^\mathsf{U}, ..., x_\ell^\mathsf{U}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
63 $\pi_\mathsf{U}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, \bot, \mathsf{com}), \bot, \bot)$
64 $\pi_\mathsf{U}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_5-G_7
65 **return** $(\mathsf{U}, x^\mathsf{U})$

SENDTERMINIT$(\mathsf{S}, t, \mathsf{U}, x^\mathsf{U})$

66 **if** $\pi_\mathsf{S}^t \neq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, \bot, x^\mathsf{S}, \mathsf{com}), \bot, \bot)$ // G_0-G_2
67 $\quad$ **return** $\bot$ // G_0-G_2
68 **if** $\pi_\mathsf{S}^t \neq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$ // G_3-G_7
69 $\quad$ **return** $\bot$ // G_3-G_7
70 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
71 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$ // G_3-G_7
72 $x^\mathsf{S} \coloneqq (x_1^\mathsf{S}, ..., x_\ell^\mathsf{S}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$ // G_3-G_7
73 **if** $T_\mathsf{G}[x^\mathsf{S}] \neq \bot$ // G_3-G_7
74 $\quad \mathbf{bad}_{\text{hide}} \coloneqq \mathbf{true}$; **return** $\bot$ // G_3-G_7
75 Replace $\diamond$ in $T_\mathsf{G}[\diamond] \coloneqq \mathsf{com}$ with x^S // G_3-G_7
76 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_\mathsf{P}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com})$ // G_1-G_7
77 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_7
78 $\quad$ **return** $\bot$ // G_1-G_7
79 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ // G_5-G_7
80 $\quad \pi_\mathsf{S}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_5-G_7
81 **else** // G_5-G_7
82 $\quad \pi_\mathsf{S}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_5-G_7
83 $z \coloneqq (z_1, ..., z_\ell) \coloneqq (s_1 \star x_1^\mathsf{U}, ..., s_\ell \star x_\ell^\mathsf{U})$
84 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$
85 $\pi_\mathsf{S}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^\mathsf{U}, x^\mathsf{S}, \mathsf{com}), K, \mathbf{true})$
86 **return** $(\mathsf{S}, x^\mathsf{S})$

Figure 19: Games G_0-G_7 for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{G}, \mathsf{H}\}$. Oracles CORRUPT, REVEAL, TEST, G and H are defined in Figure 20.

Note that when $\mathbf{bad}_{\text{coll}}$ is not raised, each instance is unique and has at most one partner. In order to bound $\mathbf{bad}_{\text{coll}}$, recall that the trace of an oracle π_P^t consists of $(\mathsf{U}, \mathsf{S}, x^\mathsf{U} = (x_1^\mathsf{U}, ..., x_\ell^\mathsf{U}), x^\mathsf{S} = (x_1^\mathsf{S}, ..., x_\ell^\mathsf{S}), \mathsf{com})$, where the user message consists of x^U and the server messages consist of $(x^\mathsf{S}, \mathsf{com})$. When the game chooses x^U, then the probability that it collides with one particular other user instances is $|\mathcal{G}|^{-\ell}$ as all group elements must be the same. On the server side, the commitment is determined by the

```
CORRUPT(U, S)
00 if (U, S) ∈ C return ⊥
01 for P ∈ {U, S}
02   if ∃t s.t. π_P^t.test = true
     and ∄P' ∈ U ∪ S, t' s.t. Partner(π_P^t, π_P'^t') = 1
03     return ⊥
04   ∀π_P^t : if ∄P' ∈ U ∪ S, t' s.t. Partner(π_P^t, π_P'^t') = 1   // G5-G7
05     π_P^t.fr = false                                            // G5-G7
06 C := C ∪ {(U, S)}
07 return pw_US

REVEAL(P, t)
08 if π_P^t.acc ≠ true or π_P^t.test = true
09   return ⊥
10 if ∃P' ∈ U ∪ S, t' s.t. Partner(π_P^t, π_P'^t') = 1
   and π_P'^t'.test = true
11   return ⊥
12 ∀(P', t') s.t. π_P'^t'.tr = π_P^t.tr                            // G5-G7
13   π_P'^t'.fr := false                                           // G5-G7
14 return π_P^t.K

TEST(P, t))
15 if Fresh(π_P^t) = false return ⊥                                // G0-G4
16 if π_P^t.fr = false return ⊥                                    // G5-G7
17 K_0^* := REVEAL(P, t)
18 if K_0^* = ⊥ return ⊥
19 K_1^* ←$ K
20 π_P^t.test := true
21 return K_β^*

G(x^S)
22 if T_G[x^S] = com ≠ ⊥
23   return com
24 T_G[x^S] ←$ {0,1}^λ
25 if ∃x^S' s.t. T_G[x^S'] = T_G[x^S]                              // G2-G7
26   bad_bind := true; return ⊥                                    // G2-G7
27 return T_G[x^S]

H(U, S, x^U, x^S, com, pw, z)
28 if T_H[U, S, x^U, x^S, com, pw, z] = K ≠ ⊥
29   return K
30 T_H[U, S, x^U, x^S, com, pw, Z] ←$ K
31 return T_H[U, S, x^U, x^S, com, pw, z]
```

Figure 20: Oracles CORRUPT, REVEAL, TEST, G and H for games G_0-G_7 in Figure 19.

choice of x^{S} and when the game chooses x^{S} , then the probability that this instance collides with one particular other server instances is $|\mathcal{G}|^{-\ell}$ as well. As there are at most $q_s + q_e$ queries, this yields

$$\Pr[\mathbf{bad}_{\text{coll}}] \leq \binom{q_s + q_e}{2} \cdot \frac{1}{|\mathcal{G}|^{\ell}} \leq \frac{(q_s + q_e)^2}{|\mathcal{G}|^{\ell}} \quad .$$

GAME G_2. In G_2 we raise flag $\mathbf{bad}_{\text{bind}}$ if two different inputs to the random oracle G return the same commitment (line 26). This is to ensure that the adversary cannot open a commitment to a different value, which might depend on previous messages. The number of queries to G is bounded by $q_{\mathsf{G}} + q_s + q_e$, thus we have

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{bind}}] \leq \frac{(q_{\mathsf{G}} + q_s + q_e)^2}{2^{\lambda}} \quad .$$

GAME G_3. In G_3 we choose the commitment uniformly at random when a session is initiated with a SENDINIT query (line 50). Then we can choose x^{S} only later in SENDTERMINIT (line 71). However, we have to take into account some subtleties.

First, in the previous games we ensured that traces are unique and also the same commitment is not chosen twice. In this case G would have returned $\perp$ in G_2, so now we take care of this explicitly in SENDINIT. In particular, we also return $\perp$ whenever there already exists an entry in T_{G} that evaluates to the commitment chosen in SENDINIT (line 52). We add an entry to T_{G} with a placeholder $\diamond$ as input (line 53) to avoid that the commitment is chosen again. (Essentially, the checks in lines 25 and 51 also consider $x^{\mathsf{S}} = \diamond$.) Also we can now only save the commitment in the trace (line 54) and will adapt the initial check in SENDTERMINIT (line 68).

Second, we have to take care of the case that the adversary has already issued a query to G on x^{S} before these values are chosen in SENDTERMINIT because then we

cannot assign com to this input. We cover this in event $\mathbf{bad}_{\text{hide}}$ (line 74). If $\mathbf{bad}_{\text{hide}}$ does not happen, we can overwrite the entry with the placeholder in T_{G} with the correct input (line 75).

Note that $\mathcal{A}$ can only distinguish these changes if $\mathbf{bad}_{\text{hide}}$ occurs. The probability that $\mathbf{bad}_{\text{hide}}$ occurs for one particular instance can be bounded by $\frac{q_{\mathsf{G}}}{|\mathcal{G}|^{\ell}}$ since x^{S} is fresh. For at most q_s instances, this yields

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{hide}}] \leq \frac{q_{\mathsf{G}} q_s}{|\mathcal{G}|^{\ell}} \ .$$

Game G_4. Whenever $\mathcal{A}$ sends a commitment com in G_4 which was not computed using G, i.e., there does not exist an entry x^{S} in T_{G} such that $T_{\mathsf{G}}[x^{\mathsf{S}}] = \mathsf{com}$, we expect that this instance will reject the key and we immediately set the acc flag to **false** (line 59). Although SendResp will still output the first message, as soon as SendTermResp is queried, the game will return $\perp$ as the initial check will fail (line 27).

This change is only observable if $\mathcal{A}$ finds a correct input to G after it has sent the commitment. Then the instance will accept the key in G_3, but not in G_4. As there are at most $q_{\mathsf{G}} + q_s + q_e$ queries to G and q_s instances, we can upper bound the difference by

$$|\Pr[\mathsf{G}_4 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \frac{(q_{\mathsf{G}} + q_s + q_e) q_s}{2^{\lambda}} \ .$$

Game G_5. In game G_5, we make the freshness explicit. To each oracle π_{P}^t, we assign an additional variable $\pi_{\mathsf{P}}^t.\mathsf{fr}$ which is updated during the game. In particular, all instances used in execute queries are marked as fresh (line 25).

A server instance is fresh if the password was not corrupted yet (line 80). Otherwise, it is not fresh (line 82). A user instance is fresh if it has a fresh partner instance (line 35) or the password was not corrupted yet (line 37). Otherwise, it is not fresh (line 39). If $\mathcal{A}$ issues a Corrupt query later, the freshness variable will also be updated (line 05). When the session key of an instance is revealed, this instance and its potential partner instance are marked as not fresh (line 13). On a query to test, the game then only checks the freshness variable (line 16).

These are only a conceptual changes, hence

$$\Pr[\mathsf{G}_5 \Rightarrow 1] = \Pr[\mathsf{G}_4 \Rightarrow 1] \ .$$

Game G_6. In game G_6, we choose random keys for all instances queried to Execute (line 22). We construct adversary $\mathcal{B}_1$ against $\mathsf{GA\text{-}StCDH}$ in Figure 21 and show that

$$|\Pr[\mathsf{G}_6 \Rightarrow 1] - \Pr[\mathsf{G}_5 \Rightarrow 1]| \leq \mathsf{Adv}_{\mathsf{EGAT}}^{\mathsf{GA\text{-}StCDH}}(\mathcal{B}_1) \ .$$

Adversary $\mathcal{B}_1$ inputs a $\mathsf{GA\text{-}StCDH}$ challenge $(x, y) = (g \star \tilde{x}, h \star \tilde{x})$ and has access to a decision oracle GA-DDH$(x, \cdot, \cdot)$. First, it generates the crs elements (x_0, x_1) as in game G_6 and then runs adversary $\mathcal{A}$. Queries to Execute are simulated as follows: For $i \in [\ell]$ it chooses random group elements u_i and s_i for user and server instances, but instead of using (x_0, x_1) to compute the set elements, $\mathcal{B}_1$ uses x for the user instance

$\mathcal{B}_1^{\text{GA-DDH}(x,\cdot,\cdot)}(x, y)$
00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T_{\mathsf{H}}, T_{\mathsf{G}}) := (\emptyset, \emptyset)$
03 $\beta \xleftarrow{\$} \{0,1\}$
04 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
05 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
06 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
07 Stop.

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)$
08 **if** $\exists (u_1, ..., u_\ell, s_1, ..., s_\ell)$ s. t.
$\quad (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, u_1, ..., u_\ell, s_1, ..., s_\ell) \in T_{\mathrm{e}}$
09 $\quad (b_1, ..., b_\ell) := \mathsf{pw}$
10 $\quad$ **for** $i \in [\ell]$
11 $\quad\quad$ **if** GA-DDH$(x, x_i^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star z_i) = 1$
12 $\quad\quad\quad$ Stop with $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i$
13 **if** $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z] = K \neq \bot$
14 $\quad$ **return** K
15 $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, Z] \xleftarrow{\$} \mathcal{K}$
16 **return** $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$
17 **if** $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
18 $\quad$ **return** $\bot$
19 $(b_1, ..., b_\ell) := \mathsf{pw}_{\mathsf{US}}$
20 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
21 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
22 $x^{\mathsf{U}} := (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) := (u_1 \star x, ..., u_\ell \star x)$
23 $x^{\mathsf{S}} := (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) := (s_1 \star y, ..., s_\ell \star y)$
24 $\mathsf{com} := \mathsf{G}(x^{\mathsf{S}})$
25 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
26 $\quad$ **return** $\bot$
27 $\forall z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$
28 $\quad$ **for** $i \in [\ell]$
29 $\quad\quad$ **if** GA-DDH$(x, x_i^{\mathsf{S}}, (u_i^{-1} \cdot g_{b_i}) \star z_i) = 1$
30 $\quad\quad\quad$ Stop with $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i$
31 $T_{\mathrm{e}} := T_{\mathrm{e}} \cup \{\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, u_1, ..., u_\ell, s_1, ..., s_\ell\}$
32 $K \xleftarrow{\$} \mathcal{K}$
33 $\pi_{\mathsf{U}}^{t_0} := ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
34 $\pi_{\mathsf{S}}^{t_1} := ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
35 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) := (\mathbf{true}, \mathbf{true})$
36 **return** $(\mathsf{U}, x^{\mathsf{U}}, \mathsf{S}, (\mathsf{com}, x^{\mathsf{S}}))$

Figure 21: Adversary $\mathcal{B}_1$ against GA-StCDH for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{G}, \mathsf{H}\}$. Oracles SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST and G are defined as in G_5. Lines written in blue show how $\mathcal{B}_1$ simulates the game.

(line 22) and y for the server instance (line 23), independent of the password bits b_i. We can rewrite this as

$$x_i^{\mathsf{U}} = u_i \star x = (u_i \cdot g) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i} \cdot g_{b_i}^{-1}) \star \tilde{x} = \underbrace{(u_i \cdot g \cdot g_{b_i}^{-1})}_{u_i'} \star x_{b_i} \ ,$$

where u_i' is the group element that the user actually needs to compute the session key. In the same way, $s_i' = s_i \cdot h \cdot g_{b_i}^{-1}$. Note that z_i is implicitly set to

$$z_i = (u_i' \cdot s_i') \star x_{b_i} = u_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ . \tag{2}$$

We want to choose a random session key now, but before that we check if there has been a query to the random oracle H that matches the session key (lines 27-30). We iterate over the entries in T_{H}, where U, S, x^{U}, x^{S} and $\mathsf{pw}_{\mathsf{US}}$ match, and check if one of the entries in z is correct. More precisely, for all $i \in [\ell]$, we check whether

$$\begin{aligned} \mathsf{GA\text{-}CDH}(x, x_i^{\mathsf{S}}) = (u_i^{-1} \cdot g_{b_i}) \star z_i \ &\Leftrightarrow \ \mathsf{GA\text{-}CDH}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = g_{b_i} \star z_i \\ &\Leftrightarrow \ \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = z_i \end{aligned}$$

using the decision oracle GA-DDH$(x, \cdot, \cdot)$.

If one z_i is correct, $\mathcal{B}_1$ aborts and outputs the solution $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i = (g \cdot h) \star \tilde{x}$ (cf. Equation (2)).

Otherwise, we store the values u_i and s_i in list T_{e} together with the trace and the password (line 31) and choose a session key uniformly at random. We need list T_{e} to

identify relevant queries to H. In particular, if the trace and password appear in a query, we retrieve the values u_i and s_i to check whether the provided z_i are correct. We do this in the same way as described above using the decision oracle (lines 08-12). If the oracle returns 1 for any i, $\mathcal{B}_1$ aborts and outputs $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_i$.

Game G_7. In game G_7, we remove the password from execute queries. In particular, we do not compute x^{U} and x^{S} to the basis x_{b_i}, but simply use $\tilde{x}$ (lines 16, 17). Note that the values have the same distribution as in the previous game. Also, the group elements u and s are not used to derive the key. Hence, this change is not observable by $\mathcal{A}$ and

$$\Pr[\mathsf{G}_7 \Rightarrow 1] = \Pr[\mathsf{G}_6 \Rightarrow 1] \ .$$

Game G_8. G_8 is given in Figure 22. In this game we want to replace the session keys by random for all fresh instances in oracles SendTermInit and SendTermResp (lines 57, 79). Therefore, we introduce an additional independent random oracle T_{s} which maps only the trace of an instance to a key (lines 58, 80). We keep partner instances consistent, i.e., in case the adversary queries SendTermResp for a user instance and there exists a fresh partner instance, then we look in T_{s} for the corresponding key and assign it to this instance as well (line 74). For all instances that are not fresh, we simply compute the correct key using random oracle H (lines 61-62, 83-84). If a session is fresh and there is an inconsistency between T_{H} and T_{s}, we raise flag **bad**. This happens in the following cases:

- a server instance is about to compute the session key, the password was not corrupted, but there already exists an entry in T_{H} with the correct password and z (lines 55-56).
- a user instance is about to compute the session key, there exists no partner instance and the password was not corrupted, but there already exists an entry in T_{H} with the correct password and z (lines 77-78).
- the random oracle is queried on some trace that appears in T_{s} together with the correct password and z (lines 02-11). At this point, we also check if the password was corrupted in the meantime and if this is the case and the adversary issues the correct query, we simply output the key stored in T_{s} (line 09) as this instance cannot be tested. This case corresponds to perfect forward secrecy which we cover in Appendix F.3.

Note that when **bad** is not raised, there is no difference between G_7 and G_8. Hence,

$$|\Pr[\mathsf{G}_8 \Rightarrow 1] - \Pr[\mathsf{G}_7 \Rightarrow 1]| \leq \Pr[\mathsf{G}_8 \Rightarrow \mathbf{bad}] \ .$$

Game G_9. G_9 is given in Figure 24. In this game we remove the password from send queries and generate passwords as late as possible, that is either when the adversary issues a corrupt query (line 37) or after it has stopped with output β' (line 07). In SendResp and SendTermInit we still choose group elements u_i and s_i uniformly at random, but now compute x_i^{U} and x_i^{S} using the origin element (line 26 and 42). Thus, depending on which password is chosen afterwards, we implicitly set

$$x_i^{\mathsf{U}} = u_i \cdot \tilde{x} = (u_i \cdot g_0^{-1}) \star x_0 = (u_i \cdot g_1^{-1}) \star x_1$$

GAMES G_8

00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T_{\mathsf{H}}, T_{\mathsf{G}}) \coloneqq (\emptyset, \emptyset)$
03 $\beta \xleftarrow{\$} \{0,1\}$
04 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
05 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
06 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
07 **return** $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$
08 **if** $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
09 $\quad$ **return** $\bot$
10 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
11 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
12 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$
13 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
14 $\mathsf{com} \coloneqq \mathsf{G}(x^{\mathsf{S}})$
15 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
16 $\quad$ **return** $\bot$
17 $K \xleftarrow{\$} \mathcal{K}$
18 $\pi_{\mathsf{U}}^{t_0} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
19 $\pi_{\mathsf{S}}^{t_1} \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
20 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$
21 **return** $(\mathsf{U}, x^{\mathsf{U}}, \mathsf{S}, (\mathsf{com}, x^{\mathsf{S}}))$

SENDINIT$(\mathsf{S}, t, \mathsf{U})$
22 **if** $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
23 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
24 $\mathsf{com} \xleftarrow{\$} \{0,1\}^\lambda$
25 **if** $\exists x^{\mathsf{S}}$ s. t. $T_{\mathsf{G}}[x^{\mathsf{S}}] = \mathsf{com}$ **return** $\bot$
26 $T_{\mathsf{G}}[\diamond] \coloneqq \mathsf{com}$
27 $\pi_{\mathsf{S}}^t \coloneqq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$
28 $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
29 **return** $(\mathsf{S}, \mathsf{com})$

SENDRESP$(\mathsf{U}, t, \mathsf{S}, \mathsf{com})$
30 **if** $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
31 **if** $\nexists x^{\mathsf{S}}$ s. t. $T_{\mathsf{G}}[x^{\mathsf{S}}] = \mathsf{com}$
32 $\quad \pi_{\mathsf{U}}^t.\mathsf{acc} \coloneqq \mathbf{false}$
33 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
34 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
35 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
36 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot, \mathsf{com}), \bot, \bot)$
37 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
38 **return** $(\mathsf{U}, x^{\mathsf{U}})$

$\mathsf{G}(x^{\mathsf{S}})$
39 **if** $T_{\mathsf{G}}[x^{\mathsf{S}}] = \mathsf{com} \neq \bot$ **return** com
40 $T_{\mathsf{G}}[x^{\mathsf{S}}] \xleftarrow{\$} \{0,1\}^\lambda$
41 **if** $\exists x^{\mathsf{S}'}$ s. t. $T_{\mathsf{G}}[x^{\mathsf{S}'}] = T_{\mathsf{G}}[x^{\mathsf{S}}]$ **return** $\bot$
42 **return** $T_{\mathsf{G}}[x^{\mathsf{S}}]$

SENDTERMINIT$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$
43 **if** $\pi_{\mathsf{S}}^t \neq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$
44 $\quad$ **return** $\bot$
45 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
46 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
47 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$
48 **if** $T_{\mathsf{G}}[x^{\mathsf{S}}] \neq \bot$
49 $\quad$ **return** $\bot$
50 Replace $\diamond$ in $T_{\mathsf{G}}[\diamond] \coloneqq \mathsf{com}$ with x^{S}
51 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
52 $\quad$ **return** $\bot$
53 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
54 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
55 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** $z_i = s_i \star x_i^{\mathsf{U}}\ \forall i \in [\ell]$
56 $\quad\quad$ **bad** $\coloneqq$ **true**
57 $\quad K \xleftarrow{\$} \mathcal{K}$
58 $\quad T_{\mathrm{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
59 **else**
60 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
61 $\quad z \coloneqq (z_1, ..., z_\ell) \coloneqq (s_1 \star x_1^{\mathsf{U}}, ..., s_\ell \star x_\ell^{\mathsf{U}})$
62 $\quad K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$
63 $\pi_{\mathsf{S}}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
64 **return** $(\mathsf{S}, x^{\mathsf{S}})$

SENDTERMRESP$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$
65 **if** $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot, \mathsf{com}), \bot, \bot)$
66 $\quad$ **return** $\bot$
67 **if** $\mathsf{G}(x^{\mathsf{S}}) \neq \mathsf{com}$
68 $\quad \pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), \bot, \mathbf{false})$
69 $\quad$ **return** $\bot$
70 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
71 $\quad$ **return** $\bot$
72 **if** $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
73 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
74 $\quad (\mathsf{S}, (s_1, ..., s_\ell), K) \coloneqq T_{\mathrm{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}]$
75 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
76 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
77 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** $z_i = u_i \star x_i^{\mathsf{S}}\ \forall i \in [\ell]$
78 $\quad\quad$ **bad** $\coloneqq$ **true**
79 $\quad K \xleftarrow{\$} \mathcal{K}$
80 $\quad T_{\mathrm{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{U}, (u_1, ..., u_\ell), K)$
81 **else**
82 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
83 $\quad z \coloneqq (z_1, ..., z_\ell) \coloneqq (u_1 \star x_1^{\mathsf{S}}, ..., u_\ell \star x_\ell^{\mathsf{S}})$
84 $\quad K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$
85 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
86 **return true**

Figure 22: Game G_8 for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{G}, \mathsf{H}\}$. Oracles REVEAL, CORRUPT and TEST are defined as in Figure 19. Random oracle H is defined in Figure 23. Differences to game G_7 are highlighted in blue.

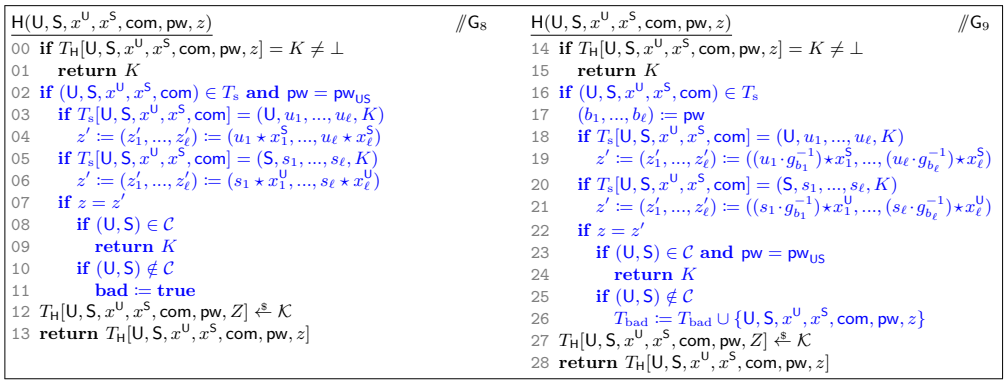

Figure 23: Random oracle H for games G_8 and G_9 in Figures 22 and 24, respectively.

and analogously for x_i^{S}. For all instances that are not fresh, we have to compute the real session key using $z_i = (s_i \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{U}}$ (line 60) or $z_i = (u_i \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{S}}$ (line 86). Note that the password is already defined for these instances.

Recall that event **bad** in game G_8 is raised whenever there is an inconsistency in the random oracle queries and the keys of fresh instances. In this game, we split event **bad** into two different events:

- $\mathbf{bad}_{\mathsf{pw}}$ captures the event that there exists more than one valid entry in T_{H} for the same trace of a fresh instance, but different passwords.
- $\mathbf{bad}_{\text{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen and captures the event that there exists a valid entry in T_{H} for the trace of a fresh instance and the correct password, where the password was not corrupted when the query to H was made.

To identify the different events, we introduce a new set T_{bad}. For all fresh instances in SendTermInit and SendTermResp, we now iterate over all entries in T_{H} that contain the corresponding trace. We check if the given password and z are valid for this trace by computing the real values z' in the same way as for non-fresh instances. If $z = z'$, we add this entry to the set T_{bad} (lines 50-54, 76-80). We essentially do the same when the random oracle H is queried on a trace that appears in T_{s}. Here, the adversary specifies the password and we check if z is valid for that password using the u_i stored in T_{s} for user instances and s_i for server instances. If z is valid and the instance is still fresh, we add the query to T_{bad} (lines 16-26). In case the password was corrupted in the meantime, we output the key stored in T_{s} as introduced in the previous game. After the adversary terminates, we check T_{bad} whether event $\mathbf{bad}_{\mathsf{pw}}$ (line 09) or event $\mathbf{bad}_{\text{guess}}$ (line 12) occurred. We will bound these events below. First note that whenever **bad** is raised in G_8, then either flag $\mathbf{bad}_{\text{guess}}$ or $\mathbf{bad}_{\mathsf{pw}}$ is raised in G_9, thus

$$\Pr[\mathsf{G}_8 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\text{guess}}] \ .$$

Finally, we bound the probabilities of the two events. We start with $\mathbf{bad}_{\mathsf{pw}}$. In Figures 25 and 26, we construct adversary $\mathcal{B}_2$ against ISim-GA-StCDH that simulates G_9. We show that when $\mathbf{bad}_{\mathsf{pw}}$ occurs, then $\mathcal{B}_2$ can solve ISim-GA-StCDH. In the proof we need to

GAMES G_9

00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T_{\mathsf{H}}, T_{\mathsf{G}}) \coloneqq (\emptyset, \emptyset)$
03 $(\mathbf{bad}_{\text{guess}}, \mathbf{bad}_{\mathsf{pw}}) = (\mathbf{false}, \mathbf{false})$
04 $\beta \xleftarrow{\$} \{0,1\}$
05 $\beta' \leftarrow \mathcal{A}^{\mathsf{O}}(x_0, x_1)$
06 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S} \setminus \mathcal{C}$
07 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
08 **if** $\exists \mathsf{pw}, \mathsf{pw}', (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, z, z')$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z) \in T_{\text{bad}}$ **and** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}', z') \in T_{\text{bad}}$
09 $\quad \mathbf{bad}_{\mathsf{pw}} \coloneqq \mathbf{true}$
10 **else**
11 $\quad$ **if** $\exists \mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\text{bad}}$
12 $\quad\quad \mathbf{bad}_{\text{guess}} \coloneqq \mathbf{true}$
13 **return** $[\![\beta = \beta']\!]$

SENDINIT$(\mathsf{S}, t, \mathsf{U})$

14 **if** $\pi^t_{\mathsf{S}} \neq \bot$ **return** $\bot$
15 $\mathsf{com} \xleftarrow{\$} \{0,1\}^\lambda$
16 **if** $\exists x^{\mathsf{S}}$ s. t. $T_{\mathsf{G}}[x^{\mathsf{S}}] = \mathsf{com}$
17 $\quad$ **return** $\bot$
18 $T_{\mathsf{G}}[\diamond] \coloneqq \mathsf{com}$
19 $\pi^t_{\mathsf{S}} \coloneqq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$
20 $\pi^t_{\mathsf{S}}.\mathsf{fr} \coloneqq \mathbf{false}$
21 **return** $(\mathsf{S}, \mathsf{com})$

SENDRESP$(\mathsf{U}, t, \mathsf{S}, \mathsf{com})$

22 **if** $\pi^t_{\mathsf{U}} \neq \bot$ **return** $\bot$
23 **if** $\nexists x^{\mathsf{S}}$ s. t. $T_{\mathsf{G}}[x^{\mathsf{S}}] = \mathsf{com}$
24 $\quad \pi^t_{\mathsf{U}}.\mathsf{acc} \coloneqq \mathbf{false}$
25 $(u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
26 $x^{\mathsf{U}} \coloneqq (x^{\mathsf{U}}_1, ..., x^{\mathsf{U}}_\ell) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$
27 $\pi^t_{\mathsf{U}} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot, \mathsf{com}), \bot, \bot)$
28 $\pi^t_{\mathsf{U}}.\mathsf{fr} \coloneqq \mathbf{false}$
29 **return** $(\mathsf{U}, x^{\mathsf{U}})$

CORRUPT(U, S)

30 **if** $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
31 **for** $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
32 $\quad$ **if** $\exists t$ s. t. $\pi^t_{\mathsf{P}}.\mathsf{test} = \mathbf{true}$ **and** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi^t_{\mathsf{P}}, \pi^{t'}_{\mathsf{P}'}) = 1$
33 $\quad\quad$ **return** $\bot$
34 $\quad \forall \pi^t_{\mathsf{P}}$: **if** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\mathsf{Partner}(\pi^t_{\mathsf{P}}, \pi^{t'}_{\mathsf{P}'}) = 1$
35 $\quad\quad \pi^t_{\mathsf{P}}.\mathsf{fr} = \mathbf{false}$
36 $\mathcal{C} \coloneqq \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
37 $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
38 **return** $\mathsf{pw}_{\mathsf{US}}$

SENDTERMINIT$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$

39 **if** $\pi^t_{\mathsf{S}} \neq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$
40 $\quad$ **return** $\bot$
41 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
42 $x^{\mathsf{S}} \coloneqq (x^{\mathsf{S}}_1, ..., x^{\mathsf{S}}_\ell) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
43 **if** $T_{\mathsf{G}}[x^{\mathsf{S}}] \neq \bot$
44 $\quad$ **return** $\bot$
45 Replace $\diamond$ in $T_{\mathsf{G}}[\diamond] \coloneqq \mathsf{com}$ with x^{S}
46 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi^{t'}_{\mathsf{P}}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
47 $\quad$ **return** $\bot$
48 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
49 $\quad \pi^t_{\mathsf{S}}.\mathsf{fr} \coloneqq \mathbf{true}$
50 $\quad \forall \mathsf{pw}, z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z) \in T_{\mathsf{H}}$
51 $\quad\quad (b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
52 $\quad\quad z' \coloneqq ((s_1 \cdot g_{b_1}^{-1}) \star x^{\mathsf{U}}_1, ..., (s_\ell \cdot g_{b_\ell}^{-1}) \star x^{\mathsf{U}}_\ell)$
53 $\quad\quad$ **if** $z = z'$
54 $\quad\quad\quad T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)\}$
55 $\quad K \xleftarrow{\$} \mathcal{K}$
56 $\quad T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
57 **else**
58 $\quad \pi^t_{\mathsf{S}}.\mathsf{fr} \coloneqq \mathbf{false}$
59 $\quad (b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
60 $\quad z \coloneqq (z_1, ..., z_\ell) \coloneqq ((s_1 \cdot g_{b_1}^{-1}) \star x^{\mathsf{U}}_1, ..., (s_\ell \cdot g_{b_\ell}^{-1}) \star x^{\mathsf{U}}_\ell)$
61 $\quad K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$
62 $\pi^t_{\mathsf{S}} \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
63 **return** $(\mathsf{S}, x^{\mathsf{S}})$

SENDTERMRESP$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$

64 **if** $\pi^t_{\mathsf{U}} \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot, \mathsf{com}), \bot, \bot)$
65 $\quad$ **return** $\bot$
66 **if** $\mathsf{G}(x^{\mathsf{S}}) \neq \mathsf{com}$
67 $\quad \pi^t_{\mathsf{U}} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), \bot, \mathbf{false})$
68 $\quad$ **return** $\bot$
69 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi^{t'}_{\mathsf{P}}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
70 $\quad$ **return** $\bot$
71 **if** $\exists t'$ s. t. $\pi^{t'}_{\mathsf{S}}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$ **and** $\pi^{t'}_{\mathsf{S}}.\mathsf{fr} = \mathbf{true}$
72 $\quad \pi^t_{\mathsf{U}}.\mathsf{fr} \coloneqq \mathbf{true}$
73 $\quad (\mathsf{S}, (s_1, ..., s_\ell), K) \coloneqq T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}]$
74 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
75 $\quad \pi^t_{\mathsf{U}}.\mathsf{fr} \coloneqq \mathbf{true}$
76 $\quad \forall \mathsf{pw}, z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z) \in T_{\mathsf{H}}$
77 $\quad\quad (b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
78 $\quad\quad z' \coloneqq ((u_1 \cdot g_{b_1}^{-1}) \star x^{\mathsf{S}}_1, ..., (u_\ell \cdot g_{b_\ell}^{-1}) \star x^{\mathsf{S}}_\ell)$
79 $\quad\quad$ **if** $z = z'$
80 $\quad\quad\quad T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)\}$
81 $\quad K \xleftarrow{\$} \mathcal{K}$
82 $\quad T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
83 **else**
84 $\quad \pi^t_{\mathsf{U}}.\mathsf{fr} \coloneqq \mathbf{false}$
85 $\quad (b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
86 $\quad z \coloneqq (z_1, ..., z_\ell) \coloneqq ((u_1 \cdot g_{b_1}^{-1}) \star x^{\mathsf{S}}_1, ..., (u_\ell \cdot g_{b_\ell}^{-1}) \star x^{\mathsf{S}}_\ell)$
87 $\quad K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z)$
88 $\pi^t_{\mathsf{U}} \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
89 **return true**

Figure 24: Game G_9 for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathsf{O} \coloneqq \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, $\mathsf{G}, \mathsf{H}\}$. Oracles REVEAL and TEST are defined as in game G_7 in Figure 19. Oracles EXECUTE and G are defined as in Figure 22. Oracle H is defined in Figure 23. Differences to game G_8 are highlighted in blue.

guess the instance and the password bit for which $\mathbf{bad}_{\mathsf{pw}}$ happens. Hence,

$$\Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] \leq q_s \ell \cdot \mathsf{Adv}^{\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2) \ .$$

Recall that in the ISim-GA-StCDH problem, $\mathcal{B}_2$ must commit to some $y \in \mathcal{X}$ to receive the challenge $x = g \star \tilde{x}$ for $g \xleftarrow{\$} \mathcal{G}$. Thus, adversary $\mathcal{B}_2$ will first guess a send query τ^* and a password bit i^* for which it will solve the problem. On a high level, the simulation of G_9 for adversary $\mathcal{A}$ works as follows: on the τ^*-th send query, where $\mathcal{A}$ sends x^{P}, $\mathcal{B}_2$ will output the i^*-th set element $x^{\mathsf{P}}_{i^*}$ as a commitment. It then embeds the challenge x in the i^*'s set element which will be output to $\mathcal{A}$. If in the end, $\mathcal{A}$ has issued two valid queries to H for that trace, where the passwords differ in the i^*'s bit, $\mathcal{B}_2$ can solve the ISim-GA-StCDH problem.

Let us now describe $\mathcal{B}_2$ in more detail. Adversary $\mathcal{B}_2$ inputs (x_0, x_1), where $x_0 = g_0 \star \tilde{x}$ and $x_1 = g_1 \star \tilde{x}$ for group elements $g_0, g_1 \in \mathcal{G}$ chosen uniformly at random. Adversary $\mathcal{B}_2$ also has access to decision oracles $\text{GA-DDH}_{x_0}(\tilde{x}, \cdot, \cdot)$, $\text{GA-DDH}_{x_1}(\tilde{x}, \cdot, \cdot)$. It guesses a send query $\tau^* \xleftarrow{\$} [q_s]$ and a password bit $i^* \xleftarrow{\$} [\ell]$ and then initializes a counter cnt (lines 01-03), before it runs $\mathcal{A}$ on (x_0, x_1).

Apart from increasing the counter, queries to SendInit are simulated exactly as in G_9 as we do not choose any set elements here, but later in SendTermInit.

In SendResp, we also increase the counter and then we choose x^{U} as in G_9. Only if this is the τ^*-th query and the commitment sent by $\mathcal{A}$ was output by G before, then $\mathcal{B}_2$ looks in the list T_{G} to find the corresponding input x^{S} and outputs $x^{\mathsf{S}}_{i^*}$ as commitment y to receive the ISim-GA-StCDH challenge $x = g \star \tilde{x}$ (lines 30-33). It replaces the i^*-th element in x^{U} with x, implicitly defining $u_{i^*} = g$. In order to recognize where the challenge was embedded, $\mathcal{B}_2$ marks the trace of this instance as the target trace tr^* (lines 35-37). From now on, $\mathcal{B}_2$ also has access to decision oracles $\text{GA-DDH}_{x_0}(x, \cdot, \cdot)$, $\text{GA-DDH}_{x_1}(x, \cdot, \cdot)$.

Queries to SendTermInit are simulated similarly (see Figure 26). After increasing the counter, $\mathcal{B}_2$ computes x^{S} and if this is the τ^*-th query, it outputs $x^{\mathsf{U}}_{i^*}$ as commitment to receive x. $x^{\mathsf{S}}_{i^*}$ is then set to x, implicitly setting $s_{i^*} = g$. $\mathcal{B}_2$ also marks the trace as tr^* (lines 05-11). Now $\mathcal{B}_2$ also needs to compute a session key. If the instance is fresh, we must check if there already exists an entry in T_{H} that causes an inconsistency. As in G_9, we iterate over all pw, z in T_{H} that contain the trace of this instance (line 19). In particular, we must check whether z_i satisfies

$$z_i = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x^{\mathsf{U}}_i, x^{\mathsf{S}}_i) = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x^{\mathsf{U}}_i, s_i \star \tilde{x}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(\tilde{x}, x^{\mathsf{U}}_i) = s_i^{-1} \star z_i \ ,$$

which we can do using the GA-DDH oracle and the equation on the right-hand side (lines 24-26). When checking for an inconsistency of tr^*, we need to call the corresponding additional oracle (where the challenge x is fixed) for the i^*-th element (lines 21-23). In this case, we need to check if

$$z_{i^*} = \mathsf{GA\text{-}CDH}_{x_{b_{i^*}}}(x^{\mathsf{U}}_{i^*}, x^{\mathsf{S}}_{i^*}) = \mathsf{GA\text{-}CDH}_{x_{b_{i^*}}}(x^{\mathsf{U}}_{i^*}, x) = \mathsf{GA\text{-}CDH}_{x_{b_{i^*}}}(x, x^{\mathsf{U}}_{i^*}) \ .$$

If all z_i are valid, then we add this entry to T_{bad}.

If the instance is not fresh, i.e., the password is corrupted, then we have to compute the correct key. We check list T_{H} for a valid entry z as explained above and if it exists, we

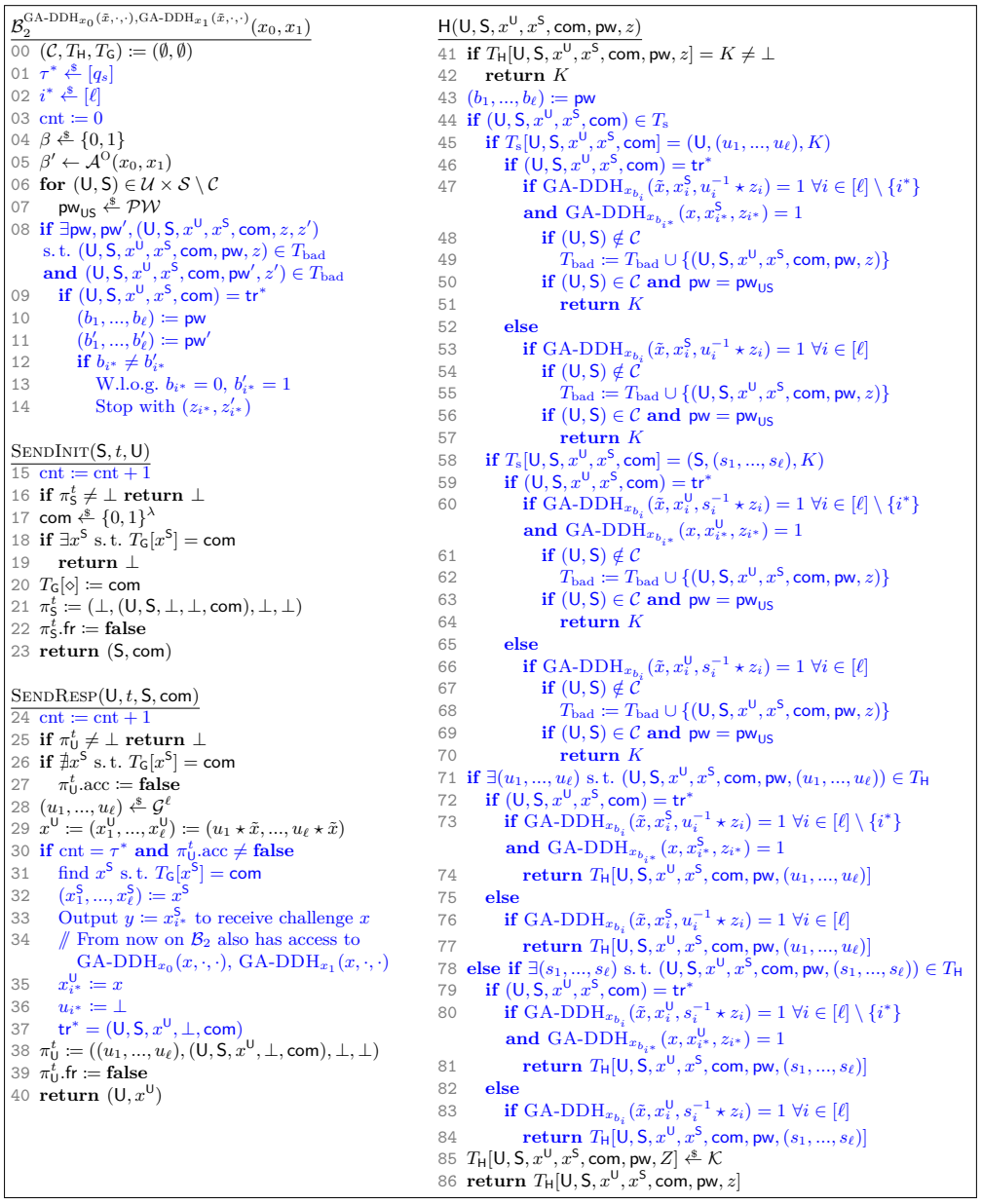

Figure 25: Adversary $\mathcal{B}_2$ against ISim-GA-StCDH for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\text{O} := \{\text{Execute}, \text{SendInit}, \text{SendResp}, \text{SendTermInit}, \text{Reveal}, \text{Corrupt}, \text{Test}, \mathsf{G}, \mathsf{H}\}$. Oracles Execute, Reveal, Corrupt, Test and G are defined as in G_9. Oracles SendTermInit and SendTermResp are defined in Figure 26. Lines written in blue show how $\mathcal{B}_2$ simulates the game.

$\underline{\text{SendTermInit}(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})}$
00 $\mathsf{cnt} \coloneqq \mathsf{cnt} + 1$
01 **if** $\pi_{\mathsf{S}}^t \neq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$
02 **return** $\bot$
03 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
04 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
05 **if** $\mathsf{cnt} = \tau^*$
06 $(x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq x^{\mathsf{U}}$
07 Output $y \coloneqq x_{i^*}^{\mathsf{U}}$ to receive challenge x
08 // From now on $\mathcal{B}_2$ also has access to GA-DDH$_{x_0}(x, \cdot, \cdot)$, GA-DDH$_{x_1}(x, \cdot, \cdot)$
09 $x_{i^*}^{\mathsf{S}} \coloneqq x$
10 $s_{i^*} \coloneqq \bot$
11 $\mathsf{tr}^* = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
12 **if** $T_{\mathsf{G}}[x^{\mathsf{S}}] \neq \bot$
13 **return** $\bot$
14 Replace $\diamond$ in $T_{\mathsf{G}}[\diamond] \coloneqq \mathsf{com}$ with x^{S}
15 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t' \text{ s.t. } \pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
16 **return** $\bot$
17 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
18 $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
19 $\forall \mathsf{pw}, z \text{ s.t. } (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z) \in T_{\mathsf{H}}$
20 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
21 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}) = \mathsf{tr}^*$
22 **if** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \; \forall i \in [\ell] \setminus \{i^*\}$ **and** GA-DDH$_{x_{b_{i^*}}}(x, x_{i^*}^{\mathsf{U}}, z_{i^*}) = 1$
23 $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)\}$
24 **else**
25 **if** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \; \forall i \in [\ell]$
26 $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)\}$
27 $K \xleftarrow{\$} \mathcal{K}$
28 $T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
29 **else**
30 $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
31 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
32 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}) = \mathsf{tr}^*$ **and** $\exists z \text{ s.t. } (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \; \forall i \in [\ell] \setminus \{i^*\}$ **and** GA-DDH$_{x_{b_{i^*}}}(x, x_{i^*}^{\mathsf{U}}, z_{i^*}) = 1$
33 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
34 **else if** $\exists z \text{ s.t. } (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1 \; \forall i \in [\ell]$
35 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
36 **else**
37 $K \xleftarrow{\$} \mathcal{K}$
38 $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, (s_1, ..., s_\ell)] \coloneqq K$
39 $\pi_{\mathsf{S}}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
40 **return** $(\mathsf{S}, x^{\mathsf{S}})$

$\underline{\text{SendTermResp}(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})}$
41 $\mathsf{cnt} \coloneqq \mathsf{cnt} + 1$
42 **if** $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot, \mathsf{com}), \bot, \bot)$
43 **return** $\bot$
44 **if** $\mathsf{G}(x^{\mathsf{S}}) \neq \mathsf{com}$
45 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), \bot, \mathbf{false})$
46 **return** $\bot$
47 **if** $\exists \mathsf{P} \in \mathcal{U}, t' \text{ s.t. } \pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
48 **return** $\bot$
49 **if** $\pi_{\mathsf{U}}^t.\mathsf{tr} = \mathsf{tr}^*$
50 $\mathsf{tr}^* \coloneqq (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
51 **if** $\exists t' \text{ s.t. } \pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
52 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
53 $(\mathsf{S}, (s_1, ..., s_\ell), K) \coloneqq T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}]$
54 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
55 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
56 $\forall \mathsf{pw}, z \text{ s.t. } (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z) \in T_{\mathsf{H}}$
57 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
58 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}) = \mathsf{tr}^*$
59 **if** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \; \forall i \in [\ell] \setminus \{i^*\}$ **and** GA-DDH$_{x_{b_{i^*}}}(x, x_{i^*}^{\mathsf{S}}, z_{i^*}) = 1$
60 $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)\}$
61 **else**
62 **if** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \; \forall i \in [\ell]$
63 $T_{\text{bad}} \coloneqq T_{\text{bad}} \cup \{(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}, z)\}$
64 $K \xleftarrow{\$} \mathcal{K}$
65 $T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{U}, (u_1, ..., u_\ell), K)$
66 **else**
67 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
68 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
69 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}) = \mathsf{tr}^*$ **and** $\exists z \text{ s.t. } (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \; \forall i \in [\ell] \setminus \{i^*\}$ **and** GA-DDH$_{x_{b_{i^*}}}(x, x_{i^*}^{\mathsf{S}}, z_{i^*}) = 1$
70 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
71 **else if** $\exists z \text{ s.t. } (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** GA-DDH$_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1 \; \forall i \in [\ell]$
72 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
73 **else**
74 $K \xleftarrow{\$} \mathcal{K}$
75 $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, (u_1, ..., u_\ell)] \coloneqq K$
76 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
77 **return true**

Figure 26: Oracles SendTermInit and SendTermResp for adversary $\mathcal{B}_2$ in Figure 25.

assign this value to the session key (lines 32-35). Here we also must treat tr^* accordingly. Otherwise, we choose a random key and add a special entry to T_{H}, which instead of z contains the secret group elements s_i (lines 37, 38) so that we can patch the random oracle later.

SendTermInit is simulated analogously, using the secret group elements u_i. Here we must first update tr^* if necessary (line 50).

Now we look at the random oracle queries to H (see Figure 25). If the trace is contained in set T_{s} (line 44) which means the corresponding instance was fresh when the send query was issued, we check if z is valid using the GA-DDH oracle. We do this as described above, depending on whether it is a user or server instance (lines 45, 58) and depending on whether it is tr^* (lines 46, 59) or not (lines 52, 65). In case z is valid, we first check if the instance is still fresh (i.e., the password was not corrupted in the meantime) and if this is the case, we add the query to T_{bad} (lines 49, 55, 62, 68). Otherwise, if the password was corrupted and is specified in the query, we return the session key stored in T_{s} (lines 51, 57, 64, 70).

Next, we check if the query matches a special entry in T_{H} that was added in SendTermInit or SendTermResp for a non-fresh instance, which means we have to output the same key that was chosen before. Again, we can use the GA-DDH oracle and differentiate between user and server instances (lines 71, 78) and tr^*(lines 72, 79).

After $\mathcal{A}$ terminates with output β', $\mathcal{B}_2$ chooses the passwords which have not been generated yet in a Corrupt query (line 07). Then we check for event $\mathbf{bad}_{\mathsf{pw}}$ (line 08). If $\mathbf{bad}_{\mathsf{pw}}$ occurred, then there must be at least two entries in T_{bad} for the same trace and different passwords pw and pw' along with values z and z'. We check if this is the case for the target trace tr^* (line 09) and if the two passwords differ in the i^*-th bit (line 12). In this case $\mathcal{B}_2$ will output z_{i^*} and z'_{i^*} for $b_{i^*} = 0$ and $b'_{i^*} = 1$ (line 14) or it must swap the output. Otherwise, $\mathcal{B}_2$ aborts.

Recall that in case tr^* belongs to a user instance, $\mathcal{B}_2$ committed on $y = x_{i^*}^{\mathsf{S}}$ and embedded the challenge x in $x_{i^*}^{\mathsf{U}}$. To solve the $\mathsf{ISim\text{-}GA\text{-}StCDH}$ problem, $\mathcal{B}_2$ needs to compute

$$y_0 = \mathsf{GA\text{-}CDH}_{x_0}(x, y) = \mathsf{GA\text{-}CDH}_{x_0}(x_{i^*}^{\mathsf{U}}, x_{i^*}^{\mathsf{S}}) = z_{i^*} \text{ if } b_{i^*} = 0,$$
$$y_1 = \mathsf{GA\text{-}CDH}_{x_1}(x, y) = \mathsf{GA\text{-}CDH}_{x_1}(x_{i^*}^{\mathsf{U}}, x_{i^*}^{\mathsf{S}}) = z'_{i^*} \text{ if } b'_{i^*} = 1,$$

which is exactly what is stored in T_{bad}. If tr^* belongs to a server instance, the analysis works analogously.

Therefore, $\mathcal{B}_2$ is successful whenever its guesses are correct and $\mathcal{A}$ issues two queries for the target trace and both password bits. This concludes the analysis of $\mathbf{bad}_{\mathsf{pw}}$.

Next, we analyze event $\mathbf{bad}_{\mathrm{guess}}$. Recall that $\mathbf{bad}_{\mathrm{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen. Hence, for each instance there is at most one entry in T_{bad} and the size of T_{bad} is at most q_s. As all entries were added before the corresponding password was sampled, the probability is bounded by

$$\Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathrm{guess}}] \leq \frac{q_s}{|\mathcal{PW}|} \ .$$

Finally, note that if none of the bad events happens in G_9, all session keys output by Test are uniformly random and the adversary can only guess β. Hence, $\Pr[\mathsf{G}_9 \Rightarrow 1] = \frac{1}{2}$. Collecting the probabilities and applying Lemma 4 yields the bound in Theorem 2. □

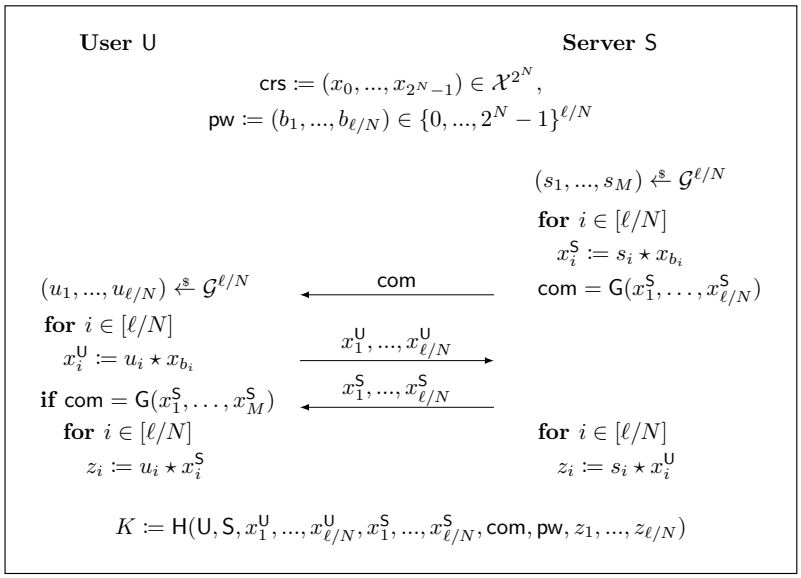

Figure 27: Com-GA-PAKE$_{\ell,N}$ with some $N \mid \ell$.

E.2 Variants of Com-GA-PAKE$_\ell$

In Section 8 we described optimizations that can be used to reduce the number of group action evaluations in an execution of the protocol. Here, we show that Com-GA-PAKE$_\ell$ remains secure when one or both optimizations is applied.

Com-GA-PAKE$_{\ell,N}$. Let N be some positive integer dividing ℓ. We set

$$\mathsf{crs} \coloneqq (x_0, \ldots, x_{2^N-1}) \in \mathcal{X}^{2^N}$$

and divide the password into ℓ/N blocks of length N

$$\mathsf{pw} = (b_1, ..., b_{\ell/N}) \in \{0, ..., 2^N - 1\}^{\ell/N}.$$

The general outline of the protocol does not change. The only difference is that both the server and the user only generate ℓ/N random group elements (instead of ℓ). Hence they only need to perform $2 \cdot \ell/N$ group action evaluations in total. We write Com-GA-PAKE$_{\ell,N}$ for this variant of the protocol. The alterations are summarized in Figure 27.

Theorem 5 (Security of Com-GA-PAKE$_{\ell,N}$)**.** *For any adversary $\mathcal{A}$ against* Com-GA-PAKE$_{\ell,N}$ *that issues at most q_e queries to oracle* Execute *and q_s queries to oracle* SendInit *and* SendResp*, there exist adversary $\mathcal{B}_1$ against* GA-StCDH*, $\mathcal{B}_2$ against* GA-GapCDH *such that*

$$\mathsf{Adv}_{\mathsf{Com\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + \frac{2q_s\ell}{N} \cdot \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)} + \frac{(q_s+q_e)^2}{|\mathcal{G}|^{\ell/N}} + \frac{q_{\mathsf{G}}q_s}{|\mathcal{G}|^{\ell/N}} + \frac{2 \cdot (q_{\mathsf{G}}+q_s+q_e)^2}{2^\lambda} + \frac{q_s}{|\mathcal{PW}|} \ ,$$

where λ is the output length of G in bits.

The proof of Theorem 5 is very similar to the proof of Theorem 2. Therefore we do not give a full proof for the security of $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell,N}$, but shortly explain the difference between the two protocols.

The main difference appears in the analysis of the event $\mathbf{bad}_{\mathsf{pw}}$. Here, it is not possible to construct an adversary against ISim-GA-StCDH as was done in the original proof. Instead, we construct an adversary against 2^{N}ISim-GA-StCDH (Definition 16). This construction is a straightforward adaption of the original construction, therefore we do not give any details here. But we show that the new assumption can be reduced to ISim-GA-StCDH (Theorem 6).

Definition 16 (2^{N}-Interactive Simultaneous GA-StCDH (2^{N}ISim-GA-StCDH))**.** *On input $(x_0 = g_0 \star \tilde{x}, ..., x_{2^N-1} = g_{2^N-1} \star \tilde{x}) \in \mathcal{X}^{2^N}$, the adversary first chooses and commits to some $y \in \mathcal{X}$. After receiving the challenge $x = g \star \tilde{x} \in \mathcal{X}$, the 2^{N}ISim-GA-StCDH problem requires to compute $y_0 = gg_i^{-1} \star y$ and $y_1 = gg_j^{-1} \star y$ for one pair $i \neq j \in \{0, ..., 2^N - 1\}$. For a group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$, we define the advantage function of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{2^{\mathsf{N}}\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{array}{l} i \neq j \\ y_0 = \mathsf{GA\text{-}CDH}_{x_i}(x, y) \\ y_1 = \mathsf{GA\text{-}CDH}_{x_j}(x, y) \end{array} \middle| \begin{array}{c} (g_0, ..., g_{2^N}) \xleftarrow{\$} \mathcal{G}^{2^N} \\ (x_0, ..., x_{2^N-1}) = (g_0 \star \tilde{x}, ..., g_{2^N-1} \star \tilde{x}) \\ y \leftarrow \mathcal{A}^{\mathrm{O}_1}(x_0, ..., x_{2^N-1}) \\ g \xleftarrow{\$} \mathcal{G} \\ x = g \star \tilde{x} \\ (y_0, y_1) \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(x) \end{array} \right],$$

where $\mathrm{O}_1 = \{\text{GA-DDH}_{x_k}(\tilde{x}, \cdot, \cdot)\}_k$, $\mathrm{O}_2 = \{\text{GA-DDH}_{x_k}(x, \cdot, \cdot)\}_k$ for $k \in \{0, ..., 2^N - 1\}$.

Theorem 6 (ISim-GA-StCDH implies 2^{N}ISim-GA-StCDH)**.** *For any adversary $\mathcal{A}$ against 2^{N}ISim-GA-StCDH, there exists adversary $\mathcal{B}$ against ISim-GA-StCDH such that*

$$\mathsf{Adv}^{2^{\mathsf{N}}\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{ISim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}) \ .$$

Proof. We construct adversary $\mathcal{B}$ as follows. On input $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$, for each $i \in \{0, ..., 2^N - 1\}$ $\mathcal{B}$ chooses a random bit $b_i \xleftarrow{\$} \{0, 1\}$, a random group element $h_i \xleftarrow{\$} \mathcal{G}$ and computes $x_i = h_i \star x_{b_i}$. Then it runs $\mathcal{A}$ on input $(x_0, ..., x_{2^N-1})$. When $\mathcal{A}$ commits on y, $\mathcal{B}$ forwards the commitment to receive the challenge x which it gives to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs (i, j) and (y_0, y_1) such that $i \neq j$ and $y_0 = \mathsf{GA\text{-}CDH}_{x_i}(x, y)$, $y_1 = \mathsf{GA\text{-}CDH}_{x_j}(x, y)$. If $b_i = b_j$ which happens with probability $1/2$, then $\mathcal{B}$ aborts. Otherwise, note that $x_i = (h_i g_{b_i}) \star \tilde{x}$ and thus $y_0 = (h_i^{-1} g_{b_i}^{-1} g) \star y$, so $\mathcal{B}$ can compute the solution $y_0' = h_i \star y_0 = g_{b_i}^{-1} g \star y$ and equivalently $y_1' = h_j \star y_1 = g_{b_j}^{-1} g \star y$. If $b_j = 0$ and $b_i = 1$, the output of $\mathcal{B}$ must be swapped.

During the experiment, $\mathcal{A}$ also has access to decision oracles $\text{GA-DDH}_{x_i}(\tilde{x}, \cdot, \cdot)$ and $\text{GA-DDH}_{x_i}(x, \cdot, \cdot)$ for $i \in \{0, ..., 2^N - 1\}$. These can be easily simulated using $\mathcal{B}$'s decision oracles for x_0 and x_1. On a query $\text{GA-DDH}_{x_i}(\tilde{x}, z_1, z_2)$, $\mathcal{B}$ queries its own oracle $\text{GA-DDH}_{x_{b_i}}(\tilde{x}, z_1, h_i \star z_2)$ and fowards the output to $\mathcal{A}$. Analogously, on a query $\text{GA-DDH}_{x_i}(x, z_1, z_2)$, $\mathcal{B}$ queries $\text{GA-DDH}_{x_{b_i}}(x, z_1, h_i \star z_2)$. □

Twisted version of Com-GA-PAKE$_\ell$. Here, we analyze the security of Com-GA-PAKE$^{\mathsf{t}}_\ell$, the twisted version of Com-GA-PAKE$_\ell$, as defined in Section 8.2. In contrast to the situation for X-GA-PAKE$^{\mathsf{t}}_\ell$, Theorem 2 needs to be changed slightly. Before presenting the new theorem, we need to introduce the the following assumption.

Definition 17 (Square GA-GapCDH (Sq-GA-GapCDH))**.** *On input $g \star \tilde{x} \in \mathcal{X}$, the* Sq-GA-GapCDH *problem requires to compute the set element $g^2 \star \tilde{x}$. To an effective group action* XXX $\in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we associate the advantage function of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{Sq\text{-}GA\text{-}GapCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr[\mathcal{A}^{\text{GA-DDH}_*}(g \star \tilde{x}) \Rightarrow g^2 \star \tilde{x}] \ ,$$

where $g \xleftarrow{\$} \mathcal{G}$ and $\mathcal{A}$ has access to a general decision oracle GA-DDH$_*$.

Note that in the CSIDH setting, the square group action computational Diffie-Hellman problem coincides with Problem 3 in [LGd21].

Theorem 7 (Security of Com-GA-PAKE$^{\mathsf{t}}_\ell$)**.** *For any adversary $\mathcal{A}$ against* Com-GA-PAKE$^{\mathsf{t}}_\ell$ *that issues at most q_e execute queries, q_s send queries and at most q_{G} and q_{H} queries to random oracles* G *and* H*, there exist an adversary $\mathcal{B}_1$ against* GA-StCDH *and an adversary $\mathcal{B}_2$ against* Sq-GA-GapCDH *such that*

$$\mathsf{Adv}_{\mathsf{Com\text{-}GA\text{-}PAKE}^{\mathsf{t}}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + q_s \ell \cdot \sqrt{\mathsf{Adv}^{\mathsf{Sq\text{-}GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)} + \frac{(q_s+q_e)^2}{|\mathcal{G}|^\ell} + \frac{q_{\mathsf{G}} q_s}{|\mathcal{G}|^\ell} + \frac{2\cdot(q_{\mathsf{G}}+q_s+q_e)^2}{2^\lambda} + \frac{q_s}{|\mathcal{PW}|} \ ,$$

where λ is the output length of G *in bits.*

Again, we do not provide a full proof of the theorem since most parts are completely analogous to the proof of Theorem 2. For the most part, one can just replace x_1 by x_0^t everywhere in the proof. The only significant difference occurs in the analysis of the event $\mathbf{bad}_{\mathsf{pw}}$. In particular the restriction $x_1 = x_0^t$ does not allow to construct an adversary against ISim-GA-StCDH. Instead, we need to consider the following alteration of ISim-GA-StCDH for the security analysis.

Definition 18 (Twisted Interactive Simultaneous GA-StCDH (TISim-GA-StCDH))**.** *On input $x_0 = g_0 \star \tilde{x} \in \mathcal{X}$, the adversary first chooses and commits to some $y \in \mathcal{X}$. After receiving the challenge $x = g \star \tilde{x} \in \mathcal{X}$, the* (TISim-GA-StCDH) *problem requires to compute $y_0 = g g_0^{-1} \star y, y_1 = g g_0 \star y$. For a group action* XXX $\in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{TISim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{array}{l|c} & g_0 \xleftarrow{\$} \mathcal{G} \\ & x_0 = g_0 \star \tilde{x} \\ y_0 = \mathsf{GA\text{-}CDH}_{x_0}(x,y) & y \leftarrow \mathcal{A}^{\mathrm{O}_1}(x_0) \\ y_1 = \mathsf{GA\text{-}CDH}_{x_0^t}(x,y) & g \xleftarrow{\$} \mathcal{G} \\ & x = g \star \tilde{x} \\ & (y_0, y_1) \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(x) \end{array}\right] ,$$

where $\mathrm{O}_1 = \{\text{GA-DDH}_{x_0}(\tilde{x}, \cdot, \cdot), \text{GA-DDH}_{x_0^t}(\tilde{x}, \cdot, \cdot)\}$ *and* $\mathrm{O}_2 = \{\text{GA-DDH}_{x_0}(x, \cdot, \cdot), \text{GA-DDH}_{x_0^t}(x, \cdot, \cdot)\}$.

Recall that for the proof of Com-GA-PAKE$_\ell$, we showed that the ISim-GA-StCDH is implied by GA-GapCDH (Lemma 4). In the same way, one can show that TISim-GA-StCDH is implied by Sq-GA-GapCDH, more precisely

$$\mathsf{Adv}^{\mathsf{TISim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \sqrt{\mathsf{Adv}^{\mathsf{Sq\text{-}GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B})} \ .$$

This explains the dependence on the new security assumption Sq-GA-GapCDH for this version of Com-GA-PAKE$_\ell$.

F PAKE with Perfect Forward Secrecy

When considering perfect forward secrecy, the security experiment is the same as $\mathsf{Exp}_{\mathsf{PAKE}}$ described in Section 4. However, we replace the forth freshness condition by the following:

3.4 No partner exists and CORRUPT was not queried prior to acceptance.

As a result, the CORRUPT oracle does not need to check anymore whether an instance that has no partner instance has been tested before or not. If it has tested, it must have been fresh at that point which is not changed by a corruption query.

Definition 19 (Security of PAKE with Forward Security)**.** *We define the security experiment as* $\mathsf{Exp}_{\mathsf{PAKE}}$ *with the additional property in the freshness definition. The advantage of an adversary* $\mathcal{A}$ *against a password authenticated key exchange protocol* PAKE *in* $\mathsf{Exp}^{\mathsf{pfs}}_{\mathsf{PAKE}}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{pfs}}_{\mathsf{PAKE}}(\mathcal{A}) := \left| \Pr[\mathsf{Exp}^{\mathsf{pfs}}_{\mathsf{PAKE}} \Rightarrow 1] - \frac{1}{2} \right| \ .$$

F.1 Perfect Forward Secrecy of GA-PAKE$_\ell$

In order to consider perfect forward secrecy, we need an interactive and password-based security assumption. This is similar as in the analysis of SPAKE2 [AB19] for example.

Definition 20 (Password-based GA-StCDH (Pw-GA-StCDH))**.** *On input* $(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x})$*, the* Pw-GA-StCDH *problem requires the adversary to commit to* $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \in \mathcal{X}^\ell$ *and then compute* $z_i = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}) = (g_{b_i}^{-1} \cdot p_i) \star y_i^{\mathsf{P}}$ $\forall i \in [\ell]$ *after receiving the challenge password* $\mathsf{pw} = (b_1, ..., b_\ell) \in \{0,1\}^\ell$*. To an effective group action* $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}\}$*, we define the advantage function of* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{Pw\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[z_i = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}) \ \forall i \in [\ell] \;\middle|\; \begin{array}{c} (g_0, g_1, p_1, ..., p_\ell) \xleftarrow{\$} \mathcal{G}^{\ell+2} \\ (x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ (x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) := (p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x}) \\ (y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) \\ \mathsf{pw} := (b_1, ..., b_\ell) \xleftarrow{\$} \mathcal{PW} \\ (z_1, ..., z_\ell) \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pw}) \end{array} \right] ,$$

$\mathcal{B}^{\text{GA-DDH}_{x_0}(x,\cdot,\cdot),\text{GA-DDH}_{x_1}(x,\cdot,\cdot)}(x, x_0, x_1, \mathsf{pw})$

00 $(p_1, ..., p_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
01 $(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) \coloneqq (p_1 \star x, ..., p_\ell \star x)$
02 $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1, x_1^{\mathsf{P}}, ...x_\ell^{\mathsf{P}})$
03 $(z_1, ..., z_\ell) \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pw})$
04 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
05 **if** $\text{GA-DDH}_{x_{b_i}}(x, y_i^{\mathsf{P}}, p_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
06 **return** $(1, (\mathsf{pw}, p_1, ..., p_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell))$
07 **return** $(0, \epsilon)$

$\{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, y', z')\}_{i\in[\ell], j\in\{0,1\}}$
08 **return** $\text{GA-DDH}_{x_j}(x, y', p_i^{-1} \star z')$

$\mathcal{C}^{\text{GA-DDH}_{x_0}(x,\cdot,\cdot),\text{GA-DDH}_{x_1}(x,\cdot,\cdot)}(x, x_0, x_1)$

09 $(b^*, \sigma, \sigma') \leftarrow \mathcal{R}_{\mathcal{B}}^{\mathrm{O}'}(x, x_0, x_1)$
10 **if** $b^* = 0$
11 **abort** and **return** $\perp$
12 $(\mathsf{pw}, p_1, ..., p_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell) \coloneqq \sigma$
13 $(\mathsf{pw}', p_1, ..., p_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1', ..., z_\ell') \coloneqq \sigma'$
14 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
15 $(b_1', ..., b_\ell') \coloneqq \mathsf{pw}'$
16 Find i such that $b_i \neq b_i'$
17 **return** $(y_i^{\mathsf{P}}, p_i^{-1} \star z_i, p_i^{-1} \star z_i')$

Figure 28: Adversaries $\mathcal{B}$ and $\mathcal{C}$ against Sim-GA-StCDH for the proof of Lemma 5. $\mathcal{A}$ has access to $\mathrm{O} = \{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, \cdot, \cdot)\}_{i\in[\ell], j\in\{0,1\}}$. Reset algorithm $\mathcal{R}_{\mathcal{B}}$ has access to $\mathrm{O}' = \{\text{GA-DDH}_{x_0}(x, \cdot, \cdot), \text{GA-DDH}_{x_1}(x, \cdot, \cdot)\}$.

where $\mathrm{O} = \{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, \cdot, \cdot)\}_{i\in[\ell], j\in\{0,1\}}$.

Lemma 5. *The simultaneous* GA-StCDH (Sim-GA-StCDH) *implies the password-based* GA-StCDH (Pw-GA-StCDH)*, more precisely*

$$\mathsf{Adv}_{\mathsf{EGA}}^{\mathsf{Pw\text{-}GA\text{-}StCDH}}(\mathcal{A}) \leq \sqrt{\mathsf{Adv}_{\mathsf{EGA}}^{\mathsf{Sim\text{-}GA\text{-}StCDH}}(\mathcal{B})} + \frac{1}{|\mathcal{PW}|}.$$

Proof. Intuitively, the term $1/|\mathcal{PW}|$ comes from the fact that the adversary can trivially solve the problem when guessing the password directly. For the proof we apply the reset lemma (see Lemma 2), where $H = \mathcal{PW}$.

Let $\mathcal{A}$ be an adversary against the Pw-GA-StCDH problem. Now consider adversary $\mathcal{B}$ in Figure 28 that takes as input three set elements (x, x_0, x_1) and a password pw. It also has access to decision oracles. First, $\mathcal{B}$ chooses $(p_1, ...p_\ell) \xleftarrow{\$} \mathcal{G}^\ell$ and computes $x_i^{\mathsf{P}} = p_i \star x$ for all $i \in [\ell]$. Then it runs $\mathcal{A}$ on input $(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}})$ and receives a commitment $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}})$. Now $\mathcal{B}$ gives pw to $\mathcal{A}$ and $\mathcal{A}$ will finally output $(z_1, ..., z_\ell)$. $\mathcal{B}$ checks if the solution is correct using the decision oracles and if this is the case, it outputs $I = 1$ and $\sigma = (\mathsf{pw}, p_1, ..., p_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell)$. Otherwise it outputs $(0, \epsilon)$.

If $\mathcal{A}$ issues a query to a decision oracle $\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, y', z')$ for some $i \in [\ell]$ and $j \in \{0, 1\}$, $\mathcal{B}$ queries its own decision oracle $\text{GA-DDH}_{x_j}(x, y', p_i^{-1} \star z')$ and forwards the output to $\mathcal{A}$.

Let IG be the algorithm that chooses $g, g_0, g_1 \xleftarrow{\$} \mathcal{G}$ and outputs $(x, x_0, x_1) = (g \star \tilde{x}, g_0 \star \tilde{x}, g_1 \star \tilde{x})$. Let acc be defined as in Lemma 2, thus

$$\text{acc} \geq \mathsf{Adv}_{\mathsf{EGA}}^{\mathsf{Pw\text{-}GA\text{-}StCDH}}(\mathcal{A}) \ .$$

Let $\mathcal{R}_{\mathcal{B}}$ be the reset algorithm associated to $\mathcal{B}$ as in Lemma 2 with access to the same decision oracles as $\mathcal{B}$. Then we construct adversary $\mathcal{C}$ against Sim-GA-StCDH as in Figure 28. $\mathcal{C}$ runs the reset algorithm to obtain a bit b^* as well as the two side outputs σ, σ' each consisting of a password, ℓ group elements p_i, ℓ set elements y_i^{P} and ℓ set elements z_i. If $b^* = 0$, it aborts. If the reset algorithm was successful, note

that $\mathsf{pw} \neq \mathsf{pw}'$, but $p_i = p'_i$ and $y_i^{\mathsf{P}} = y_i^{\mathsf{P}'}$ for all $i \in [\ell]$ as we run $\mathcal{B}$ on the same random coins. Now $\mathcal{C}$ looks for the first index i where the two passwords differ and outputs the solution $(y, y_0, y_1) = (y_i^{\mathsf{P}}, p_i^{-1} \star z_i, p_i^{-1} \star z'_i)$ which solves Sim-GA-StCDH as $z_i = \mathsf{GA\text{-}CDH}_{x_0}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}) = (g_0^{-1} \cdot p_i) \star x_i^{\mathsf{P}}$ and $z'_i = \mathsf{GA\text{-}CDH}_{x_1}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}) = (g_1^{-1} \cdot p_i) \star y_i^{\mathsf{P}}$.

Applying Lemma 2, we get the bound stated in Lemma 5. □

Theorem 8 (Perfect Forward Secrecy of $\mathsf{GA\text{-}PAKE}_\ell$)**.** *For any adversary $\mathcal{A}$ against $\mathsf{GA\text{-}PAKE}_\ell$ that issues at most q_e execute queries and q_s send queries and where H is modeled as a random oracle, there exist adversary $\mathcal{B}_1$ against* GA-StCDH *and adversaries $\mathcal{B}_2$, $\mathcal{B}_3$ against* Sim-GA-StCDH *such that*

$$\mathsf{Adv}^{\mathsf{pfs}}_{\mathsf{GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{Sim\text{-}GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_2) + q_s \cdot \sqrt{\mathsf{Adv}^{\mathsf{Sim\text{-}GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_3)} + \frac{2q_s}{|\mathcal{PW}|} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^\ell} \ .$$

Proof. The proof follows the one of Theorem 3 very closely. However, we have to consider that an instance is fresh if the password was not corrupted when the instance accepts. We keep games G_0-G_6 from Figures 13, 15 and 16 almost as they are, with the following differences.

In G_2, we do not update the freshness variable in the CORRUPT oracle in case the instance was tested (line 88, Figure 13) because this is now a valid query.

Recall that in G_5, we raise flag **bad** whenever there is an inconsistency between the random oracle list T and the list of keys from send queries T_{s}. Now we also want keys to be random even if the password was corrupted afterwards, hence we raise **bad** in H in this case as well (line 38, Figure 15), instead of outputting the real session key.

This also translates to G_6, where we will then raise flag $\mathbf{bad}_{\mathrm{pfs}}$ at this point (line 38, Figure 16). Thus, we now have

$$\Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathrm{guess}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathrm{pfs}}] \ ,$$

where $\mathbf{bad}_{\mathsf{pw}}$ and $\mathbf{bad}_{\mathrm{guess}}$ can be bounded as in the proof of Theorem 3.

It remains to bound $\mathbf{bad}_{\mathrm{pfs}}$. Therefore, we construct an adversary $\mathcal{B}_3$ against Pw-GA-StCDH in Figure 29 and show that

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathrm{pfs}}] \leq q_s \cdot \mathsf{Adv}^{\mathsf{Pw\text{-}GA\text{-}StCDH}}_{\mathsf{EGA}}(\mathcal{B}_3) \ .$$

We cannot achieve a tight bound for $\mathbf{bad}_{\mathrm{pfs}}$, as an adversary against Pw-GA-StCDH must commit to ℓ set elements before it receives the challenge password. Thus, adversary $\mathcal{B}_3$ first guesses a send query τ^* which it will use to solve the problem. On a high level, the simulation of G_6 for adversary $\mathcal{A}$ works as follows: on the τ^*-th send query, $\mathcal{B}_3$ will output the ℓ set elements x_i^{P} provided by the assumption and it uses the input to that send query as the commitment y_i^{P}. Then, if $\mathcal{A}$ decides to corrupt the password, it will receive the challenge password and if it then issues the correct query to H, we can solve Pw-GA-StCDH, where we use the decision oracle to simulate the other instances.

We will now describe adversary $\mathcal{B}_3$ in more detail. $\mathcal{B}_3$ inputs set elements $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$ and $x_i^{\mathsf{P}} = (p_i \star \tilde{x})$ for $g_0, g_1, p_i \xleftarrow{\$} \mathcal{G}$ and $i \in [\ell]$. It chooses index τ^* uniformly at random from $[q_s]$ and initializes a counter to keep track of the number of

$\underline{\mathcal{B}_3^{\{\text{GA-DDH}_{x_0}(x_i^{\mathsf{P}},\cdot,\cdot),\text{GA-DDH}_{x_1}(x_i^{\mathsf{P}},\cdot,\cdot)\}_{i\in[\ell]}}(x_0,x_1,x_1^{\mathsf{P}},...,x_\ell^{\mathsf{P}})}$
00 $(\mathcal{C},T,T_{\mathsf{s}}) := (\emptyset,\emptyset,\emptyset)$
01 $\tau^* \xleftarrow{\$} [q_{\mathsf{s}}]$
02 $\mathsf{tr}^* := \bot$
03 $\mathrm{cnt} := 0$
04 $\beta \xleftarrow{\$} \{0,1\}$
05 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0,x_1)$
06 Stop.

$\underline{\textsc{SendInit}(\mathsf{U},t,\mathsf{S})}$
07 $\mathrm{cnt} := \mathrm{cnt}+1$
08 **if** $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
09 $(u_1,...,u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
10 $x^{\mathsf{U}} := (x_1^{\mathsf{U}},...,x_\ell^{\mathsf{U}}) := (u_1 \star x_1^{\mathsf{P}},...,u_\ell \star x_\ell^{\mathsf{P}})$
11 $\pi_{\mathsf{U}}^t := ((u_1,...,u_\ell),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\bot),\bot,\bot)$
12 **return** $(\mathsf{U},x^{\mathsf{U}})$

$\underline{\textsc{Corrupt}(\mathsf{U},\mathsf{S})}$
13 **if** $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$ **return** $\bot$
14 $\mathcal{C} := \mathcal{C} \cup \{(\mathsf{U},\mathsf{S})\}$
15 **if** $\mathsf{tr}^* \neq (\mathsf{U},\mathsf{S},\cdot,\cdot)$
16 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
17 **return** $\mathsf{pw}_{\mathsf{US}}$

$\underline{\mathsf{H}(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},z)}$
18 **if** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},z] = K \neq \bot$
19 $\quad$ **return** K
20 **if** $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}) \in T_{\mathsf{s}}$
21 $\quad (b_1,...,b_\ell) := \mathsf{pw}$
22 $\quad$ **if** $T_{\mathsf{s}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}] = (\mathsf{U},(u_1,...,u_\ell),K)$
23 $\quad\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}},x_i^{\mathsf{S}},u_i^{-1}\star z_i) = 1\ \forall i \in [\ell]$ **and** $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} := \mathsf{pw}_{\mathsf{US}}$
24 $\quad\quad\quad$ **if** $\mathsf{tr}^* \neq (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$ **abort**
25 $\quad\quad\quad$ Output $(u_1^{-1}\star z_1,...,u_\ell^{-1}\star z_\ell)$
26 $\quad$ **if** $T_{\mathsf{s}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}] = (\mathsf{S},(s_1,...,s_\ell),K)$
27 $\quad\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}},x_i^{\mathsf{U}},s_i^{-1}\star z_i) = 1\ \forall i \in [\ell]$ **and** $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} := \mathsf{pw}_{\mathsf{US}}$
28 $\quad\quad\quad$ **if** $\mathsf{tr}^* \neq (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$ **abort**
29 $\quad\quad\quad$ Output $(s_1^{-1}\star z_1,...,s_\ell^{-1}\star z_\ell)$
30 **if** $\exists (u_1,...,u_\ell)$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},(u_1,...,u_\ell)) \in T$
31 $\quad (b_1,...,b_\ell) := \mathsf{pw}$
32 $\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}},x_i^{\mathsf{S}},u_i^{-1}\star z_i) = 1\ \forall i \in [\ell]$
33 $\quad\quad$ **return** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},(u_1,...,u_\ell)]$
34 **else if** $\exists (s_1,...,s_\ell)$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},(s_1,...,s_\ell)) \in T$
35 $\quad (b_1,...,b_\ell) := \mathsf{pw}$
36 $\quad$ **if** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}},x_i^{\mathsf{U}},s_i^{-1}\star z_i) = 1\ \forall i \in [\ell]$
37 $\quad\quad$ **return** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},(s_1,...,s_\ell)]$
38 $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},z] \xleftarrow{\$} \mathcal{K}$
39 **return** $T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw},z]$

$\underline{\textsc{SendResp}(\mathsf{S},t,\mathsf{U},x^{\mathsf{U}})}$
40 $\mathrm{cnt} := \mathrm{cnt}+1$
41 **if** $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
42 $(s_1,...,s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
43 $x^{\mathsf{S}} := (x_1^{\mathsf{S}},...,x_\ell^{\mathsf{S}}) := (s_1 \star x_1^{\mathsf{P}},...,s_\ell \star x_\ell^{\mathsf{P}})$
44 **if** $\exists \mathsf{P} \in \mathcal{U}\cup\mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$
45 $\quad$ **return** $\bot$
46 **if** $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
47 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{true}$
48 $\quad$ **if** $\mathrm{cnt} = \tau^*$
49 $\quad\quad \mathsf{tr}^* := (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$
50 $\quad\quad$ Output $y^{\mathsf{P}} := x^{\mathsf{U}}$ to receive $\mathsf{pw}_{\mathsf{US}}$
51 $\quad K \xleftarrow{\$} \mathcal{K}$
52 $\quad T_{\mathsf{s}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}] := (\mathsf{S},(s_1,...,s_\ell),K)$
53 **else**
54 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{false}$
55 $\quad (b_1,...,b_\ell) := \mathsf{pw}_{\mathsf{US}}$
56 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z) \in T$ **and** $\forall i \in [\ell] : \text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}},x_i^{\mathsf{U}},s_i^{-1}\star z_i) = 1$
57 $\quad\quad K := T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z]$
58 $\quad$ **else**
59 $\quad\quad K \xleftarrow{\$} \mathcal{K}$
60 $\quad\quad T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},(s_1,...,s_\ell)] := K$
61 $\pi_{\mathsf{S}}^t := ((s_1,...,s_\ell),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}),K,\mathbf{true})$
62 **return** $(\mathsf{S},x^{\mathsf{S}})$

$\underline{\textsc{SendTermInit}(\mathsf{U},t,\mathsf{S},x^{\mathsf{S}})}$
63 $\mathrm{cnt} := \mathrm{cnt}+1$
64 **if** $\pi_{\mathsf{U}}^t \neq ((u_1,...,u_\ell),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},\bot),\bot,\bot)$
65 $\quad$ **return** $\bot$
66 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$
67 $\quad$ **return** $\bot$
68 **if** $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
69 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{true}$
70 $\quad (\mathsf{S},(s_1,...,s_\ell),K) := T_{\mathsf{s}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},B]$
71 **else if** $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
72 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{true}$
73 $\quad$ **if** $\mathrm{cnt} = \tau^*$
74 $\quad\quad \mathsf{tr}^* := (\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}})$
75 $\quad\quad$ Output $y^{\mathsf{P}} := x^{\mathsf{S}}$ to receive $\mathsf{pw}_{\mathsf{US}}$
76 $\quad K \xleftarrow{\$} \mathcal{K}$
77 $\quad T_{\mathsf{s}}[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}] := (\mathsf{U},(u_1,...,u_\ell),K)$
78 **else**
79 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{false}$
80 $\quad (b_1,...,b_\ell) := \mathsf{pw}_{\mathsf{US}}$
81 $\quad$ **if** $\exists z$ s. t. $(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z) \in T$ **and** $\forall i \in [\ell] : \text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}},x_i^{\mathsf{S}},u_i^{-1}\star z_i) = 1$
82 $\quad\quad K := T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},z]$
83 $\quad$ **else**
84 $\quad\quad K \xleftarrow{\$} \mathcal{K}$
85 $\quad\quad T[\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}},\mathsf{pw}_{\mathsf{US}},(u_1,...,u_\ell)] := K$
86 $\pi_{\mathsf{U}}^t := ((u_1,...,u_\ell),(\mathsf{U},\mathsf{S},x^{\mathsf{U}},x^{\mathsf{S}}),K,\mathbf{true})$
87 **return true**

Figure 29: Adversary $\mathcal{B}_3$ against Pw-GA-StCDH for the proof of Theorem 8. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTermInit}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{H}\}$. Oracles Execute, Reveal and Test are defined as in Figure 16. Lines written in blue show how $\mathcal{B}_3$ simulates the game.

send queries issued so far (lines 01, 03). The counter is incremented whenever a send query is made (lines 07, 40, 63). As we do not know whether the τ^*-th send query will be issued for a user or a server instance, $\mathcal{B}_3$ will embed the elements x_i^{P} for all instances, similarly to adversary $\mathcal{B}_2$ in Figure 17. In particular,

$$x_i^{\mathsf{U}} = u_i \star x_i^{\mathsf{P}} = (u_i \cdot p_i \cdot g_0^{-1}) \star x_0 = (u_i \cdot p_i \cdot g_1^{-1}) \star x_1$$
$$x_i^{\mathsf{S}} = s_i \star x_i^{\mathsf{P}} = (s_i \cdot p_i \cdot g_0^{-1}) \star x_0 = (s_i \cdot p_i \cdot g_1^{-1}) \star x_1$$

If the password of an instance is not corrupted when the send query is issued, $\mathcal{B}_3$ checks whether this is the τ^*-th query. If this is the case, it marks the trace of this instance as tr^* and outputs x^{U} in SendResp or x^{S} in SendTermInit as the commitment y^{P} (lines 48-50, 73-75) to receive the challenge password $\mathsf{pw}_{\mathsf{US}}$. This is the password which $\mathcal{B}_3$ will output when the adversary corrupts this pair (U, S). For all other corrupt queries, it samples a password uniformly at random (line 16). Instances that are not fresh will be simulated with the decision oracles as in the simulation of adversary $\mathcal{B}_2$. For server instances (lines 56-57), this means that we check if there already exists an entry in T such that

$$z_i = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, s_i \star x_i^{\mathsf{P}}) \;\Leftrightarrow\; \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{U}}) = s_i^{-1} \star z_i \;\; .$$

Equivalently, we check if $\mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{S}}) = u_i^{-1} \star z_i$ for user instances (lines 81-82). If there does not exist an entry yet, $\mathcal{B}_3$ adds a special entry to T (lines 60, 85), which contains the secret group elements s_i or u_i so that we can patch the random oracle later (lines 30-37).

Finally, we look at random oracle queries for all fresh instances contained in T_{s} (lines 20-29). $\mathcal{B}_3$ checks if the provided z is valid using the decision oracles as explained above. Then it checks for event $\mathbf{bad}_{\mathrm{pfs}}$, i.e., if the password was corrupted and it matches the one in the query. If the query now additionally contains the target trace tr^*, we can solve the Pw-GA-StCDH problem by outputting $u_i^{-1} \star z_i$ in case of a user instance or $s_i^{-1} \star z_i$ in case of a server instance. If the trace is not the target trace, $\mathcal{B}_3$ aborts.

This concludes the analysis of $\mathbf{bad}_{\mathrm{pfs}}$. Collecting the bounds and applying Lemma 5 yields the bound in Theorem 8. □

F.2 Perfect Forward Secrecy for X-GA-PAKE$_\ell$

In order to prove forward secrecy of X-GA-PAKE$_\ell$, we need the following interactive problem which is non-tightly implied by the SqInv-GA-StCDH problem.

Definition 21 (Double Password-based GA-StCDH (DPw-GA-StCDH))**.** *On input* $(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}}) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x}, \hat{p}_1 \star \tilde{x}, ..., \hat{p}_\ell \star \tilde{x})$*, the* DPw-GA-StCDH *problem requires the adversary to commit to* $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \in \mathcal{X}^\ell$ *and then compute* $z_i = (g_{b_i}^{-1} \cdot p_i) \star y_i^{\mathsf{P}}$ $\forall i \in [\ell]$*, as well as* $\hat{z}_i = (g_{b_i}^{-1} \cdot \hat{p}_i) \star y_i^{\mathsf{P}}$ $\forall i \in [\ell]$ *after receiving the challenge password* $\mathsf{pw} = (b_1, ..., b_\ell) \in \{0,1\}^\ell$*. To an effective group action*

$\underline{\mathcal{B}^{\{\text{GA-DDH}_{x_j}(w_i,\cdot,\cdot)\}_{i,j\in\{0,1\}}}(x_0, x_1, w_0, w_1, \mathsf{pw})}$

00 $(p_1, ..., p_\ell, \hat{p}_1, ..., \hat{p}_\ell) \xleftarrow{\$} \mathcal{G}^{2\ell}$
01 $(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) \coloneqq (p_1 \star x, ..., p_\ell \star w_0)$
02 $(\hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}}) \coloneqq (\hat{p}_1 \star x, ..., \hat{p}_\ell \star w_1)$
03 $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1, x_1^{\mathsf{P}}, ...x_\ell^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}})$
04 $(z_1, ..., z_\ell, \hat{z}_1, ..., \hat{z}_\ell) \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pw})$
05 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
06 **if** $\text{GA-DDH}_{x_{b_i}}(w_0, y_i^{\mathsf{P}}, p_i^{-1} \star z_i) = 1 \ \forall i \in [\ell]$
and $\text{GA-DDH}_{x_{b_i}}(w_1, y_i^{\mathsf{P}}, \hat{p}_i^{-1} \star \hat{z}_i) = 1 \ \forall i \in [\ell]$
07 **return** $(1, (\mathsf{pw}, p_1, ..., p_\ell, \hat{p}_1, ..., \hat{p}_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell, \hat{z}_1, ..., \hat{z}_\ell))$
08 **return** $(0, \epsilon)$

$\underline{\{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, y', z')\}_{i\in[\ell],j\in\{0,1\}}}$
09 **return** $\text{GA-DDH}_{x_j}(w_0, y', p_i^{-1} \star z')$

$\underline{\{\text{GA-DDH}_{x_j}(\hat{x}_i^{\mathsf{P}}, y', z')\}_{i\in[\ell],j\in\{0,1\}}}$
10 **return** $\text{GA-DDH}_{x_j}(w_1, y', \hat{p}_i^{-1} \star z')$

Figure 30: Adversary $\mathcal{B}$ for the proof of Lemma 6. Adversary $\mathcal{A}$ has access to oracles $\mathrm{O} = \{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, \cdot, \cdot), \text{GA-DDH}_{x_j}(\hat{x}_i^{\mathsf{P}}, \cdot, \cdot)\}_{i\in[\ell],j\in\{0,1\}}$.

$\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{DPw\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{array}{l} z_i = \text{GA-CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}) \ \forall i \in [\ell] \\ \hat{z}_i = \text{GA-CDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, y_i^{\mathsf{P}}) \ \forall i \in [\ell] \end{array} \middle| \begin{array}{c} (g_0, g_1, p_1, ..., p_\ell, \hat{p}_1, ..., \hat{p}_\ell) \xleftarrow{\$} \mathcal{G}^{2\ell+2} \\ (x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ (x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) \coloneqq (p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x}) \\ (\hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}}) \coloneqq (\hat{p}_1 \star \tilde{x}, ..., \hat{p}_\ell \star \tilde{x}) \\ (y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}}) \\ \mathsf{pw} \coloneqq (b_1, ..., b_\ell) \xleftarrow{\$} \mathcal{PW} \\ (z_1, ..., z_\ell, \hat{z}_1, ..., \hat{z}_\ell) \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pw}) \end{array} \right],$$

where $\mathrm{O} = \{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, \cdot, \cdot), \text{GA-DDH}_{x_j}(\hat{x}_i^{\mathsf{P}}, \cdot, \cdot)\}_{i\in[\ell],j\in\{0,1\}}$.

Lemma 6. *The square-inverse* GA-StCDH (SqInv-GA-StCDH) *implies the double password-based* GA-StCDH (DPw-GA-StCDH)*, more precisely*

$$\mathsf{Adv}^{\mathsf{DPw\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \sqrt{\mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B})} + \frac{1}{|\mathcal{PW}|} \ .$$

Proof. Instead of considering the SqInv-GA-StCDH problem, we will show that the DSim-GA-StCDH problem implies the DPw-GA-StCDH problem and then appy Lemma 1. The proof is similar to the one of Lemma 5. We apply the reset lemma (Lemma 2), where $H = \mathcal{PW}$.

Let $\mathcal{A}$ be an adversary against the DPw-GA-StCDH problem. Consider adversary $\mathcal{B}$ in Figure 30 that takes as input four set elements (x_0, x_1, w_0, w_1) and a password pw. It also has access to decision oracles $\text{GA-DDH}_{x_j}(w_i, \cdot, \cdot)$ with $i, j \in \{0, 1\}$. First, $\mathcal{B}$ generates $(p_1, ...p_\ell, \hat{p}_1, ..., \hat{p}_\ell) \xleftarrow{\$} \mathcal{G}^{2\ell}$ and computes $x_i^{\mathsf{P}} = p_i \star w_0$ and $\hat{x}_i^{\mathsf{P}} = \hat{p}_i \star w_1$ for all $i \in [\ell]$. Then it runs $\mathcal{A}$ on input $(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}})$ and receives a commitment $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}})$. Now $\mathcal{B}$ sends pw to $\mathcal{A}$ and $\mathcal{A}$ will finally output $(z_1, ..., z_\ell, \hat{z}_1, ..., \hat{z}_\ell)$. $\mathcal{B}$ checks if the solution is correct using the decision oracles and if this is the case, it outputs $b = 1$ and $\sigma = (\mathsf{pw}, p_1, ..., p_\ell, \hat{p}_1, ..., \hat{p}_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell, \hat{z}_1, ..., \hat{z}_\ell)$. Otherwise it outputs $(0, \epsilon)$.

If $\mathcal{A}$ issues a query to a decision oracle $\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, y', z')$ for some $i \in [\ell]$ and $j \in \{0, 1\}$, $\mathcal{B}$ queries its own decision oracle $\text{GA-DDH}_{x_j}(w_0, y', p_i^{-1} \star z')$ and forwards the output to $\mathcal{A}$. Analogously, if $\mathcal{A}$ issues a query to a decision oracle $\text{GA-DDH}_{x_j}(\hat{x}_i^{\mathsf{P}}, y', z')$, $\mathcal{B}$ queries $\text{GA-DDH}_{x_j}(w_1, y', \hat{p}_i^{-1} \star z')$.

Let IG be the algorithm that chooses $g_0, g_1, h_0, h_1 \xleftarrow{\$} \mathcal{G}$ and outputs

$$(x_0, x_1, w_0, w_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x}).$$

$\mathcal{C}^{\{\text{GA-DDH}_{x_j}(w_i,\cdot,\cdot)\}_{i,j\in\{0,1\}}}(x_0, x_1, w_0, w_1)$

$(b, \sigma, \sigma') \leftarrow \mathcal{R}_{\mathcal{B}}^{\mathrm{O}}(x_0, x_1, w_0, w_1)$
if $b = 0$
  abort and **return** $\bot$
$(\mathsf{pw}, p_1, ..., p_\ell, \hat{p}_1, ..., \hat{p}_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell, \hat{z}_1, ..., \hat{z}_\ell) \coloneqq \sigma$
$(\mathsf{pw}', p_1, ..., p_\ell, \hat{p}_1, ..., \hat{p}_\ell, y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z'_1, ..., z'_\ell, \hat{z}'_1, ..., \hat{z}'_\ell) \coloneqq \sigma'$
$(b_1, ..., b_\ell) \coloneqq \mathsf{pw}$
$(b'_1, ..., b'_\ell) \coloneqq \mathsf{pw}'$
Find i such that $b_i \neq b'_i$
return $(y_i^{\mathsf{P}}, p_i^{-1} \star z_i, \hat{p}_i^{-1} \star \hat{z}_i, p_i^{-1} \star z'_i, \hat{p}_i^{-1} \star \hat{z}'_i)$

Figure 31: Adversary $\mathcal{C}$ against DSim-GA-StCDH for the proof of Lemma 6. $\mathcal{R}_{\mathcal{B}}$ has access to oracles $\mathrm{O} = \{\text{GA-DDH}_{x_j}(w_i, \cdot, \cdot)\}_{i,j\in\{0,1\}}$.

Let acc be defined as in Lemma 2, thus

$$\text{acc} \geq \mathsf{Adv}^{\mathsf{DPw\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \ .$$

Let $\mathcal{R}_{\mathcal{B}}$ be the forking algorithm associated to $\mathcal{B}$ as in Lemma 2 with access to the same decision oracles as $\mathcal{B}$. Then we construct adversary $\mathcal{C}$ against DSim-GA-StCDH as in Figure 31. $\mathcal{C}$ runs the reset algorithm to obtain a bit b^* as well as the two side outputs σ, σ'. $\mathcal{C}$ looks for the first index i where pw and pw′ differ and outputs the solution $(y, y_0, y_1, y_2, y_3) = (y_i^{\mathsf{P}}, p_i^{-1} \star z_i, \hat{p}_i^{-1} \star \hat{z}_i, p_i^{-1} \star z'_i, \hat{p}_i^{-1} \star \hat{z}'_i)$.

To see that the latter solves the DSim-GA-StCDH problem, note that $p_i^{-1} \star x_i^{\mathsf{P}} = w_0$. This implies,

$$\mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, y_i^{\mathsf{P}}) = p_i^{-1} \star \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}) = p_i^{-1} \star z_i.$$

And similarly $\mathsf{GA\text{-}CDH}_{x_{b_i}}(w_1, y_i^{\mathsf{P}}) = \hat{p}_i^{-1} \star \hat{z}_i$.

Applying Lemmata 2 and 1, we get the bound stated in Lemma 6. □

Theorem 9 (Perfect Forward Secrecy of X-GA-PAKE$_\ell$)**.** *For any adversary $\mathcal{A}$ against* X-GA-PAKE$_\ell$ *that issues at most q_e execute queries and q_s send queries and where* H *is modeled as a random oracle, there exist adversary $\mathcal{B}_1$ against* GA-StCDH*, and adversaries $\mathcal{B}_2$, $\mathcal{B}_3$ against* SqInv-GA-StCDH *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pfs}}_{\mathsf{X\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2) \\ & + q_s \cdot \sqrt{\mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_3)} + \frac{2q_s}{|\mathcal{PW}|} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^{2\ell}} \ .\end{aligned}$$

Proof. The proof follows the one of Theorem 1 very closely and we make the same adaptions as for the proof of Theorem 8. In particular, we also keep games G_0-G_6 from Figures 5, 8 and 9 with the same changes as described in the proof of Theorem 8, until we get

$$\Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{pfs}}] \ ,$$

where it remains to bound $\mathbf{bad}_{\text{pfs}}$. For this purpose, we construct an adversary $\mathcal{B}_3$ against DPw-GA-StCDH in Figure 32 and show that

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{pfs}}] \leq q_s \cdot \mathsf{Adv}^{\mathsf{DPw\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_3) \ .$$

$\mathcal{B}_3^{\{\text{GA-DDH}_{x_b}(y,\cdot,\cdot)\}_{b\in\{0,1\},y\in\{x_i^{\mathsf{P}},\hat{x}_i^{\mathsf{P}}\},i\in[\ell]}}(x_0, x_1, x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \hat{x}_1^{\mathsf{P}}, ..., \hat{x}_\ell^{\mathsf{P}})$

$(\mathcal{C}, T, T_{\mathsf{s}}) := (\emptyset, \emptyset, \emptyset)$
$\tau^* \xleftarrow{\$} [q_{\mathsf{s}}]$
$\mathsf{tr}^* := \bot$
$\mathrm{cnt} := 0$
$\beta \xleftarrow{\$} \{0,1\}$
$\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
Stop.

SendInit$(\mathsf{U}, t, \mathsf{S})$
$\mathrm{cnt} := \mathrm{cnt} + 1$
if $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
$u := (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{u} := (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{U}} := (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) := (u_1 \star x_1^{\mathsf{P}}, ..., u_\ell \star x_\ell^{\mathsf{P}})$
$\hat{x}^{\mathsf{U}} := (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) := (\hat{u}_1 \star \hat{x}_1^{\mathsf{P}}, ..., \hat{u}_\ell \star \hat{x}_\ell^{\mathsf{P}})$
$\pi_{\mathsf{U}}^t := ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
return $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$

Corrupt(U, S)
if $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
$\mathcal{C} := \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
if $\mathsf{tr}^* \neq (\mathsf{U}, \mathsf{S}, \cdot, \cdot)$
  $\mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
return $\mathsf{pw}_{\mathsf{US}}$

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z)$
if $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$
  return K
if $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}) \in T_{\mathsf{s}}$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  if $T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{U}, (u, \hat{u}), K)$
    if $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{S}}, u_i^{-1} \star z_{i,1}) = 1\ \forall i \in [\ell]$
    and $\text{GA-DDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, x_i^{\mathsf{S}}, \hat{u}_i^{-1} \star z_{i,2}) = 1\ \forall i \in [\ell]$
    and $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} := \mathsf{pw}_{\mathsf{US}}$
      if $\mathsf{tr}^* \neq (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ **abort**
      Output $(u_1^{-1} \star z_{1,1}, ..., u_\ell^{-1} \star z_{\ell,1}, \hat{u}_1^{-1} \star z_{1,2}, ..., \hat{u}_\ell^{-1} \star z_{\ell,2})$
  if $T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] = (\mathsf{S}, (s, \hat{s}), K)$
    if $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{U}}, s_i^{-1} \star z_{i,1}) = 1\ \forall i \in [\ell]$
    and $\text{GA-DDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, x_i^{\mathsf{U}}, \hat{s}_i^{-1} \star z_{i,3}) = 1\ \forall i \in [\ell]$
    and $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **and** $\mathsf{pw} := \mathsf{pw}_{\mathsf{US}}$
      if $\mathsf{tr}^* \neq (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ **abort**
      Output $(s_1^{-1} \star z_{1,1}, ..., s_\ell^{-1} \star z_{\ell,1}, \hat{s}_1^{-1} \star z_{1,3}, ..., \hat{s}_\ell^{-1} \star z_{\ell,3})$
if $\exists (u, \hat{u})$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, (u, \hat{u})) \in T$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  if $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{S}}, u_i^{-1} \star z_{i,1}) = 1\ \forall i \in [\ell]$
  and $\text{GA-DDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, x_i^{\mathsf{S}}, \hat{u}_i^{-1} \star z_{i,2}) = 1\ \forall i \in [\ell]$
    return $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, (u, \hat{u})]$
else if $\exists (s, \hat{s})$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, (s, \hat{s})) \in T$
  $(b_1, ..., b_\ell) := \mathsf{pw}$
  if $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{U}}, s_i^{-1} \star z_{i,1}) = 1\ \forall i \in [\ell]$
  and $\text{GA-DDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, x_i^{\mathsf{U}}, \hat{s}_i^{-1} \star z_{i,3}) = 1\ \forall i \in [\ell]$
    return $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, (s, \hat{s})]$
$T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z] \xleftarrow{\$} \mathcal{K}$
return $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z]$

SendResp$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$
$\mathrm{cnt} := \mathrm{cnt} + 1$
if $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
$s := (s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$\hat{s} := (\hat{s}_1, ..., \hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
$x^{\mathsf{S}} := (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) := (s_1 \star x_1^{\mathsf{P}}, ..., s_\ell \star x_\ell^{\mathsf{P}})$
$\hat{x}^{\mathsf{S}} := (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) := (\hat{s}_1 \star \hat{x}_1^{\mathsf{P}}, ..., \hat{s}_\ell \star \hat{x}_\ell^{\mathsf{P}})$
if $\exists P \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
  return $\bot$
if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{true}$
  if $\mathrm{cnt} = \tau^*$
    $\mathsf{tr}^* := (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
    Output $y^{\mathsf{P}} := x^{\mathsf{U}}$ to receive $\mathsf{pw}_{\mathsf{US}}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] := (\mathsf{S}, (s, \hat{s}), K)$
else
  $\pi_{\mathsf{S}}^t.\mathsf{fr} := \mathbf{false}$
  $(b_1, ..., b_\ell) := \mathsf{pw}_{\mathsf{US}}$
  if $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$
  and $\forall i \in [\ell] : \text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{U}}, s_i^{-1} \star z_{i,1}) = 1$
  and $\forall i \in [\ell] : \text{GA-DDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, x_i^{\mathsf{U}}, \hat{s}_i^{-1} \star z_{i,3}) = 1$
    $K := T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z]$
  else
    $K \xleftarrow{\$} \mathcal{K}$
    $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, (s, \hat{s})] := K$
$\pi_{\mathsf{S}}^t := ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
return $(\mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

SendTermInit$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
$\mathrm{cnt} := \mathrm{cnt} + 1$
if $\pi_{\mathsf{U}}^t \neq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
  return $\bot$
if $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
  return $\bot$
if $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
and $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
  $\pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{true}$
  $(\mathsf{S}, (s, \hat{s}), K) := T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}]$
else if $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
  $\pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{true}$
  if $\mathrm{cnt} = \tau^*$
    $\mathsf{tr}^* := (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
    Output $y^{\mathsf{P}} := x^{\mathsf{S}}$ to receive $\mathsf{pw}_{\mathsf{US}}$
  $K \xleftarrow{\$} \mathcal{K}$
  $T_{\mathsf{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}] := (\mathsf{U}, (u, \hat{u}), K)$
else
  $\pi_{\mathsf{U}}^t.\mathsf{fr} := \mathbf{false}$
  $(b_1, ..., b_\ell) := \mathsf{pw}_{\mathsf{US}}$
  if $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z) \in T$
  and $\forall i \in [\ell] : \text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{S}}, u_i^{-1} \star z_{i,1}) = 1$
  and $\forall i \in [\ell] : \text{GA-DDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{P}}, x_i^{\mathsf{S}}, \hat{u}_i^{-1} \star z_{i,2}) = 1$
    $K := T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z]$
  else
    $K \xleftarrow{\$} \mathcal{K}$
    $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, (u, \hat{u})] := K$
$\pi_{\mathsf{U}}^t := ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
return true

Figure 32: Adversary $\mathcal{B}_3$ against DPw-GA-StCDH for the proof of Theorem 9. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{Execute}, \text{SendInit}, \text{SendResp}, \text{SendTermInit}, \text{Reveal}, \text{Corrupt}, \text{Test}, \mathsf{H}\}$. Oracles Execute, Reveal and Test are defined as in Figure 9. Lines written in blue show how $\mathcal{B}_3$ simulates the game.

As in the proof of Theorem 8, we cannot achieve a tight bound for $\mathbf{bad}_{\text{pfs}}$. The adversary against DPw-GA-StCDH must commit to ℓ set elements before it receives the challenge password. Thus, we apply the same guessing and simulation strategy. Due to the similarities to Theorem 8, we will only briefly describe $\mathcal{B}_3$.

It inputs set elements $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$ and $x_i^{\mathsf{P}} = (p_i \star \tilde{x})$, $\hat{x}_i^{\mathsf{P}} = (\hat{p}_i \star \tilde{x})$ for $g_0, g_1, p_i, \hat{p}_i \xleftarrow{\$} \mathcal{G}$ and $i \in [\ell]$. It embeds the elements x_i^{P}, $\hat{x}_i^{\mathsf{P}}$ in all instances queried to SendInit and SendResp.

If the password of an instance is not corrupted when one of oracles SendResp or SendTermInit is queried, $\mathcal{B}_3$ checks whether this is the τ^*-th query and in this case outputs x^{U} in SendResp or x^{S} in SendTermInit as the commitment y^{P} to receive the challenge password $\mathsf{pw}_{\mathsf{US}}$. This is the password $\mathcal{B}_3$ will output when the adversary corrupts this pair of user and server. Instances that are not fresh will be simulated with the decision oracles.

Finally, we look at random oracle queries for all fresh instances contained in T_{s}. $\mathcal{B}_3$ checks if the provided z is valid using the decision oracles. Then it checks for event $\mathbf{bad}_{\text{pfs}}$, i.e., if the password was corrupted and matches the one in the query. If the query now additionally contains the target trace, we can solve the DPw-GA-StCDH problem. If the trace is not the target trace, $\mathcal{B}_3$ aborts. This concludes the analysis of $\mathbf{bad}_{\text{pfs}}$. Collecting the bounds and applying Lemma 6 yields the bound in Theorem 9. □

F.3 Perfect Forward Secrecy of Com-GA-PAKE$_\ell$

In order to prove forward secrecy of Com-GA-PAKE$_\ell$, we need the following interactive problem which is non-tightly implied by the GA-GapCDH problem.

Definition 22 (Interactive Password-based GA-StCDH (IPw-GA-StCDH))**.** *On input $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$, the* IPw-GA-StCDH *problem requires the adversary to commit to $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \in \mathcal{X}^\ell$. Then it receives $(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) = (p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x})$ and the challenge password $\mathsf{pw} = (b_1, ..., b_\ell) \in \{0,1\}^\ell$ and must compute $z_i = (g_{b_i}^{-1} \cdot p_i) \star y_i^{\mathsf{P}}$ $\forall i \in [\ell]$. To an effective group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$, we associate the advantage function of $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{IPw\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[z_i = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}})\ \forall i \in [\ell] \;\middle|\; \begin{array}{c} (g_0, g_1, p_1, ..., p_\ell) \xleftarrow{\$} \mathcal{G}^{\ell+2} \\ (x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ (y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \leftarrow \mathcal{A}^{\mathrm{O}_1}(x_0, x_1) \\ (x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) := (p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x}) \\ \mathsf{pw} := (b_1, ..., b_\ell) \xleftarrow{\$} \mathcal{PW} \\ (z_1, ..., z_\ell) \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \mathsf{pw}) \end{array} \right],$$

where $\mathrm{O}_1 = \{\text{GA-DDH}_{x_j}(\tilde{x}, \cdot, \cdot)\}_{j \in \{0,1\}}$ *and* $\mathrm{O}_2 = \{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, \cdot, \cdot)\}_{i \in [\ell], j \in \{0,1\}}$.

Lemma 7. *The group action gap computational Diffie-Hellman problem* (GA-GapCDH) *implies the interactive password-based* GA-StCDH (IPw-GA-StCDH)*, more precisely*

$$\mathsf{Adv}^{\mathsf{IPw\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B})} + \frac{1}{|\mathcal{PW}|}\ .$$

$\underline{\mathcal{B}^{\text{GA-DDH}_*}(x_0, x_1, (x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \mathsf{pw}))}$
00 $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}) \leftarrow \mathcal{A}^{\mathrm{O}_1}(x_0, x_1)$
01 $(z_1, ..., z_\ell) \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \mathsf{pw})$
02 $(b_1, ..., b_\ell) := \mathsf{pw}$
03 **if** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, y_i^{\mathsf{P}}, z_i) = 1 \ \forall i \in [\ell]$
04 **return** $(1, (y_1^{\mathsf{P}}, ...y_\ell^{\mathsf{P}}, z_1, ..., z_\ell))$
05 **return** $(0, \epsilon)$

$\underline{\mathcal{C}^{\text{GA-DDH}_*}(x_0, x_1)}$
06 Pick random coins ρ for $\mathcal{B}$
07 $(p_1, ..., p_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
08 $x^{\mathsf{P}} := (x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}) := (p_1 \star \tilde{x}, ..., p_\ell \star \tilde{x})$
09 $\mathsf{pw} := (b_1, ..., b_\ell) \xleftarrow{\$} \mathcal{PW}$
10 $(b, \sigma) \leftarrow \mathcal{B}^{\text{GA-DDH}_*}(x_0, x_1^t, (x^{\mathsf{P}}, \mathsf{pw}); \rho)$
11 **if** $b = 0$ **return** $\perp$
12 $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell) := \sigma$
13 $(\alpha_1, ..., \alpha_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
14 $x^{\mathsf{P}'} := (x_1^{\mathsf{P}'}, ..., x_\ell^{\mathsf{P}'}) := (\alpha_1 \star z_1^t, ..., \alpha_\ell \star z_\ell^t)$
15 $\mathsf{pw}' := (b_1', ..., b_\ell') \xleftarrow{\$} \mathcal{PW}$
16 $(b', \sigma') \leftarrow \mathcal{B}^{\text{GA-DDH}_*}(x_0, x_1^t, (x^{\mathsf{P}'}, \mathsf{pw}'); \rho)$
17 **if** $b = 0$ **return** $\perp$
18 $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1', ..., z_\ell') := \sigma'$
19 Find i such that $b_i \neq b_i'$
20 **if** $b_i = 0$ **and** $b_i' = 1$
21 **return** $\alpha_i^{-1} \cdot p_i \star z_i'$
22 **else**
23 **return** $(\alpha_i^{-1} \cdot p_i \star z_i')^t$

Figure 33: Adversaries $\mathcal{B}$ and $\mathcal{C}$ against GA-GapCDH for the proof of Lemma 7. Adversary $\mathcal{A}$ has access to decision oracles $\mathrm{O}_1 = \{\text{GA-DDH}_{x_j}(\tilde{x}, \cdot, \cdot)\}_{j \in \{0,1\}}$ and $\mathrm{O}_2 = \{\text{GA-DDH}_{x_j}(x_i^{\mathsf{P}}, \cdot, \cdot)\}_{i \in [\ell], j \in \{0,1\}}$, which $\mathcal{B}$ simulates using the gap oracle GA-DDH_*.

Instead of proving that the ISim-GA-StCDH problem implies the password-based version IPw-GA-StCDH, we directly use the GA-GapCDH problem to achieve a better bound. The proof uses techniques that are also used in the proof of Lemma 4.

Proof. We use the reset lemma (see Lemma 2) with $H = \mathcal{X}^\ell \times \mathcal{PW}$. Let $\mathcal{A}$ be an adversary against IPw-GA-StCDH. Consider adversary $\mathcal{B}$ against GA-GapCDH in Figure 33 that takes input (x_0, x_1) as well as $(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \mathsf{pw})$. It also has access to a gap oracle GA-DDH_*. First, $\mathcal{B}$ runs $\mathcal{A}$ on (x_0, x_1) to receive a commitment $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}})$. Now $\mathcal{B}$ sends $(x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \mathsf{pw})$ to $\mathcal{A}$ and $\mathcal{A}$ will finally output $(z_1, ..., z_\ell)$. $\mathcal{B}$ checks if the solution is correct using the decision oracle and if this is the case, it outputs $b = 1$ and $\sigma = (y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell)$ as side output. Otherwise it outputs $(0, \epsilon)$. As $\mathcal{B}$ has access to a full gap oracle, it can forward all queries of $\mathcal{A}$.

Let IG be the algorithm that chooses $g_0, g_1 \xleftarrow{\$} \mathcal{G}$ and outputs $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$. Let acc be defined as in Lemma 2, thus

$$\text{acc} \geq \mathsf{Adv}_{\mathsf{EGAT}}^{\text{IPw-GA-StCDH}}(\mathcal{A}) \ .$$

Let $\mathcal{R}_{\mathcal{B}}$ be the reset algorithm associated to $\mathcal{B}$ as in Lemma 2 with access to the same decision oracles as $\mathcal{B}$.

We construct an adversary $\mathcal{C}$ against GA-GapCDH (Figure 33), but instead of running the reset algorithm, $\mathcal{C}$ will simulate $\mathcal{R}_{\mathcal{B}}$ running $\mathcal{B}$ directly.

$\mathcal{C}$ inputs (x_0, x_1) and has access to a gap oracle. First, it chooses random coins ρ for $\mathcal{B}$. It also samples a random element from H by first picking $p_i \xleftarrow{\$} \mathcal{G}$ for $i \in [\ell]$ and a password pw and then computing $x_i^{\mathsf{P}} = p_i \star \tilde{x}$. Then it runs $\mathcal{B}$ on $(x_0, x_1^t, (x_1^{\mathsf{P}}, ..., x_\ell^{\mathsf{P}}, \mathsf{pw})$ using random coins ρ. Note that we use the twist of x_1. $\mathcal{B}$ outputs a bit b and side output σ. If $\mathcal{B}$ was successful, i.e., $b = 1$, then $\mathcal{C}$ parses σ as $(y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z_1, ..., z_\ell)$. Otherwise it aborts. Now it runs $\mathcal{B}$ a second time, this time on input $(x_0, x_1^t, (x_1^{\mathsf{P}'}, ..., x_\ell^{\mathsf{P}'}, \mathsf{pw}')$, where

$x_i^{\mathsf{P}\prime} = \alpha_i \star z_i^t$ for $\alpha_i \xleftarrow{\$} \mathcal{G}$ and $i \in [\ell]$ and pw' is a fresh random password, using the same random coins ρ. Note that all x_i^{P} are also uniformly distributed over $\mathcal{X}$. If $\mathcal{B}$ is successful again, it outputs $(1, (y_1^{\mathsf{P}}, ..., y_\ell^{\mathsf{P}}, z'_1, ..., z'_\ell))$, where the first ℓ set elements are the same as in the first run of $\mathcal{B}$ since we run $\mathcal{B}$ on the same random coins. Now $\mathcal{C}$ can solve GA-GapCDH as follows: Let $y_i^{\mathsf{P}} = h_i \star \tilde{x}$ for some $h_i \in \mathcal{G}$. Then we have

$$\alpha_i \star z_i^t = \begin{cases} \alpha_i \cdot g_0 \cdot p_i^{-1} \cdot h_i^{-1} \star \tilde{x} & \text{if } b_i = 0 \\ \alpha_i \cdot g_1^{-1} \cdot p_i^{-1} \cdot h_i^{-1} \star \tilde{x} & \text{if } b_i = 1 \end{cases}$$

and for z'_i it holds that

$$z'_i = \begin{cases} g_0^{-1} \cdot \alpha_i \cdot g_0 \cdot p_i^{-1} \star \tilde{x} & \text{if } b_i = 0, b'_i = 0 \\ g_1 \cdot \alpha_i \cdot g_0 \cdot p_i^{-1} \star \tilde{x} & \text{if } b_i = 0, b'_i = 1 \\ g_0^{-1} \cdot \alpha_i \cdot g_1^{-1} \cdot p_i^{-1} \star \tilde{x} & \text{if } b_i = 1, b'_i = 0 \\ g_1 \cdot \alpha_i \cdot g_1^{-1} \cdot p_i^{-1} \star \tilde{x} & \text{if } b_i = 1, b'_i = 1 \end{cases}$$

where h_i cancels out in all cases. If $\mathsf{pw} \neq \mathsf{pw}'$, then they must differ in at least one bit. Let i be the first index such that $b_i \neq b'_i$. Using the knowledge of p_i and α_i, $\mathcal{C}$ outputs $\alpha^{-1} \cdot p_i \star z'_i = \mathsf{GA\text{-}CDH}(x_0, x_1)$ in case $b_i = 0$ and $b'_i = 1$. Otherwise if $b_i = 1$ and $b'_i = 0$, it outputs $(\alpha_i^{-1} p_i \star z'_i)^t = \mathsf{GA\text{-}CDH}(x_0, x_1)$. □

Theorem 10 (Perfect Forward Secrecy of $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$)**.** *For any adversary $\mathcal{A}$ against $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ that issues at most q_e execute queries, q_s send queries and at most q_{G} and q_{H} queries to random oracles G and H, there exist adversary $\mathcal{B}_1$ against GA-StCDH and adversaries $\mathcal{B}_2$, $\mathcal{B}_3$ against GA-GapCDH such that*

$$\mathsf{Adv}^{\mathsf{pfs}}_{\mathsf{Com\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + q_s \ell \cdot \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)} + q_s \cdot \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_3)} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^\ell} + \frac{q_{\mathsf{G}} q_s}{|\mathcal{G}|^\ell} + \frac{2 \cdot (q_{\mathsf{G}} + q_s + q_e)^2}{2^\lambda} + \frac{q_s}{|\mathcal{PW}|} \ ,$$

where λ is the output length of G in bits.

Proof. The proof follows the one of Theorem 2 very closely and we make the same adaptions as for the proof of Theorem 8. In particular, we also keep games G_0-G_9 from Figures 19, 22 and 24 with the same changes as described in the proof of Theorem 8, until we get

$$\Pr[\mathsf{G}_8 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathrm{guess}}] + \Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathrm{pfs}}] \ ,$$

where it remains to bound $\mathbf{bad}_{\mathrm{pfs}}$. For this purpose, we construct an adversary $\mathcal{B}_3$ against IPw-GA-StCDH in Figures 34 and 35 and show that

$$\Pr[\mathsf{G}_9 \Rightarrow \mathbf{bad}_{\mathrm{pfs}}] \leq q_s \cdot \mathsf{Adv}^{\mathsf{IPw\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_3) \ .$$

We apply the same guessing and simulation strategy. Due to the similarities to Theorem 8 and adversary $\mathcal{B}_2$ in Theorem 2, we will only briefly describe $\mathcal{B}_3$. It inputs set elements $(x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x})$.

Figure 34: Adversary $\mathcal{B}_3$ against $\mathsf{IPw\text{-}GA\text{-}StCDH}$ for the proof of Theorem 10. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTermInit}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{G}, \mathsf{H}\}$. Oracles $\textsc{Execute}$, $\textsc{Reveal}$, $\textsc{Test}$ and G are defined as in Figure 24. Oracles $\textsc{SendTermInit}$ and $\textsc{SendTermResp}$ are defined in Figure 35. Lines written in blue show how $\mathcal{B}_3$ simulates the game.

In $\textsc{SendResp}$, we check if this is the τ^*-th query and the commitment sent by $\mathcal{A}$ was output by G before, then $\mathcal{B}_3$ looks in the list T_G to find the corresponding input x^S and outputs x^S as commitment y^P to receive the $\mathsf{IPw\text{-}GA\text{-}StCDH}$ challenge x^P and pw_US. Queries to $\textsc{SendTermInit}$ are simulated similarly. $\mathcal{B}_2$ computes x^S and if this is the τ^*-th query, it outputs x^U as commitment y^P to receive x^P and pw_US. Instances that are not fresh will be simulated with the decision oracles.

SendTermInit$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}})$

00 $\mathrm{cnt} \coloneqq \mathrm{cnt} + 1$
01 **if** $\pi_{\mathsf{S}}^t \neq (\bot, (\mathsf{U}, \mathsf{S}, \bot, \bot, \mathsf{com}), \bot, \bot)$
02 **return** $\bot$
03 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
04 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$
05 **if** $\mathrm{cnt} = \tau^*$
06 Output $y^{\mathsf{P}} \coloneqq x^{\mathsf{U}}$ to receive challenge $x^{\mathsf{P}}, \mathsf{pw}_{\mathsf{US}}$
07 // From now on $\mathcal{B}_3$ also has access to $\text{GA-DDH}_{x_0}(x_i^{\mathsf{P}}, \cdot, \cdot), \text{GA-DDH}_{x_1}(x_i^{\mathsf{P}}, \cdot, \cdot)\ \forall i \in [\ell]$
08 $x^{\mathsf{S}} \coloneqq x^{\mathsf{P}}$
09 $(s_1, ..., s_\ell) \coloneqq \bot$
10 $\mathsf{tr}^* = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
11 **if** $T_{\mathsf{G}}[x^{\mathsf{S}}] \neq \bot$
12 **return** $\bot$
13 Replace $\diamond$ in $T_{\mathsf{G}}[\diamond] \coloneqq \mathsf{com}$ with x^{S}
14 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
15 **return** $\bot$
16 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
17 $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
18 $K \xleftarrow{\$} \mathcal{K}$
19 $T_{\mathrm{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{S}, (s_1, ..., s_\ell), K)$
20 **else**
21 $\pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
22 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
23 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}) = \mathsf{tr}^*$ **and** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{U}}, z_i) = 1\ \forall i \in [\ell]$
24 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
25 **else if** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** $\text{GA-DDH}_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{U}}, s_i^{-1} \star z_i) = 1\ \forall i \in [\ell]$
26 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
27 **else**
28 $K \xleftarrow{\$} \mathcal{K}$
29 $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, (s_1, ..., s_\ell)] \coloneqq K$
30 $\pi_{\mathsf{S}}^t \coloneqq ((s_1, ..., s_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
31 **return** $(\mathsf{S}, x^{\mathsf{S}})$

SendTermResp$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}})$

32 $\mathrm{cnt} \coloneqq \mathrm{cnt} + 1$
33 **if** $\pi_{\mathsf{U}}^t \neq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \bot, \mathsf{com}), \bot, \bot)$
34 **return** $\bot$
35 **if** $\mathsf{G}(x^{\mathsf{S}}) \neq \mathsf{com}$
36 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), \bot, \mathbf{false})$
37 **return** $\bot$
38 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s. t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
39 **return** $\bot$
40 **if** $\pi_{\mathsf{U}}^t.\mathsf{tr} = \mathsf{tr}^*$
41 $\mathsf{tr}^* \coloneqq (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$
42 **if** $\exists t'$ s. t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com})$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$
43 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
44 $(\mathsf{S}, (s_1, ..., s_\ell), K) \coloneqq T_{\mathrm{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}]$
45 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
46 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$
47 $K \xleftarrow{\$} \mathcal{K}$
48 $T_{\mathrm{s}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}] \coloneqq (\mathsf{U}, (u_1, ..., u_\ell), K)$
49 **else**
50 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$
51 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
52 **if** $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}) = \mathsf{tr}^*$ **and** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** $\text{GA-DDH}_{x_{b_i}}(x_i^{\mathsf{P}}, x_i^{\mathsf{S}}, z_i) = 1\ \forall i \in [\ell]$
53 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
54 **else if** $\exists z$ s. t. $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z) \in T_{\mathsf{H}}$ **and** $\text{GA-DDH}_{x_{b_i}}(\tilde{x}, x_i^{\mathsf{S}}, u_i^{-1} \star z_i) = 1\ \forall i \in [\ell]$
55 $K \coloneqq T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, z]$
56 **else**
57 $K \xleftarrow{\$} \mathcal{K}$
58 $T_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}, \mathsf{pw}_{\mathsf{US}}, (u_1, ..., u_\ell)] \coloneqq K$
59 $\pi_{\mathsf{U}}^t \coloneqq ((u_1, ..., u_\ell), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, x^{\mathsf{S}}, \mathsf{com}), K, \mathbf{true})$
60 **return true**

Figure 35: Oracles SendTermInit and SendTermResp for adversary $\mathcal{B}_3$ in Figure 34.

Finally, we look at random oracle queries for all fresh instances contained in T_{s}. $\mathcal{B}_3$ checks if the provided z is valid using the decision oracles. Then it checks for event $\mathbf{bad}_{\mathrm{pfs}}$, i.e., if the password was corrupted and it matches the one in the query. If the query now additionally contains the target trace, we can solve the IPw-GA-StCDH problem. If the trace is not the target trace, $\mathcal{B}_3$ aborts. This concludes the analysis of $\mathbf{bad}_{\mathrm{pfs}}$ and applying Lemma 7 yields the bound in Theorem 10. □