J Cryptol (2022)35:26
https://doi.org/10.1007/s00145-022-09438-y

Journal of
CRYPTOLOGY

*Research Article*

# Signed (Group) Diffie–Hellman Key Exchange with Tight Security

Jiaxin Pan · Chen Qian · Magnus Ringerud
Department of Mathematical Sciences, NTNU Norwegian University of Science and Technology,
Trondheim, Norway
jiaxin.pan@ntnu.no
chen.qian@ntnu.no
magnus.ringerud@ntnu.no

Communicated by Marc Fischlin

Received 4 November 2021 / Revised 9 August 2022 / Accepted 10 August 2022
Online publication 27 September 2022

**Abstract.** We propose the *first* tight security proof for the ordinary two-message signed Diffie–Hellman key exchange protocol in the random oracle model. Our proof is based on the strong computational Diffie–Hellman assumption and the multiuser security of a digital signature scheme. With our security proof, the signed DH protocol can be deployed with optimal parameters, independent of the number of users or sessions, without the need to compensate any security loss. We abstract our approach with a new notion called verifiable key exchange. In contrast to a known tight three-message variant of the signed Diffie–Hellman protocol (Gjøsteen and Jager, in: Shacham, Boldyreva (eds) CRYPTO 2018, Part II. LNCS, Springer, Heidelberg, 2018), we do not require any modification to the original protocol, and our tightness result is proven in the "Single-Bit-Guess" model which we know can be tightly composed with symmetric cryptographic primitives to establish a secure channel. Finally, we extend our approach to the group setting and construct the *first* tightly secure group authenticated key exchange protocol.

**Keywords.** Authenticated key exchange, Group key exchange, Signed Diffie–Hellman, Tight security.

## 1. Introduction

Authenticated key exchange (AKE) protocols are protocols where two users can securely share a session key in the presence of active adversaries. Beyond passively observing, adversaries against an AKE protocol can modify messages and adaptively corrupt users' long-term keys or the established session key between users. Hence, it is very challenging to construct a secure AKE protocol.

The signed Diffie–Hellman (DH) key exchange protocol is a classical AKE protocol. It is a two-message (namely, two message-moves or one-round) protocol and can be viewed as a generic method to transform a passively secure Diffie–Hellman key exchange

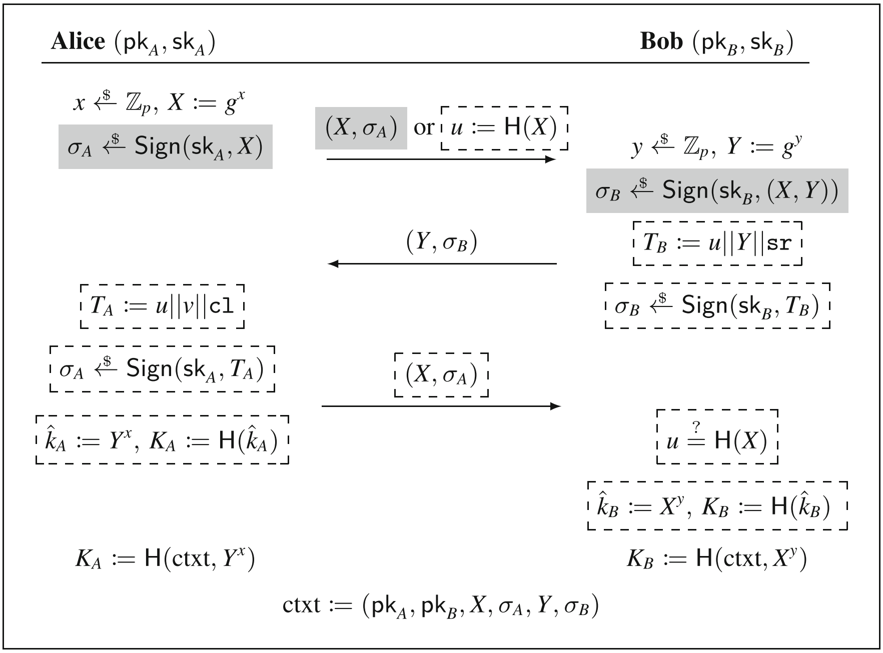
**Alice** $(\mathsf{pk}_A, \mathsf{sk}_A)$
**Bob** $(\mathsf{pk}_B, \mathsf{sk}_B)$
$x \xleftarrow{\$} \mathbb{Z}_p,\ X := g^x$
$\sigma_A \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_A, X)$
$(X, \sigma_A)$ or $u := \mathsf{H}(X)$
$y \xleftarrow{\$} \mathbb{Z}_p,\ Y := g^y$
$\sigma_B \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_B, (X, Y))$
$(Y, \sigma_B)$
$T_B := u||Y||\texttt{sr}$
$T_A := u||v||\texttt{cl}$
$\sigma_B \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_B, T_B)$
$\sigma_A \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_A, T_A)$
$(X, \sigma_A)$
$\hat{k}_A := Y^x,\ K_A := \mathsf{H}(\hat{k}_A)$
$u \overset{?}{=} \mathsf{H}(X)$
$\hat{k}_B := X^y,\ K_B := \mathsf{H}(\hat{k}_B)$
$K_A := \mathsf{H}(\text{ctxt}, Y^x)$
$K_B := \mathsf{H}(\text{ctxt}, X^y)$
$\text{ctxt} := (\mathsf{pk}_A, \mathsf{pk}_B, X, \sigma_A, Y, \sigma_B)$

**Fig. 1.** Our signed Diffie–Hellman key exchange protocol and the tight variant of Gjøsteen and Jager [24]. The functions H and H are hash functions. Operations marked with a gray box are for our signed DH protocol, and dashed boxes are for Gjøsteen and Jager's. Operations without a box are performed by both protocols. All signatures are verified upon arrival with the corresponding messages, and the protocol aborts if any verification fails .

protocol [19] into a secure AKE protocol using digital signatures. Figure 1 visualizes the protocol. The origin of signed DH is unclear to us, but its idea has been used in and serves as a solid foundation for many well-known AKE protocols, including the Station-to-Station protocol [20], IKE protocol [26], the one in TLS 1.3 [42] and many others [7,24,29,30,33].

Tight Security. Security of a cryptographic scheme is usually proven by constructing a reduction. Asymptotically, a reduction reduces any efficient adversary $\mathcal{A}$ against the scheme into an adversary $\mathcal{R}$ against the underlying computational problem. Concretely, a reduction provides a security bound for the scheme, $\varepsilon_{\mathcal{A}} \leq \ell \cdot \varepsilon_{\mathcal{R}}$, where $\varepsilon_{\mathcal{A}}$ is the success probability of $\mathcal{A}$ and $\varepsilon_{\mathcal{R}}$ is that of $\mathcal{R}$. We say a reduction is *tight* if $\ell$ is a small constant and the running time of $\mathcal{A}$ is approximately the same as that of $\mathcal{R}$. For the same scheme, it is more desirable to have a tight security proof than a non-tight one, since a tight security proof enables implementations without the need to compensate a security loss with increased parameters.

Multi-Challenge Security for AKE. An adversary against an AKE protocol has full control of the communication channel and, additionally, it can adaptively corrupt users' long-term keys and reveal session keys. The goal of an adversary is to distinguish between a (non-revealed) session key and a random bit-string of the same length, which is captured by the Test query. We follow the Bellare-Rogaway (BR) model [5] to capture these capabilities, but formalize it with the game-based style of [28]. Instead of weak perfect forward secrecy, our model captures the (full) perfect forward secrecy.

Unlike the BR model, our model captures multi-challenge security, where an adversary can make $T$ many Test queries which are answered with a single random bit. This is a standard and well-established multi-challenge notion, and [28] called it "Single-Bit-Guess" (SBG) security. Another multi-challenge notion is the "Multi-Bit-Guess" (MBG) security where each Test query is answered with a different random bit. Although several tightly secure AKE protocols [2,24,36,46] are proven in the MBG model, we stress that the SBG model is well-established and allows tight composition of the AKE with symmetric cryptographic primitives, which is not the case for the nonstandard MBG model. Thus, the SBG multi-challenge model is more desirable than the MBG model. More details about this have been provided by Jager et al.[28, Introduction] and Cohn-Gordon et al.[14, Section 3].

The Non-Tight Security of Signed DH. Many existing security proofs of signed DH-like protocols [7,29,30] lose a quadratic factor, $O(\mu^2 S^2)$, where $\mu$ and $S$ are the maximum numbers of users and sessions. In the SBG model with $T$ many Test queries, these proofs also lose an additional multiplicative factor $T$.

At CRYPTO 2018, Gjøsteen and Jager [24] proposed a tightly secure variant of it by introducing an additional message move into the ordinary signed DH protocol. They showed that if the signature scheme is tightly secure in the multiuser setting then their protocol is tightly secure. They required the underlying signature scheme to be strongly unforgeable against adaptive Corruption and Chosen-Message Attacks (StCorrCMA) which is a notion in the multiuser setting and an adversary can adaptively corrupt some of the honest users to see their secret keys. Moreover, they constructed a tightly multiuser secure signature scheme based on the decisional Diffie–Hellman (DDH) assumption in the random oracle model [4]. Combining these two results, they gave a practical three message fully tight AKE. We note that their tight security is proven in the less desirable MBG model, and, to the best of our knowledge, the MBG security can only non-tightly imply the SBG security [28]. Due to the "commitment problem", the additional message is crucial for the tightness of their protocol. In particular, the "commitment problem" seems to be the reason why most security proofs for AKEs are non-tight.

### 1.1. *Our Contribution*

In this paper, we propose a new tight security proof of the ordinary two-message signed Diffie–Hellman key exchange protocol in the random oracle model. More precisely, we prove the security of the signed DH protocol *tightly* based on the multiuser security of the underlying signature scheme in the random oracle model. Our proof improves upon the work of Gjøsteen and Jager [24] in the sense that we do not require any modification to the signed DH protocol and our tight multi-challenge security is in the SBG model. This implies that our analysis supports the optimal implementation of the ordinary signed DH protocol with theoretically sound security in a meaningful model.

Our technique is a new approach to resolve the "commitment problem". At the core of it is a new notion called *verifiable key exchange protocols*. We first briefly recall the "commitment problem" and give an overview of our approach.

Technical Difficulty: The "commitment problem". As explained in [24], this problem is the reason why almost all proofs of classical AKE protocols are non-tight. In a security proof of an AKE protocol, the reduction needs to embed a hard problem instance into the protocol messages of Test sessions so that in the end the reduction can extract a solution to the hard problem from the adversary $\mathcal{A}$. After the instance is embedded, $\mathcal{A}$ has not committed itself to which sessions it will query to Test yet, and, for instance, $\mathcal{A}$ can ask the reduction for Reveal queries on sessions with a problem instance embedded to get the corresponding session keys. At this point, the reduction cannot respond to these Reveal queries. A natural way to resolve this is to guess which sessions $\mathcal{A}$ will query Test on, and to embed a hard problem instance in those sessions only. However, this introduces an extremely large security loss. To resolve this "commitment problem", a tight reduction should be able to answer both Test and Reveal for every session without any guessing. Gjøsteen and Jager achieved this for the signed DH by adding an additional message.

In this paper, we show that this additional message is not necessary for tight security.

Our Approach: Verifiable Key Exchange. In this work, we, for simplicity, use the signed Diffie–Hellman protocol based on the plain Diffie–Hellman protocol [19] (as described in Fig. 1) to explain our approach. In the technical part, we abstract and present our idea with a new notion called verifiable key exchange protocols.

Let $\mathbb{G} := \langle g \rangle$ be a cyclic group of prime-order $p$ where the computational Diffie–Hellman (CDH) problem is hard. Let $(g^{\alpha}, g^{\beta})$ (where $\alpha, \beta \leftarrow_{\$} \mathbb{Z}_p$) be an instance of the CDH problem. By its random self-reducibility, we can efficiently randomize it to multiple independently distributed instances $(g^{\alpha_i}, g^{\beta_i})$, and given some $g^{\alpha_i \beta_i}$, we can extract the solution $g^{\alpha\beta}$.

For preparation, we assume that a Test session does not contain any valid forgeries. This can be tightly justified by the StCorrCMA security of the underlying signature scheme, which can be instantiated with the recent scheme in [17].

After that, an adversary can only observe the protocol transcripts or forward the honestly generated transcripts in arbitrary orders. This is the most important step for bounding such an adversary tightly without involving the "commitment problem". Our reduction embeds the randomized instance $(g^{\alpha_i}, g^{\beta_i})$ into each session. Now it seems we can answer neither Test nor Reveal queries: The answer has the form $K := \mathsf{H}(\text{ctxt}, g^{xy})$, but the term $g^{xy}$ cannot be computed by the reduction, since $g^x$ and $g^y$ are from the CDH challenge and the reduction knows neither $x$ nor $y$. However, our reduction can answer this by carefully simulating the random oracle $\mathsf{H}$ and keeping the adversary's view consistent. More precisely, we answer Test and Reveal queries with a random $K$, and we carefully program the random oracle $\mathsf{H}$ so that adversary $\mathcal{A}$ cannot detect this change. To achieve this, when we receive a random oracle query $\mathsf{H}(\text{ctxt}, Z)$, we answer it consistently if the secret element $Z$ corresponds to the context ctxt and ctxt belongs to one of the Test or Reveal queries. This check can be efficiently done by using the strong DH oracle [1]. Our approach is motivated by the two-message AKE in [14].

The approach described above can be abstracted by a notion called verifiable key exchange (VKE) protocols. Roughly speaking, a VKE protocol is firstly passively secure, namely a passive observer cannot compute the secret session key. Additionally, a VKE provides an oracle to an adversary to check whether a session key belongs to some honestly generated session, and to forward messages in a different order to create non-matching sessions. This VKE notion gives rise to a tight security proof of the signed DH protocol. We believe this is of independent interest.

On the Strong CDH Assumption. Our techniques require the Strong CDH assumption [1] for the security of our VKE protocol. We refer to [15, Appendix C] for a detailed analysis of this assumption in the generic group model (GGM). Without using the GGM, we can use the twinning technique [13] to remove this strong assumption and base the VKE security tightly on the (standard) CDH assumption. This approach will double the number of group elements. Alternatively, we can use the group of signed Quadratic Residues (QR) [27] to instantiate our VKE protocol, and then the VKE security is tightly based on the factoring assumption (by [27, Theorem 2]).

Real-World Impacts. As mentioned earlier, the signed DH protocol serves as a solid foundation for many real-world protocols, including the one in TLS 1.3 [42], IKE [26], and the Station-to-Station [20] protocols. We believe our approach can naturally be extended to tighten the security proofs of these protocols. In particular, our notion of VKE protocols can abstract some crucial steps in a recent tight proof of TLS 1.3 [15].

Another practical benefit of our tight security proof is that, even if we implement the underlying signature with a standardized, non-tight scheme (such as Ed25519 [8] or RSA-PKCS #1 v1.5 [40]), our implementation does not need to lose the additional factor that is linear in the number of sessions. In today's Internet, there can be easily $2^{30}$ sessions per year. For instance, Facebook has about $2^{30}$ monthly active users[1]. Assuming that each user only logs in once a month, this already leads to $2^{30}$ sessions.

### 1.2. *Protocol Comparison*

We compare the instantiation of signed DH according to our tight proof with the existing explicitly authenticated key exchange protocols in Fig. 2. For complete tightness, all these protocols require tight multiuser security of their underlying signature scheme. We implement the signature scheme in all protocols with the recent efficient scheme from Diemert et al. [17] whose signatures contain 3 $\mathbb{Z}_p$ elements, and whose security is based on the DDH assumption. The implementation of TLS is according to the recent tight proofs in [15,18], and we instantiate the underlying signature scheme with the same DDH-based scheme from [17].

We note that the non-tight protocol from Cohn-Gorden et al. [14], whose security loss is linear in the number of users, has better communication efficiency (2, 0, 0). However, its security is weaker than all protocols listed in Fig. 2, since their protocol is only implicitly authenticated and achieves weak perfect forward secrecy.

We detail the comparison with $\mathsf{JKRS}$ [28]. Using the DDH-based signature scheme in [17], the communication complexity of our signed DH protocol is (2, 0, 6), while that of $\mathsf{JKRS}$ is (5, 1, 3). We suppose the efficiency of our protocol is comparable to $\mathsf{JKRS}$.

Our main weakness is that our security model is weaker than that of $\mathsf{JKRS}$. Namely, ours does not allow adversaries to corrupt any internal secret state. We highlight that our proof does not inherently rely on any decisional assumption. In particular, if there is a tightly multiuser secure signature scheme based on only search assumptions, our proof

[1] Cf. https://investor.fb.com/investor-news/press-release-details/2022/Meta-Reports-Fourth-Quarter-and-Full-Year-2021-Results/default.aspx

| **Protocol** | **Comm.**<br>$(\mathbb{G}, \{0,1\}^\lambda, \mathbb{Z}_p)$ | **#Msg.** | **Assumption** | **Auth.** | **Model** | **State Reveal** | **Security loss** |
|---|---|---|---|---|---|---|---|
| $\mathsf{TLS}^*$ [15,18] | $(2,4,6)$ | 3 | $\mathsf{StCDH} + \mathsf{DDH}$ | expl. | SBG | no | $O(1)$ |
| $\mathsf{GJ}$ [24] | $(2,1,6)$ | 3 | $\mathsf{DDH}$ | expl. | MBG | no | $O(1)$ |
| $\mathsf{LLGW}$ [36] | $(3,0,6)$ | 2 | $\mathsf{DDH}$ | expl. | MBG | no | $O(1)$ |
| $\mathsf{JKRS}$ [28] | $(5,1,3)$ | 2 | $\mathsf{DDH}$ | expl. | SBG | yes | $O(1)$ |
| This work | $(2,0,6)$ | 2 | $\mathsf{StCDH} + \mathsf{DDH}$ | expl. | SBG | no | $O(1)$ |

**Fig. 2.** Comparison of $\mathsf{AKE}$ protocols. We denote **Comm.** as the communication complexity of the protocols in terms of the number of group elements, hashes and $\mathbb{Z}_p$ elements (which is due to the use of the signature scheme in [17]). The column **Model** lists the $\mathsf{AKE}$ security model and distinguishes between multi-bit guessing (MBG) and the single-bit-guessing (SBG) security .

directly gives a tightly secure AKE based on search assumptions only, which is not the case for [28].

### 1.3. *An Extension and Open Problems*

We extend our approach to group AKE (GAKE) protocols, where a group of users can agree on a session key, and construct the first tightly secure GAKE protocol. Research on GAKE has a long history and several GAKE protocols have been constructed in the literature [9–11,25,31]. However, none of the existing GAKE protocols enjoy a tight security proof. We suppose that tight security is more desirable for GAKE than AKE, since many applications require GAKE protocols (such as online audio–video conference systems and instant messaging [43]) are often in a truly large-scale setting.

Similar to the two-party setting, we propose the notion of verifiable group key exchange (VGKE) protocols and transform a VGKE to GAKE using a signature scheme. Our transformation is tightness-preserving. As an instantiation of our approach, we prove that under the strong CDH assumption the classical Burmester–Desmedt protocol is a tightly secure VGKE protocol [12]. Combining with a tightly $\mathsf{StCorrCMA}$-secure signature (for instance, [17]), it yields the *first* tightly secure GAKE protocol. Alternatively, our transformation can be viewed as a tight improvement on the (non-tight) generic compiler of Katz and Yung [31] where we require the underlying non-authenticated group key exchange protocol to be verifiable.

Open Problems. We do not know of any tightly multiuser secure signature schemes with corruptions based on a search assumption, and the schemes in [39] based on search assumptions do not allow any corruption. It is therefore insufficient for our purpose, and we leave constructing a tightly secure AKE based purely on search assumptions as an open problem.

### 1.4. *History of This Paper*

This is the full version of a paper published at CT-RSA 2021 [38]. The main change here is to extend our approach in the group key exchange setting and propose the first tightly secure GAKE protocol (cf. Sect. 6). Due to this main extension, we (slightly) change the title to the current one. Moreover, we give a detailed proof for the multiuser security of Schnorr's signature scheme in the generic group model (cf. Appendix A).

## 2. Preliminaries

For $n \in \mathbb{N}$, let $[n] = \{1, \dots, n\}$. For a finite set $\mathcal{S}$, we denote the sampling of a uniform random element $x$ by $x \leftarrow_{\$} \mathcal{S}$. By $[\![B]\!]$, we denote the bit that is 1 if the evaluation of the Boolean statement $B$ is **true** and 0 otherwise.

ALGORITHMS. For an algorithm $\mathcal{A}$ which takes $x$ as input, we denote its computation by $y \leftarrow \mathcal{A}(x)$ if $\mathcal{A}$ is deterministic, and $y \leftarrow_{\$} \mathcal{A}(x)$ if $\mathcal{A}$ is probabilistic. We assume all the algorithms (including adversaries) in this paper to be probabilistic unless we state it. We denote an algorithm $\mathcal{A}$ with access to an oracle O by $\mathcal{A}^{\mathrm{O}}$. In terms of running time, if a reduction's running time $t'$ is dominated by that of an adversary $t$ (more precisely, $t' = t + s$ where $s \ll t$), we write $t' \approx t$.

GAMES. We use code-based games [6] to present our definitions and proofs. We implicitly assume all Boolean flags to be initialized to 0 (**false**), numerical variables to 0, sets to $\emptyset$ and strings to $\perp$. We make the convention that a procedure terminates once it has returned an output. $G^{\mathcal{A}} \Rightarrow b$ denotes the final (Boolean) output $b$ of game $G$ running adversary $\mathcal{A}$, and if $b = 1$ we say $\mathcal{A}$ wins $G$. The randomness in $\Pr[G^{\mathcal{A}} \Rightarrow 1]$ is over all the random coins in game $G$. Within a procedure, "**abort**" means that we terminate the run of an adversary $\mathcal{A}$.

DIGITAL SIGNATURES. We recall the syntax and security of a digital signature scheme. Let $\mathsf{par}$ be some system parameters shared among all participants.

**Definition 1.** (*Digital Signature*) A digital signature scheme $\mathsf{SIG} := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is defined as follows.

- The key generation algorithm $\mathsf{Gen}(\mathsf{par})$ returns a public key and a secret key $(\mathsf{pk}, \mathsf{sk})$. We assume that $\mathsf{pk}$ implicitly defines a message space $\mathcal{M}$ and a signature space $\Sigma$.
- The signing algorithm $\mathsf{Sign}(\mathsf{sk}, m \in \mathcal{M})$ returns a signature $\sigma \in \Sigma$ on $m$.
- The deterministic verification algorithm $\mathsf{Ver}(\mathsf{pk}, m, \sigma)$ returns 1 (accept) or 0 (reject).

$\mathsf{SIG}$ is perfectly correct, if for all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{par})$ and all messages $m \in \mathcal{M}$, $\mathsf{Ver}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$.

In addition, we say that $\mathsf{SIG}$ has $\alpha$ bits of (public) key min-entropy if an honestly generated public key $\mathsf{pk}$ is chosen from a distribution with at least $\alpha$ bits min-entropy. Formally, for all bit-strings $\mathsf{pk}'$ we have $\Pr[\mathsf{pk} = \mathsf{pk}' : (\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{par})] \leq 2^{-\alpha}$.

We include the definition of entropy here because our proofs require an estimate on the probability of a collision in the public keys.

**Definition 2.** ($\mathsf{StCorrCMA}$ *Security* [17,24]) A digital signature scheme $\mathsf{SIG}$ is $(t, \varepsilon, \mu, Q_s, Q_{\mathrm{COR}})$-$\mathsf{StCorrCMA}$ secure (Strong unforgeability against Corruption

```
GAME StCorrCMA:
    for $i \in [\mu]$: $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{Gen}(\mathsf{par})$
    $(i^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathrm{O}}(\{\mathsf{pk}_i\}_{i \in [\mu]})$
    return $[\![\mathsf{Ver}(\mathsf{pk}_{i^*}, m^*, \sigma^*)$
        $\wedge (i^*, m^*, \sigma^*) \notin \mathcal{L}_{\mathcal{S}} \wedge i^* \notin \mathcal{L}_{\mathcal{C}}]\!]$

SIGN$(i, m)$:
    $\sigma := \mathsf{Sign}(\mathsf{sk}_i, m)$
    $\mathcal{L}_{\mathcal{S}} := \mathcal{L}_{\mathcal{S}} \cup \{(i, m, \sigma)\}$
    return $\sigma$

CORR$(i)$:
    $\mathcal{L}_{\mathcal{C}} := \mathcal{L}_{\mathcal{C}} \cup \{i\}$
    return $\mathsf{sk}_i$
```

**Fig. 3.** StCorrCMA security game for a signature scheme SIG. $\mathcal{A}$ has access to the oracles $\mathrm{O} := \{\textsc{Sign}, \textsc{Corr}\}$
.

and Chosen Message Attacks), if for all adversaries $\mathcal{A}$ running in time at most $t$, interacting with $\mu$ users, making at most $Q_s$ queries to the signing oracle SIGN, and at most $Q_{\textsc{Cor}}$ ($Q_{\textsc{Cor}} < \mu$) queries to the corruption oracle CORR as in Fig. 3, we have

$$\Pr[\mathsf{StCorrCMA}^{\mathcal{A}} \Rightarrow 1] \leq \varepsilon.$$

SECURITY IN THE RANDOM ORACLE MODEL. A common approach to analyze the security of signature schemes that involve a hash function is to use the random oracle model [4] where hash queries are answered by an oracle $\mathsf{H}$, where $\mathsf{H}$ is defined as follows: On input $x$, it first checks whether $\mathsf{H}(x)$ has previously been defined. If so, it returns $\mathsf{H}(x)$. Otherwise, it sets $\mathsf{H}(x)$ to a uniformly random value in the range of $\mathsf{H}$ and then returns $\mathsf{H}(x)$. We parameterize the maximum number of hash queries in our security notions. For instance, we define $(t, \varepsilon, \mu, Q_s, Q_{\textsc{Cor}}, Q_{\mathsf{H}})$-StCorrCMA as security against any adversary that makes at most $Q_{\mathsf{H}}$ queries to $\mathsf{H}$ in the StCorrCMA game. Furthermore, we make the standard convention that any random oracle query that is asked as a result of a query to the signing oracle in the StCorrCMA game is also counted as a query to the random oracle. This implies that $Q_s \leq Q_{\mathsf{H}}$.

SIGNATURE SCHEMES. The tight security of our authenticated key exchange (AKE) protocols is established based on the StCorrCMA security of the underlying signature schemes. To obtain a completely tight AKE, we use the recent signature scheme from [17] to implement our protocols.

By adapting the non-tight proof in [23], the standard unforgeability against chosen-message attacks (UF-CMA) notion for signature schemes implies the StCorrCMA security of the same scheme non-tightly (with security loss $\mu$). Thus, many widely used signature schemes (such as the Schnorr [44], Ed25519 [8] and RSA-PKCS #1 v1.5 [40] signature schemes) are non-tightly StCorrCMA secure. We do not know any better reductions for these schemes. We leave proving the StCorrCMA security of these schemes without losing a linear factor of $\mu$ as a future direction. However, our tight proof for the signed DH protocol strongly indicates that the aforementioned non-tight reduction is optimal for these practical schemes. This is because if we can prove these schemes tightly secure, we can combine them with our tight proof to obtain a tightly secure AKE with unique and verifiable private keys, which may contradict the impossibility result from [14].

For the Schnorr signature, we analyze its StCorrCMA security in the generic group model (GGM) [37,45]. We recall the Schnorr signature scheme below and show the GGM bound of its StCorrCMA security in Theorem 1.

Let $\mathsf{par} = (p, g, \mathbb{G})$, where $\mathbb{G} = \langle g \rangle$ is a cyclic group of prime order $p$ with a hard discrete logarithm problem. Let $\mathsf{H} : \{0, 1\}^* \to \mathbb{Z}_p$ be a hash function. Schnorr's signature scheme, $\mathsf{Schnorr} := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, is defined as follows:

```
Gen(par):
    $x \leftarrow_\$ \mathbb{Z}_p$
    $X := g^x$
    $\mathsf{pk} := X$
    $\mathsf{sk} := x$
    $\mathbf{return}(\mathsf{pk}, \mathsf{sk})$
```

```
Sign(sk, m):
    $\mathbf{parse} x =: \mathsf{sk}$
    $r \leftarrow_\$ \mathbb{Z}_p;\ \ R := g^r$
    $h := \mathsf{H}(R, m)$
    $s := r + x \cdot h$
    $\mathbf{return}(h, s)$
```

```
Ver(pk, m, σ):
    $\mathbf{parse}(h, s) =: \sigma$
    $\mathbf{parse} X =: \mathsf{pk}$
    $R = g^s \cdot X^{-h}$
    $\mathbf{return}[\![\mathsf{H}(R, m) = h]\!]$
```

**Theorem 1.** (StCorrCMA Security of Schnorr in the GGM) *Schnorr's signature* SIG *is* $(t, \varepsilon, \mu, Q_s, Q_{\text{COR}}, Q_{\mathsf{H}})$-StCorrCMA-*secure in the* GGM *and in the programmable random oracle model, where*

$$\varepsilon \leq \frac{(Q_{\mathbb{G}} + \mu + 1)^2}{2p} + \frac{(\mu - Q_{\text{COR}})}{p} + \frac{Q_{\mathsf{H}} Q_s + 1}{p}, \quad \textit{and} \quad t' \approx t.$$

*Here, $Q_{\mathbb{G}}$ is the number of group operations queried by the adversary.*

The proof of Theorem 1 is following the approach in [3,32]: We first define an algebraic interactive assumption, CorrIDLOG, which is tightly equivalent to the StCorrCMA security of Schnorr, and then we analyze the hardness of CorrIDLOG in the GGM. CorrIDLOG stands for Interactive Discrete Logarithm with Corruption. It is motivated by the IDLOG (Interactive Discrete Logarithm) assumption in [32]. CorrIDLOG is a stronger assumption than IDLOG in the sense that it allows an adversary to corrupt the secret exponents of some public keys. Details are given in Appendix A.

## 3. Security Model for Two-Message Authenticated Key Exchange

In this section, we use the security model in [28] to define the security of two-message authenticated key exchange protocols. This section is almost verbatim to Section 4 of [28]. We highlight the difference we make for our protocol: Since our protocols do not have security against (ephemeral) state reveal attacks (as in the extended Canetti-Krawczyk (eCK) model [34]), we do not consider state reveals in our model.

A two-message key exchange protocol $\mathsf{AKE} := (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathsf{R}}, \mathsf{Der}_{\mathrm{I}})$ consists of four algorithms which are executed interactively by two parties as shown in Fig. 4. We denote the party which initiates the session by $\mathsf{P}_i$ and the party which responds to the session by $\mathsf{P}_r$. 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 $\mathsf{sk}_i$ and the responder's long-term public key $\mathsf{pk}_r$, and outputs a message $m_i$ and a state st. The responder's derivation algorithm $\mathsf{Der}_{\mathsf{R}}$ takes as input the responder's long-term secret key, the initiator's public key $\mathsf{pk}_i$ and a message $m_i$. It computes a message $m_r$ and a session key $K$. The initiator's derivation algorithm $\mathsf{Der}_{\mathrm{I}}$ inputs the initiator's long-term key $\mathsf{sk}_i$, the responder's long-term public key $\mathsf{pk}_r$, the responder's message $m_r$ and the state st. Note that the responder is not required to save any internal state information besides the session key $K$.

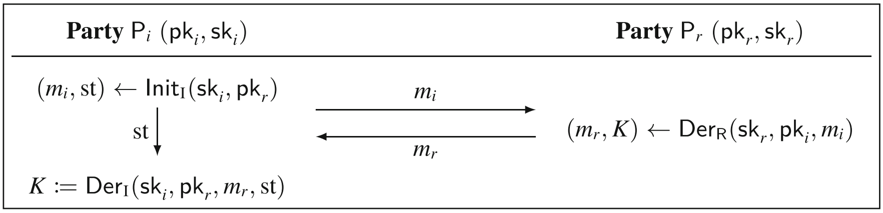
Party $\mathsf{P}_i$ $(\mathsf{pk}_i, \mathsf{sk}_i)$
Party $\mathsf{P}_r$ $(\mathsf{pk}_r, \mathsf{sk}_r)$
$(m_i, \mathrm{st}) \leftarrow \mathsf{Init}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r)$
$m_i$
st
$m_r$
$(m_r, K) \leftarrow \mathsf{Der}_{\mathsf{R}}(\mathsf{sk}_r, \mathsf{pk}_i, m_i)$
$K := \mathsf{Der}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st})$

**Fig. 4.** Running an authenticated key exchange protocol between two parties .

We give a security game written in pseudocode. We define a model for *explicit authenticated* protocols achieving (full) forward secrecy instead of weak forward secrecy. Namely, an adversary in our model can be active and corrupt the user who owns the TEST session sID*, and the only restriction is that if there is no matching session to sID*, then the peer of sID* must not be corrupted before the session finishes.

Here, explicit authentication means entity authentication in the sense that a party can explicitly confirm that he is talking to the actual owner of the recipient's public key. The key confirmation property is only implicit [21], where a party is assured that the other identified party can compute the same session key. The game $\mathsf{IND\text{-}FS}$ is given in Figs. 5 and 6. We refer readers to [16] for more details on different types of authentication for key exchange protocols.

EXECUTION ENVIRONMENT. We consider $\mu$ parties $\mathsf{P}_1, \ldots, \mathsf{P}_\mu$ with long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$, $n \in [\mu]$. When two parties A and B want to communicate, the initiator, say, A first creates a session. To identify this session, we increase the global identification number sID and assign the current state of sID to identify this session owned by A. The state of sID will increase after every assignment. Moreover, a message will be sent to the responder. The responder then similarly creates a corresponding session which is assigned the current state of sID. Hence, each conversation includes two sessions. We then define variables in relation to the identifier sID:

- init[sID] $\in [\mu]$ denotes the initiator of the session.
- resp[sID] $\in [\mu]$ 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.
- $\mathsf{Msg}_{\mathsf{I}}$[sID] denotes the message that was computed by the initiator.
- $\mathsf{Msg}_{\mathsf{R}}$[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 $\text{SESSION}_{\text{I}}$ and $\text{SESSION}_{\mathsf{R}}$, where the first one starts a session of type "In" and the second one of type "Re". The $\text{SESSION}_{\mathsf{R}}$ oracle also runs the $\mathsf{Der}_{\mathsf{R}}$ algorithm to compute its session key and complete the session, as it has access to all the required variables. In order to complete the initiator's session, the oracle $\text{DER}_{\text{I}}$ has to be queried.

Following [28], we do not allow the adversary to register adversarially controlled parties by providing long-term public keys, as the registered keys would be treated no differently than regular corrupted keys. If we would include the key registration oracle,

```
GAME IND-FS
for $n \in [\mu]$
   $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{AKE}}$
$b \overset{\$}{\leftarrow} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
for $\mathrm{sID}^* \in \mathcal{S}$
   if $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$
      return $b$                          //session not fresh
   if $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
      return $b$                          //no valid attack
return $[\![b = b']\!]$

$\textsc{Session}_{\mathsf{R}}((i,r) \in [\mu]^2, m_i)$
$\mathrm{cnt_S}$ ++
$\mathrm{sID} := \mathrm{cnt_S}$
$(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i, r)$
$\mathrm{type}[\mathrm{sID}] :=$ “Re”
$\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[i]$
$(m_r, K) \leftarrow \mathsf{Der}_{\mathsf{R}}(\mathsf{sk}_r, \mathsf{pk}_i, m_i)$
$(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) := (m_i, m_r, K)$
return $(\mathrm{sID}, m_r)$

$\textsc{Test}(\mathrm{sID})$
if $\mathrm{sID} \in \mathcal{S}$ return $\perp$          //already tested
if $\mathrm{sKey}[\mathrm{sID}] = \perp$ return $\perp$
$\mathcal{S} := \mathcal{S} \cup \{\mathrm{sID}\}$
$K_0^* := \mathrm{sKey}[\mathrm{sID}]$
$K_1^* \overset{\$}{\leftarrow} \mathcal{K}$
return $K_b^*$

$\textsc{Session}_{\mathrm{I}}((i,r) \in [\mu]^2)$
$\mathrm{cnt_S}$ ++
$\mathrm{sID} := \mathrm{cnt_S}$
$(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i, r)$
$\mathrm{type}[\mathrm{sID}] :=$ “In”
$(m_i, \mathrm{st}) \leftarrow \mathsf{Init}_{\mathsf{I}}(\mathsf{sk}_i, \mathsf{pk}_r)$
$(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) := (m_i, \mathrm{st})$
return $(\mathrm{sID}, m_i)$

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID} \in [\mathrm{cnt_S}], m_r)$
if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$ or $\mathrm{type}[\mathrm{sID}] \neq$ “In”
   return $\perp$                        //no re-use
$(i, r) := (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
$\mathrm{st} := \mathrm{state}[\mathrm{sID}]$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[r]$
$K := \mathsf{Der}_{\mathsf{I}}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st})$
$(\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) := (m_r, K)$
return $\epsilon$

$\textsc{Reveal}(\mathrm{sID})$
$\mathrm{revealed}[\mathrm{sID}] := \mathbf{true}$
return $\mathrm{sKey}[\mathrm{sID}]$

$\textsc{Corr}(n \in [\mu])$
$\mathrm{corrupted}[n] := \mathbf{true}$
return $\mathsf{sk}_n$
```

**Fig. 5.** Game IND-FS for AKE. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}\}$. Helper procedures Fresh and Valid are defined in Fig. 6. If there exists any test session which is neither fresh nor valid, the game will return $b$ .

```
$\textsc{Fresh}(\mathrm{sID}^*)$
$(i^*, r^*) := (\mathrm{init}[\mathrm{sID}^*], \mathrm{resp}[\mathrm{sID}^*])$
$\mathfrak{M}(\mathrm{sID}^*) := \{\mathrm{sID} \mid (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) = (i^*, r^*) \ \wedge \ (\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]) =$
                  $(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}^*], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}^*]) \ \wedge \ \mathrm{type}[\mathrm{sID}] \neq \mathrm{type}[\mathrm{sID}^*]\}$   //matching sessions
if $\mathrm{revealed}[\mathrm{sID}^*]$ or $(\exists \mathrm{sID} \in \mathfrak{M}(\mathrm{sID}^*) : \mathrm{revealed}[\mathrm{sID}] = \mathbf{true})$
   return false                          //$\mathcal{A}$ trivially learned the test session's key
if $\exists \mathrm{sID} \in \mathfrak{M}(\mathrm{sID}^*)$ s. t. $\mathrm{sID} \in \mathcal{S}$
   return false                          //$\mathcal{A}$ also tested a matching session
return true

$\textsc{Valid}(\mathrm{sID}^*)$
$(i^*, r^*) := (\mathrm{init}[\mathrm{sID}^*], \mathrm{resp}[\mathrm{sID}^*])$
$\mathfrak{M}(\mathrm{sID}^*) := \{\mathrm{sID} \mid (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) = (i^*, r^*) \ \wedge \ (\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]) =$
                  $(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}^*], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}^*]) \ \wedge \ \mathrm{type}[\mathrm{sID}] \neq \mathrm{type}[\mathrm{sID}^*]\}$   //matching sessions
for attack $\in$ Table 2
   if attack $=$ true return true
return false
```

**Fig. 6.** Helper procedures Fresh and Valid for game IND-FS defined in Fig. 5. 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 2 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 .

then our proof requires a stronger notion of signature schemes in the sense that our signature challenger can generate the system parameters with some trapdoor. With the trapdoor, the challenger can simulate a valid signature under the adversarially registered public keys. This is the case for the Schnorr signature and the tight scheme in [17], since they are honest-verifier zero-knowledge and the aforementioned property can be achieved by programming the random oracles.

Finally, the adversary has access to oracles CORR and REVEAL to obtain secret information. 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 $\mathsf{P}_n$ was given to the adversary.
- revealed[sID] denotes whether the session key was given to the adversary.
- peerCorrupted[sID] denotes whether the peer of the session was corrupted and its long-term key was given to the adversary at the time the owner's session key was 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:

- **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 ($\mathsf{Msg_I}$[sID] = $\mathsf{Msg_I}$[sID$'$] and $\mathsf{Msg_R}$[sID] = $\mathsf{Msg_R}$[sID$'$]) and they are of different types (type[sID] $\neq$ type[sID$'$]).

Our protocols use signatures to preserve integrity so that any successful no-match attacks described in [35] will lead to a signature forgery and thus can be excluded.

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, and store test sessions in a set $\mathcal{S}$. The adversary can obtain information on the interactions between two parties by querying the long-term secret keys 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 Table 2.

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”),

**Table 1.** Full table of attacks for adversaries against explicitly authenticated two-message protocols.

| | $\mathcal{A}$ gets (Initiator, Responder) | corrupted[$i^*$] | corrupted[$r^*$] | peerCorrupted[sID$^*$] | type[sID$^*$] | $\lvert\mathfrak{M}(\text{sID}^*)\rvert$ |
|---|---|---|---|---|---|---|
| 0. | **multiple matching sessions** | – | – | – | – | $> 1$ |
| 1. | **(long-term, long-term)** | – | – | – | “In” | 1 |
| 2. | **(long-term, long-term)** | – | – | – | “Re” | 1 |
| 3. | **(long-term, $\bot$)** | – | **T** | **T** | “In” | 0 |
| 4. | **($\bot$, long-term)** | **T** | – | **T** | “Re” | 0 |
| 5. | **(long-term, long-term)** | – | – | **F** | “In” | 0 |
| 6. | **(long-term, long-term)** | – | – | **F** | “Re” | 0 |

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.” The trivial attacks where the session’s peer is corrupted when the key is derived, and the corresponding variables are set to **T**, are marked with gray . The $\bot$ symbol indicates that the adversary cannot query anything more from this party, as he already possesses the long-term key

- all combinations of long-term secret key reveal, taking into account when a corruption happened (corrupted and peerCorrupted variables),
- the number of matching sessions, i.e., whether the adversary acted passively (matching session) or actively (no matching session).

The purpose of these tables is to make our proofs precise by listing all the possible attacks.

How to read the tables. Table 1 lists all possible attacks from an adversary. By excluding trivial attacks and merging similar attacks, we obtain Table 2 from Table 1. If the set of variables corresponding to a test session is set as in any row of Table 2, this row will evaluate to **true** in line 10 in Fig. 6. We now describe the different attacks in Table 1 in more detail:

Row 0. If the protocol does not use appropriate randomness, it should not be considered secure. In this case, there can be multiple matching sessions to a test session, which an adversary can take advantage of. For an honest run of the protocol, the underlying min-entropy ensures that this attack will only happen with negligible probability.

Row 1. Here, the tested session has one matching session, is of type “In”, and both parties might be corrupted. Since there is a matching session, the adversary has acted passively during the execution of the protocol. Thus, even if both parties were corrupted during the execution, the adversary cannot break the $\mathsf{AKE}$ security without breaking the passive security of the underlying protocol. Hence, it should make no difference if the parties were corrupted before or after the key was computed, and the corrupted and peerCorrupted columns can take any value.

**Table 2.** Distilled table of attacks for adversaries against explicitly authenticated two-message protocols without ephemeral state reveals.

| $\mathcal{A}$ gets (Initiator, Responder) | corrupted[$i^*$] | corrupted[$r^*$] | peerCorrupted[sID$^*$] | type[sID$^*$] | $\lvert\mathfrak{M}(\mathrm{sID}^*)\rvert$ |
|---|---|---|---|---|---|
| 0. **multiple matching sessions** | – | – | – | – | $> 1$ |
| 1.+2. **(long-term, long-term)** | – | – | – | – | 1 |
| 5.+6. **(long-term, long-term)** | – | – | **F** | – | 0 |

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"

Row 2. This attack is similar to the one above, the only difference is the session type.

Row 3. Here, the responder of the session was corrupted when the initiator computed its key, and there is no matching session. This means that the adversary has performed an active attack and changed or reordered the message being sent. This can lead to a trivial attack, because the adversary can impersonate the responder with the corrupted secret key. By knowing the underlying message, he can compute the same session key as the initiator will compute, and test the initiators session. Whether the adversary corrupts the initiator makes no difference, and hence this column can take any value.

Row 4. Similar to the attack above, with the types switched, and hence the initiator was corrupted by the time the responder computed the key. This leads to a trivial attack in the same way.

Row 5. Here, there is no matching session, but we specify that the responder was not corrupted when the initiator computed its key. The adversary can choose whether or not to corrupt the initiator before the responder computes its key. The key point is that whether he can impersonate the initiator or not, he does not know the internal state of the initiator, and to break security he must either break the underlying key exchange protocol, or impersonate the responder and break the authentication directly. Hence, this column can take any value. After the initiators key is computed, it should not matter whether the responder gets corrupted or not, and hence, this column can also take any value.

Row 6. Similar to above, but with the types changed so that the initiator was not corrupted when the responder computed its key.

From the 6 attacks in total presented in Table 1, rows (3.) and (4.) are trivial wins for the adversary and can thus be excluded. Note that rows (1.) and (2.) denote similar attacks against initiator and responder sessions. Since the session's type does not change the queries the adversary is allowed to make in this case, we can merge these rows. For the same reason, we can also merge rows (5.) and (6.). The resulting table is given in Table 2.

Attacks covered in our model capture *forward secrecy* (FS) and *key compromise impersonation* (KCI) attacks.

Note that we do not include reflection attacks, where the adversary makes a party run the protocol with himself. For the $\mathsf{KE}_{\mathsf{DH}}$ protocol, we could include these and create an additional reduction to the square Diffie–Hellman assumption (given $g^x$, to compute $g^{x^2}$), but for simplicity of our presentation we will not consider reflection attacks in this paper.

For all test sessions, at least one attack has to evaluate to true. Then, the adversary wins if he distinguishes the session keys from uniformly random keys which he obtains through queries to the Test oracle.

**Definition 3.** (*Key Indistinguishability of AKE*) We define game $\mathsf{IND\text{-}FS}$ as in Figs. 5 and 6. A protocol $\mathsf{AKE}$ is $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}})$-$\mathsf{IND\text{-}FS}$-secure if for all adversaries $\mathcal{A}$ attacking the protocol in time $t$ with $\mu$ users, $S$ sessions, $T$ test queries and $Q_{\mathrm{COR}}$ corruptions, we have

$$\left|\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right| \leq \varepsilon.$$

Note that if there exists a session which is neither fresh nor valid, the game outputs the bit $b$, which implies that $\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}} \Rightarrow 1] = 1/2$, giving the adversary an advantage equal to 0. This captures that an adversary will not gain any advantage by performing a trivial attack.

## 4. Verifiable Key Exchange Protocols

A key exchange protocol $\mathsf{KE} := (\mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathsf{R}}, \mathsf{Der}_{\mathrm{I}})$ can be run between two (unauthenticated) parties $i$ and $r$, and can be visualized as in Fig. 4, but with differences where (1): parties do not hold any public key or private key, and (2): public and private keys in algorithms $\mathsf{Init}_{\mathrm{I}}$, $\mathsf{Der}_{\mathsf{R}}$, $\mathsf{Der}_{\mathrm{I}}$ are replaced with the corresponding users' (public) identities.

The standard signed Diffie–Hellman (DH) protocol can be viewed as a generic way to transform a passively secure key exchange protocol to an actively secure AKE protocol using digital signatures. Our tight transformation does not modify the construction of the signed DH protocol, but requires a security notion (i.e., One-Wayness against Honest and key Verification attacks, or $\mathsf{OW\text{-}HV}$) that is (slightly) stronger than passive security: Namely, in addition to passive attacks, an adversary is allowed to check if a key corresponds to some honestly generated transcripts and to forward transcripts in a different order to create non-matching sessions. Here, we require that all the involved transcripts must be honestly generated by the security game and not by the adversary. This is formally defined by Definition 4 with security game $\mathsf{OW\text{-}HV}$ as in Fig. 7.

**Definition 4.** (One-Wayness against Honest and key Verification attacks ($\mathsf{OW\text{-}HV}$)) A key exchange protocol $\mathsf{KE}$ is $(t, \varepsilon, \mu, S, Q_V)$-$\mathsf{OW\text{-}HV}$ secure, where $\mu$ is the number of users, $S$ is the number of sessions and $Q_V$ is the number of calls to KVer, if for all adversaries $\mathcal{A}$ attacking the protocol in time at most $t$, we have

```
GAME OW-HV
$(\mathrm{sID}^*, K^*) \xleftarrow{\$} \mathcal{A}^{\mathrm{O}}(\mu)$
if $\mathrm{sID}^* > \mathrm{cnt_S}$
    return 0
else
    return $\textsc{KVer}(\mathrm{sID}^*, K^*)$

$\textsc{KVer}(\mathrm{sID}, K)$
return $[\![\mathrm{sKey}[\mathrm{sID}] = K]\!]$

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, Y)$
if $\mathrm{sKey}[\mathrm{sID}] \neq \bot$ or $\mathrm{type}[\mathrm{sID}] \neq$ "In"
    return $\bot$
if $\forall \mathrm{sID}' \in [\mathrm{cnt_S}] : \mathsf{Msg_R}[\mathrm{sID}'] \neq Y$
    return $\bot$                          //$Y$ is not honest
$(i, r) := (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
$\mathrm{st} := \mathrm{state}[\mathrm{sID}]$
$K := \mathsf{Der_I}(i, r, Y, \mathrm{st})$
$(\mathsf{Msg_R}[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) := (Y, K)$
return $\epsilon$
```

```
$\textsc{Session}_{\mathrm{I}}((i, r) \in [\mu]^2)$              //$i \neq r$
$\mathrm{cnt_S}$ ++
$\mathrm{sID} := \mathrm{cnt_S}$
$(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i, r)$
$\mathrm{type}[\mathrm{sID}] :=$ "In"
$(X, \mathrm{st}) \xleftarrow{\$} \mathsf{Init_I}(i, r)$
$(\mathsf{Msg_I}[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) := (X, \mathrm{st})$
return $(\mathrm{sID}, X)$

$\textsc{Session}_{\mathrm{R}}((i, r) \in [\mu]^2, X)$           //$i \neq r$
if $\forall \mathrm{sID} \in [\mathrm{cnt_S}] : \mathsf{Msg_I}[\mathrm{sID}] \neq X$
    return $\bot$                       //$X$ is not honest
$\mathrm{cnt_S}$ ++
$\mathrm{sID}' := \mathrm{cnt_S}$
$(\mathrm{init}[\mathrm{sID}'], \mathrm{resp}[\mathrm{sID}']) := (i, r)$
$\mathrm{type}[\mathrm{sID}'] :=$ "Re"
$\mathsf{Msg_I}[\mathrm{sID}'] := X$
$(Y, K') \xleftarrow{\$} \mathsf{Der_R}(r, i, X)$
$\mathsf{Msg_R}[\mathrm{sID}'] := Y$
$\mathrm{sKey}[\mathrm{sID}'] := K'$
return $(\mathrm{sID}', Y)$
```

**Fig. 7.** Game OW-HV for KE. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{KVer}\}$ .

$$\Pr[\mathsf{OW\text{-}HV}^{\mathcal{A}} \Rightarrow 1] \leq \varepsilon.$$

We require that a key exchange protocol KE has $\alpha$ *bits of min-entropy*, i.e., that for all messages $m'$ we have $\Pr[m = m'] \leq 2^{-\alpha}$, where $m$ is output by either $\mathsf{Init_I}$ or $\mathsf{Der_R}$.

### 4.1. *Example: Plain Diffie–Hellman Protocol*

We show that the plain Diffie–Hellman (DH) protocol over prime-order group [19] is a OW-HV-secure key exchange under the strong computational DH (StCDH) assumption [1]. We use our syntax to recall the original DH protocol $\mathsf{KE_{DH}}$ in Fig. 8.

Let $\mathsf{par} = (p, g, \mathbb{G})$ be a set of system parameters, where $\mathbb{G} := \langle g \rangle$ is a cyclic group of prime order $p$.

**Definition 5.** (*Strong CDH Assumption*) The strong CDH (StCDH) assumption is said to be $(t, \varepsilon, Q_{\mathrm{DH}})$-hard in $\mathsf{par} = (p, g, \mathbb{G})$, if for all adversaries $\mathcal{A}$ running in time at most $t$ and making at most $Q_{\mathrm{DH}}$ queries to the DH predicate oracle $\textsc{Dh}_a$, we have:

$$\Pr\left[ Z = B^a \,\middle|\, \begin{array}{l} a, b \leftarrow_\$ \mathbb{Z}_p;\ \ A := g^a\ \ B := g^b \\ Z \leftarrow_\$ \mathcal{A}^{\textsc{Dh}_a}(A, B) \end{array} \right] \leq \varepsilon,$$

where the DH predicate oracle $\textsc{Dh}_a(C, D)$ outputs 1 if $D = C^a$ and 0 otherwise.

**Lemma 1.** *Let* $\mathsf{KE_{DH}}$ *be the DH key exchange protocol as in Fig.* 8. *Then* $\mathsf{KE_{DH}}$ *has* $\alpha = \log_2 p$ *bits of min-entropy, and for every adversary* $\mathcal{A}$ *that breaks the* $(t, \varepsilon, \mu, S, Q_V)$-OW-HV-*security of* $\mathsf{KE_{DH}}$, *there is an adversary* $\mathcal{B}$ *that breaks the* $(t', \varepsilon', Q_{\mathrm{DH}})$-StCDH

```
$\mathsf{Init_I}(i, r)$:
01 $\mathrm{st} := x \xleftarrow{\$} \mathbb{Z}_p$
02 $X := g^x$
03 return $(X, \mathrm{st})$

$\mathsf{Der_R}(r, i, X \in \mathbb{G})$
04 $y \xleftarrow{\$} \mathbb{Z}_p$
05 $Y := g^y$
06 $K := X^y$
07 return $(Y, K)$

$\mathsf{Der_I}(i, r, Y \in \mathbb{G}, \mathrm{st} \in \mathbb{Z}_p)$
08 $K := Y^{\mathrm{st}}$
09 return $K$
```

**Fig. 8.** The Diffie–Hellman key exchange protocol, $\mathsf{KE_{DH}}$, in our syntax definition .

```
$\mathcal{B}^{\mathrm{DH}_a}(A, B)$
01 $(\mathrm{sID}^*, K^*) \xleftarrow{\$} \mathcal{A}^{\mathrm{O}}(\mu)$
02 if $\mathrm{sID}^* > \mathrm{cnt_S}$ or $\mathrm{KVer}(\mathrm{sID}^*, K^*) = 0$
03     return 0
04 else
05     $(X, Y) := (\mathsf{Msg_I}[\mathrm{sID}^*], \mathsf{Msg_R}[\mathrm{sID}^*])$
06     fetch $\mathrm{sID}_1$ s.t. $\mathrm{type}[\mathrm{sID}_1] =$ "In" and $\mathsf{Msg_I}[\mathrm{sID}_1] = X$
07     fetch $\mathrm{sID}_1$ s.t. $\mathrm{type}[\mathrm{sID}_2] =$ "Re" and $\mathsf{Msg_R}[\mathrm{sID}_2] = Y$
08     $Z := K^* / (Y^{\alpha[\mathrm{sID}_1]} \cdot A^{\alpha[\mathrm{sID}_2]})$
09     return $[\![Z \in \mathsf{Win_{StCDH}}]\!]$          //break StCDH

$\mathrm{KVer}(\mathrm{sID}, K)$
10 $(X, Y) := (\mathsf{Msg_I}[\mathrm{sID}], \mathsf{Msg_R}[\mathrm{sID}])$
11 fetch $\mathrm{sID}_1$ s.t. $\mathrm{type}[\mathrm{sID}_1] =$ "In" and $\mathsf{Msg_I}[\mathrm{sID}_1] = X$
12 fetch $\mathrm{sID}_2$ s.t. $\mathrm{type}[\mathrm{sID}_2] =$ "Re" and $\mathsf{Msg_R}[\mathrm{sID}_2] = Y$
13 if $\mathrm{sID}_1 = \perp$ or $\mathrm{sID}_2 = \perp$
14     return $\perp$
15 return $\mathrm{DH}_a(Y, K/Y^{\alpha[\mathrm{sID}_1]})$

$\mathrm{Der_I}(\mathrm{sID}, Y)$
16 if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$ or $\mathrm{type}[\mathrm{sID}] \neq$ "In"
17     return $\perp$
18 if $\forall \mathrm{sID}' \in [\mathrm{cnt_S}] : \mathsf{Msg_R}[\mathrm{sID}'] \neq Y$
19     return $\perp$          //$Y$ is not honest
20 return $\epsilon$

$\mathrm{Session_I}((i, r) \in [\mu]^2)$          //$i \neq r$
21 $\mathrm{cnt_S}$ ++
22 $\mathrm{sID} := \mathrm{cnt_S}$
23 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i, r)$
24 $\mathrm{type}[\mathrm{sID}] :=$ "In"
25 $\alpha[\mathrm{sID}] \xleftarrow{\$} \mathbb{Z}_p$
26 $X := A \cdot g^{\alpha[\mathrm{sID}]}$
27 $(\mathsf{Msg_I}[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) := (X, \perp)$
28 return $(\mathrm{sID}, X)$

$\mathrm{Session_R}((i, r) \in [\mu]^2, X)$          //$i \neq r$
29 if $\forall \mathrm{sID} \in [\mathrm{cnt_S}] : \mathsf{Msg_I}[\mathrm{sID}] \neq X$
30     return $\perp$          //$X$ is not honest
31 $\mathrm{cnt_S}$ ++
32 $\mathrm{sID}' := \mathrm{cnt_S}$
33 $(\mathrm{init}[\mathrm{sID}'], \mathrm{resp}[\mathrm{sID}']) := (i, r)$
34 $\mathrm{type}[\mathrm{sID}'] :=$ "Re"
35 $\mathsf{Msg_I}[\mathrm{sID}'] := X$
36 $\alpha[\mathrm{sID}'] \xleftarrow{\$} \mathbb{Z}_p$
37 $Y := B \cdot g^{\alpha[\mathrm{sID}']}$
38 $\mathsf{Msg_R}[\mathrm{sID}'] := Y$
39 return $(\mathrm{sID}', Y)$
```

**Fig. 9.** Reduction $\mathcal{B}$ that breaks the StCDH assumption and simulates the OW-HV game for $\mathcal{A}$, when $A = g^a$ and $B = g^b$ for some unknown $a$ and $b$ .

*assumption with*

$$\varepsilon' = \varepsilon, \quad t' \approx t, \quad \textit{and} \quad Q_{\mathrm{DH}} = Q_V + 1. \tag{1}$$

*Proof.* The min-entropy assertion is straightforward, as the DH protocol generates messages by drawing exponents $x, y \leftarrow_\$ \mathbb{Z}_p$ uniformly as random.

We prove the rest of the lemma by constructing a reduction $\mathcal{B}$ which inputs the StCDH challenge $(A, B)$ and is given access to the decisional oracle $\mathrm{DH}_a$. $\mathcal{B}$ simulates the OW-HV security game for the adversary $\mathcal{A}$, namely answers $\mathcal{A}$'s oracle access as in Fig. 9. More precisely, $\mathcal{B}$ uses the random self-reducibility of StCDH to simulate the whole security game, instead of using the $\mathsf{Init_I}$ and $\mathsf{Der_R}$ algorithms. The most relevant codes are highlighted with **bold** line numbers.

We show that $\mathcal{B}$ simulates the OW-HV game for $\mathcal{A}$ perfectly:

- Since $X$ generated in line 26 and $Y$ generated in line 37 are uniformly random, the outputs of $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathsf{R}}$ are distributed as in the real protocol. Note that the output of $\textsc{Der}_{\mathrm{I}}$ does not get modified.
- For $\textsc{KVer}(\mathrm{sID}, K)$, if $K$ is a valid key that corresponds to session sID, then there must exist sessions $\mathrm{sID}_1$ and $\mathrm{sID}_2$ such that $\mathrm{type}[\mathrm{sID}_1] =$ "In" (defined in line 24) and $\mathrm{type}[\mathrm{sID}_2] =$ "Re" (defined in line 34) and

$$K = (B \cdot g^{\alpha[\mathrm{sID}_2]})^{(a+\alpha[\mathrm{sID}_1])} = Y^a \cdot Y^{\alpha[\mathrm{sID}_1]}. \tag{2}$$

  where $\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}] = \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}_1] = A \cdot g^{\alpha[\mathrm{sID}_1]}$ (defined in line 26) and $\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}] = \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}_2] = Y := B \cdot g^{\alpha[\mathrm{sID}_2]}$ (defined in line 37). Thus, the output of $\textsc{KVer}(\mathrm{sID}, K)$ is the same as that of $\mathrm{DH}_a(Y, K/Y^{\alpha[\mathrm{sID}_1]})$.

Finally, $\mathcal{A}$ returns $\mathrm{sID}^* \in [\mathrm{cnt}_{\mathsf{S}}]$ and a key $K^*$. If $\mathcal{A}$ wins, then $\textsc{KVer}(\mathrm{sID}^*, K^*) = 1$ which means that there exists sessions $\mathrm{sID}_1$ and $\mathrm{sID}_2$ such that $\mathrm{type}[\mathrm{sID}_1] =$ "In", $\mathrm{type}[\mathrm{sID}_2] =$ "Re" and

$$\begin{aligned} K^* &= g^{(a+\alpha[\mathrm{sID}_1])(b+\alpha[\mathrm{sID}_2])} = g^{ab} \cdot A^{\alpha[\mathrm{sID}_2]} \cdot B^{\alpha[\mathrm{sID}_1]} g^{\alpha[\mathrm{sID}_1]\alpha[\mathrm{sID}_2]} \\ &= g^{ab} \cdot A^{\alpha[\mathrm{sID}_2]} \cdot Y^{\alpha[\mathrm{sID}_1]}, \end{aligned}$$

where $Y = \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}_2] = B \cdot g^{\alpha[\mathrm{sID}_2]}$. This means $\mathcal{B}$ breaks the $\mathsf{StCDH}$ with $g^{ab} = K^*/(Y^{\alpha[\mathrm{sID}_1]} \cdot A^{\alpha[\mathrm{sID}_2]})$ as in line 08, if $\mathcal{A}$ break the $\mathsf{OW\text{-}HV}$ of $\mathsf{KE}_{\mathsf{DH}}$. Hence, $\varepsilon = \varepsilon'$. The running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ plus one exponentiation for every call to $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathsf{R}}$, so we get $t \approx t'$. The number of calls to $\mathrm{DH}_a$ is the number of calls to $\textsc{KVer}$, plus one additional call to verify the adversary's forgery, and hence $Q_{\mathrm{DH}} = Q_V + 1$. □

*Group of Signed Quadratic Residues* Our construction of a key exchange protocol in Fig. 8 is based on the $\mathsf{StCDH}$ assumption over a prime order group. Alternatively, we can instantiate our $\mathsf{VKE}$ protocol in a group of signed quadratic residues $\mathbb{QR}_N^+$ [27]. As the $\mathsf{StCDH}$ assumption in $\mathbb{QR}_N^+$ groups is tightly implied by the factoring assumption (by [27, Theorem 2]), our $\mathsf{VKE}$ protocol is secure based on the classical factoring assumption.

## 5. Signed Diffie–Hellman, revisited

Following the definition in Sect. 3, we want to construct a $\mathsf{IND\text{-}FS}$-secure authenticated key exchange protocol $\mathsf{AKE} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{I}}, \mathsf{Der}_{\mathsf{R}})$ by combining a $\mathsf{StCorrCMA}$-secure signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, a $\mathsf{OW\text{-}HV}$-secure key exchange protocol $\mathsf{KE} = (\mathsf{Init}_{\mathrm{I}}', \mathsf{Der}_{\mathrm{I}}', \mathsf{Der}_{\mathsf{R}}')$, and a random oracle $\mathsf{H}$. The construction is given in Fig. 10, and follow the execution order from Fig. 4.

We now prove that this construction is in fact a secure $\mathsf{AKE}$ protocol.

**Theorem 2.** *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathsf{H}}, Q_{\mathrm{COR}})$-$\mathsf{IND\text{-}FS}$-security of a protocol $\mathsf{AKE}$ constructed as in Fig. 10, we can construct an adversary $\mathcal{B}$ against the $(t', \varepsilon', \mu, Q_s, Q_{\mathrm{COR}}')$-$\mathsf{StCorrCMA}$-security of a signature scheme $\mathsf{SIG}$ with*

```
$\mathsf{Gen}_{\mathsf{AKE}}(\mathsf{par})$:
    $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{par})$
    return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Der}_{\mathsf{R}}(\mathsf{sk}_r, \mathsf{pk}_i, X, \sigma_i)$
    if $\mathsf{Ver}(\mathsf{pk}_i, X, \sigma_i) = 0$
        return $\bot$
    $(Y, K^*) \leftarrow \mathsf{Der}_{\mathsf{R}}'(r, i, X)$
    $\sigma_r \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_r, (X, Y))$
    $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
    $K := \mathsf{H}(\mathrm{ctxt}, K^*)$
    return $((Y, \sigma_r), K)$
```

```
$\mathsf{Init}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r)$:
    $(X, \mathrm{st}) \xleftarrow{\$} \mathsf{Init}_{\mathrm{I}}'(i, r)$
    $\sigma_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, X)$
    return $(X, \mathrm{st}, \sigma_i)$

$\mathsf{Der}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r, Y, \sigma_r, \mathrm{st})$
    if $\mathsf{Ver}(\mathsf{pk}_r, (X, Y), \sigma_r) = 0$
        return $\bot$
    $K^* := \mathsf{Der}_{\mathrm{I}}'(i, r, Y, \mathrm{st})$
    $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
    $K := \mathsf{H}(\mathrm{ctxt}, K^*)$
    return $K$
```

**Fig. 10.** Generic construction of AKE from SIG, KE and a random oracle H .

*$\alpha$ bits of key min-entropy, and an adversary $\mathcal{C}$ against the $(t'', \varepsilon'', \mu, S', Q_V)$-*OW-HV *security of a key exchange protocol* KE *with $\beta$ bits of min-entropy, such that*

$$\begin{array}{l} \varepsilon \leq 2\varepsilon' + \dfrac{\varepsilon''}{2} + \dfrac{\mu^2}{2^{\alpha+1}} + \dfrac{S^2}{2^{\beta+1}} \\ t' \approx t, \quad Q_s \leq S, \quad Q'_{\mathrm{COR}} = Q_{\mathrm{COR}} \\ t'' \approx t, \quad S' = S, \quad Q_V \leq Q_{\mathsf{H}}. \end{array} \tag{3}$$

*Proof.* We will prove this by using the following hybrid games, which are illustrated in Fig. 11.

Game $G_0$: This is the IND-FS security game for the protocol AKE. We assume that all long-term keys, and all messages output by $\mathsf{Init}_{\mathrm{I}}$ and $\mathsf{Der}_{\mathsf{R}}$ are distinct. If a collision happens, the game aborts. To bound the probability of this happening, we use that SIG has $\alpha$ bits of key min-entropy, and KE has $\beta$ bits of min-entropy. We can upper bound the probability of a collision happening in the keys as $\mu^2/2^{\alpha+1}$ for $\mu$ parties, and the probability of a collision happening in the messages as $S^2/2^{\beta+1}$ for $S$ sessions, as each session computes one message. Thus, we have

$$\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}} \Rightarrow 1] \leq \Pr[G_0^{\mathcal{A}} \Rightarrow 1] + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}}. \tag{4}$$

Game $G_1$: In this game, when the oracles $\textsc{Der}_{\mathrm{I}}$ and $\textsc{Session}_{\mathsf{R}}$ try to derive a session key, they will abort if the input message does not correspond to a previously sent message, and the corresponding signature is valid *w.r.t.* an uncorrupted party (namely, $\mathcal{A}$ generates the message itself).

This step is to exclude the active attacks where an adversary creates its own messages. An adversary cannot notice this change, since it requires the adversary to forge a signature on the underlying St-UF-CMA secure signature scheme. Later on, we will formally prove this. Moreover, this is the preparation step for reducing an IND-FS adversary of AKE to an OW-HV adversary of KE. Note that in this game we do not exclude all the non-matching Test sessions, but it is already enough for the "IND-FS-to-OW-HV" reduction. For instance, $\mathcal{A}$ can still force some responder session to be non-matching by

```
GAMES $G_0$-$G_2$
01 $\mathrm{cnt_S} := 0$                                   //session counter
02 for $n \in [\mu]$
03     $(\mathsf{pk}_n, \mathsf{sk}_n) \xleftarrow{\$} \mathsf{Gen_{AKE}}$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
06 for $\mathrm{sID}^* \in \mathcal{S}$
07     if $\mathrm{FRESH}(\mathrm{sID}^*) = \mathbf{false}$
08         return $b$
09     if $\mathrm{VALID}(\mathrm{sID}^*) = \mathbf{false}$
10         return $b$
11 return $[\![b = b']\!]$

$\mathrm{SESSION_R}((i,r) \in [\mu]^2, (X, \sigma_i))$
12 $\mathrm{cnt_S}$ ++
13 $\mathrm{sID} := \mathrm{cnt_S}$
14 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i, r)$
15 $\mathrm{type}[\mathrm{sID}] :=$ "Re"
16 $\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[i]$
17 $((Y, \sigma_r), K) \xleftarrow{\$} \mathsf{Der_R}(\mathsf{sk}_r, \mathsf{pk}_i, (X, \sigma_i))$
18 if $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$ and
   $\nexists \mathrm{sID}'$: $(\mathrm{init}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], \mathsf{Msg_I}[\mathrm{sID}'])$
   $= (i, \text{"In"}, (X, \sigma_i))$                      //$G_{1\text{-}2}$
19     $\mathsf{AbortDer_R} := \mathbf{true}$                 //$G_{1\text{-}2}$
20     abort                                                //$G_{1\text{-}2}$
21 $(\mathsf{Msg_I}[\mathrm{sID}], \mathsf{Msg_R}[\mathrm{sID}]) := ((X, \sigma_i), (Y, \sigma_r))$
22 $\mathrm{sKey}[\mathrm{sID}] := K$
23 return $(\mathrm{sID}, (Y, \sigma_r))$

$\mathrm{SESSION_I}((i,r) \in [\mu]^2)$
24 $\mathrm{cnt_S}$ ++
25 $\mathrm{sID} := \mathrm{cnt_S}$
26 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i, r)$
27 $\mathrm{type}[\mathrm{sID}] :=$ "In"
28 $(X, \mathrm{st}, \sigma_i) \xleftarrow{\$} \mathsf{Init_I}(\mathsf{sk}_i, \mathsf{pk}_r)$
29 $(\mathsf{Msg_I}[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) := ((X, \sigma_i), \mathrm{st})$
30 return $(\mathrm{sID}, (X, \sigma_i))$

$\mathrm{DER_I}(\mathrm{sID}, (Y, \sigma_r))$
31 if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$ or $\mathrm{type}[\mathrm{sID}] \neq$ "In"
32     return $\perp$                                       //no re-use
33 $(i, r) := (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
34 $\mathrm{st} := \mathrm{state}[\mathrm{sID}]$
35 $\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[r]$
36 $K := \mathsf{Der_I}(\mathsf{sk}_i, \mathsf{pk}_r, Y, \sigma_r, \mathrm{st})$
37 $(X, \sigma_i) := \mathsf{Msg_I}[\mathrm{sID}]$
38 if $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$ and
   $\nexists \mathrm{sID}'$: $(\mathrm{resp}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], \mathsf{Msg_I}[\mathrm{sID}'], \mathsf{Msg_R}[\mathrm{sID}'])$
   $= (r, \text{"Re"}, (X, \sigma_i), (Y, \sigma_r))$          //$G_{1\text{-}2}$
39     $\mathsf{AbortDer_I} := \mathbf{true}$                 //$G_{1\text{-}2}$
40     abort                                                //$G_{1\text{-}2}$
41 $(\mathsf{Msg_R}[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) := ((Y, \sigma_r), K)$
42 return $\epsilon$

$\mathrm{TEST}(\mathrm{sID})$
43 if $\mathrm{sID} \in \mathcal{S}$ return $\perp$            //already tested
44 if $\mathrm{sKey}[\mathrm{sID}] = \perp$ return $\perp$
45 $\mathcal{S} := \mathcal{S} \cup \{\mathrm{sID}\}$
46 $K_0^* := \mathrm{sKey}[\mathrm{sID}]$                      //$G_{0\text{-}1}$
47 $K_0^* \xleftarrow{\$} \mathcal{K}$                         //$G_2$
48 $K_1^* \xleftarrow{\$} \mathcal{K}$
49 return $K_b^*$
```

**Fig. 11.** Games $G_0$-$G_2$. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\mathrm{SESSION_I}, \mathrm{SESSION_R}, \mathrm{DER_I}, \mathrm{REVEAL}, \mathrm{CORR}, \mathrm{TEST}\}$, where REVEAL and CORR are simulated as in the original IND-FS game in Fig. 5. Game $G_0$ implicitly assumes that there is no collision between long term keys or messages output by the experiment .

reusing some of the previous initiator messages to query $\mathrm{SESSION_R}$, and then $\mathcal{A}$ uses the non-matching responder session to query TEST.

The only way to distinguish $G_0$ and $G_1$ is to trigger the new abort event in either line 19 (i.e., $\mathsf{AbortDer_R}$) or line 39 (i.e., $\mathsf{AbortDer_I}$) of Fig. 11. We define the event $\mathsf{AbortDer} := \mathsf{AbortDer_I} \vee \mathsf{AbortDer_R}$ and have that

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{AbortDer}].$$

To bound this probability, we construct an adversary $\mathcal{B}$ against the $(t', \varepsilon', \mu, Q_s, Q'_{\mathrm{COR}})$-StCorrCMA-security of SIG in Fig. 12.

We note that AbortDer is **true** only if $\mathcal{A}$ performs attacks 5+6 in Table 2 which may lead to a session without any matching session. If $\mathsf{AbortDer} = \mathbf{true}$ then $\Sigma$ is defined in lines 26 and 42 of Fig. 12 and $\Sigma$ is a valid StCorrCMA forge for SIG. We only show that for the case when $\mathsf{AbortDer_R} = \mathbf{true}$ here, and the argument is similar for the case when $\mathsf{AbortDer_I} = \mathbf{true}$. Given that $\mathsf{AbortDer_R}$ happens, we have that $\mathsf{Ver}(\mathsf{pk}_i, X, \sigma_i) = 1$ and $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$. Due to the criteria in line 40, the pair $(X, \sigma_i)$ has not been output by $\mathrm{SESSION_I}$ on input $(i, r)$ for any $r$, and hence $(i, X)$ has never been queried to the $\mathrm{SIGN}'$ oracle. Therefore, $\mathcal{B}$ aborts $\mathcal{A}$ in the IND-FS game and returns $(i, X, \sigma_i)$ to

```
$\mathcal{B}^{\textsc{Corr}', \textsc{Sign}'}(\mathsf{pk}_1, \dots, \mathsf{pk}_\mu)$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \dots, \mathsf{pk}_\mu)$
for $\mathrm{sID}^* \in \mathcal{S}$
    if $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$
        return $b$
    if $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
        return $b$
return $[\![\Sigma \in \mathsf{Win}_{\mathsf{StCorrCMA}}]\!]$        //break StCorrCMA

$\textsc{Session}_{\mathrm{I}}((i,r) \in [\mu]^2)$
$\mathrm{cnt}_{\mathrm{S}}$ ++
$\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
$(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i,r)$
$\mathrm{type}[\mathrm{sID}] :=$ “In”
$(X, \mathrm{st}) \xleftarrow{\$} \mathsf{Init}'_{\mathrm{I}}(i,r)$
$\sigma_i \xleftarrow{\$} \textsc{Sign}'(\mathsf{pk}_i, X)$
$(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) := ((X, \sigma_i), \mathrm{st})$
return $(\mathrm{sID}, (X, \sigma_i))$

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, (Y, \sigma_r))$
if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$ or $\mathrm{type}[\mathrm{sID}] \neq$ “In”
    return $\perp$        //no re-use
$(i,r) := (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
$\mathrm{st} := \mathrm{state}[\mathrm{sID}]$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[r]$
if $\mathsf{Ver}(\mathsf{pk}_r, (X,Y), \sigma_r) = 0$
    return $\perp$
if $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$ and
  $\nexists \mathrm{sID}'$: $(\mathrm{resp}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}'])$
    $= (r,$ “Re”$, (X, \sigma_i), (Y, \sigma_r))$
    $\mathsf{AbortDer}_{\mathsf{I}} := \mathbf{true}$
    $\Sigma := (r, (X,Y), \sigma_r)$        //valid forgery
    abort
$K^* := \mathsf{Der}'_{\mathrm{I}}(i, r, Y, \mathrm{st})$
$\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
$K := \mathsf{H}(\mathrm{ctxt}, K^*)$
$(\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) := ((Y, \sigma_r), K)$
return $\epsilon$

$\textsc{Session}_{\mathrm{R}}((i,r) \in [\mu]^2, (X, \sigma_i))$
$\mathrm{cnt}_{\mathrm{S}}$ ++
$\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
$(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i,r)$
$\mathrm{type}[\mathrm{sID}] :=$ “Re”
$\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[i]$
if $\mathsf{Ver}(\mathsf{pk}_i, X, \sigma_i) = 0$
    return $\perp$
if $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$ and
  $\nexists \mathrm{sID}'$: $(\mathrm{init}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'])$
    $= (i,$ “In”$, (X, \sigma_i))$
    $\mathsf{AbortDer}_{\mathsf{R}} := \mathbf{true}$
    $\Sigma := (i, X, \sigma_i)$        //valid forgery
    abort
$(Y, K^*) \xleftarrow{\$} \mathsf{Der}_{\mathsf{R}}'(r, i, X)$
$\sigma_r \xleftarrow{\$} \textsc{Sign}'(\mathsf{pk}_r, (X,Y))$
$\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
$K := \mathsf{H}(\mathrm{ctxt}, K^*)$
$(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]) := ((X, \sigma_i), (Y, \sigma_r))$
$\mathrm{sKey}[\mathrm{sID}] := K$
return $(\mathrm{sID}, (Y, \sigma_r))$

$\textsc{Corr}(n \in [\mu])$
$\mathrm{corrupted}[n] := \mathbf{true}$
$\mathsf{sk}_n \leftarrow \textsc{Corr}'(n)$
return $\mathsf{sk}_n$

$\mathsf{H}(\mathsf{pk}_i, \mathsf{pk}_r, X, Y, K^*)$
$\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, Y)$
if $\mathsf{H}[\mathrm{ctxt}, K^*] = K$
    return $K$
$K \xleftarrow{\$} \mathcal{K}$
$\mathsf{H}[\mathrm{ctxt}, K^*] := K$
return $K$
```

**Fig. 12.** Adversary $\mathcal{B}$ against the $(t', \varepsilon', \mu, Q_S, Q'_{\mathrm{COR}})$-StCorrCMA-security of SIG. The StCorrCMA game provides oracles $\textsc{Sign}', \textsc{Corr}'$. The adversary $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{H}\}$, where $\textsc{Reveal}$ and $\textsc{Test}$ remain the same as in Fig. 4. We highlight the most relevant codes with **bold** line numbers .

the StCorrCMA challenger to win the StCorrCMA game. Therefore, we have

$$\Pr[\mathsf{AbortDer}_{\mathsf{R}}] \leq \varepsilon', \tag{5}$$

which implies that

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{AbortDer}_{\mathsf{I}}] + \Pr[\mathsf{AbortDer}_{\mathsf{R}}] \leq 2\varepsilon'. \tag{6}$$

The running time of $\mathcal{B}$ is the same as that of $\mathcal{A}$, plus the time used to run the key exchange algorithms $\mathsf{Init}'_{\mathrm{I}}$, $\mathsf{Der}_{\mathsf{R}}'$, $\mathsf{Der}'_{\mathrm{I}}$ and the signature verification algorithm $\mathsf{Ver}$. This gives $t' \approx t$. For the number of signature queries, we have $Q_s \leq S$, since $\textsc{Session}_{\mathsf{R}}$ can abort before it queries the signature oracle, and the adversary can reuse messages output by $\textsc{Session}_{\mathrm{I}}$. For the number of corruptions, we have $Q'_{\mathrm{COR}} = Q_{\mathrm{COR}}$.

Game $G_2$: Intuitively, since in $G_1$ an adversary $\mathcal{A}$ is not allowed to create its own message to attack the protocol, $\mathcal{A}$ can only use the honestly generated messages, but it may forward these messages in an different order. The $\mathsf{OW\text{-}HV}$ security of the underlying $\mathsf{KE}$ allows us to tightly prove that such an $\mathcal{A}$ cannot distinguish a real session key from a random one, which conclude our security proof. To formally prove it, in $G_2$, Test oracle always returns a uniformly random key, independent on the bit $b$ (Fig. 13).

Since we have excluded collisions in the messages output by the experiment, it is impossible to create two sessions of the same type that compute the same session key. Hence, an adversary must query the random oracle $\mathsf{H}$ on the correct input of a test session to detect the change between $G_1$ and $G_2$ (which is only in case $b = 0$). More precisely, we have $\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 1] = \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 1]$ and

$$\begin{aligned}\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| &= \frac{1}{2}\Big|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] + \Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 1] \\ &\quad - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 1]\Big| \\ &= \frac{1}{2}\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right|. \qquad (7)\end{aligned}$$

To bound Eq. (7), we construct an adversary $\mathcal{C}$ to $(t'', \varepsilon'', \mu, S', Q_V)$-break the $\mathsf{OW\text{-}HV}$ security of $\mathsf{KE}$. The input to $\mathcal{C}$ is the number of parties $\mu$, and system parameters $\mathsf{par}$. In addition, $\mathcal{C}$ has access to oracles $\textsc{Session}'_{\mathrm{I}}$, $\textsc{Session}'_{\mathsf{R}}$, $\textsc{Der}'_{\mathrm{I}}$ and KVer.

We firstly show that the outputs of $\textsc{Session}_{\mathrm{I}}$, $\textsc{Session}_{\mathsf{R}}$ and $\textsc{Der}_{\mathrm{I}}$ (simulated by $\mathcal{C}$) are distributed the same as in $G_1$. Due to the abort conditions introduced in $G_1$, for all sessions that has finished computing a key without making the game abort, their messages are honestly generated, although they may be in a different order and there are non-matching sessions. Hence, $\textsc{Session}_{\mathrm{I}}$, $\textsc{Session}_{\mathsf{R}}$ and $\textsc{Der}_{\mathrm{I}}$ can be perfectly simulated using $\textsc{Session}'_{\mathrm{I}}$, $\textsc{Session}'_{\mathsf{R}}$ and $\textsc{Der}'_{\mathrm{I}}$ of the $\mathsf{OW\text{-}HV}$ game and the signing key.

It is also easy to see that the random oracle $\mathsf{H}$ simulated by $\mathcal{C}$ has the same output distribution as in $G_1$. We stress that if line 66 is executed then adversary $\mathcal{A}$ may use the sID to distinguish $G_2$ and $G_1$ for $b = 0$, which is the only case for $\mathcal{A}$ to see the difference. At the same time, we obtain a valid attack $\Sigma := (\mathrm{sID}, K^*)$ for the $\mathsf{OW\text{-}HV}$ security. Thus, we have

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

As before, the running time of $\mathcal{C}$ is that of $\mathcal{A}$, plus generating and verifying signatures, and we have $t'' \approx t$. Furthermore, $S' = S$, as the counter for the $\mathsf{OW\text{-}HV}$ game increases once for every call to $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathsf{R}}$.

At last, for game $G_2$ we have $\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2}$, as the response from the Test oracle is independent of the bit $b$. Summing up all the equations, we obtain

$$\begin{aligned}\varepsilon \leq \left|\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right| &\leq \left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}} - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \\ &= \left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1] + \Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1] + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}}\right|\end{aligned}$$

```
$\mathcal{C}^{\mathrm{O}'}(\mu)$
01 for $n \in [\mu]$
02     $(\mathsf{pk}_n, \mathsf{sk}_n) \xleftarrow{\$} \mathsf{Gen}(\mathsf{par})$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\mu)$
05 for $\mathrm{sID}^* \in \mathcal{S}$
06     if $\textsc{Fresh}(\mathrm{sID}^*) =$ false
07         return $b$
08     if $\textsc{Valid}(\mathrm{sID}^*) =$ false
09         return $b$
10 return $[\![\Sigma \in \mathsf{Win}_{\mathsf{OW\text{-}HV}}]\!]$

$\textsc{Session}_{\mathrm{I}}((i,r) \in [\mu]^2)$
11 $(\mathrm{sID}, X) \xleftarrow{\$} \textsc{Session}_{\mathrm{I}}'(i,r)$
12 $\mathrm{cnt}_{\mathrm{S}}$++
13 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i,r)$
14 $\mathrm{type}[\mathrm{sID}] :=$ "In"
15 $\sigma_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, X)$
16 $\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}] := (X, \sigma_i)$
17 return $(\mathrm{sID}, (X, \sigma_i))$

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, (Y, \sigma_r))$
18 if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$ or $\mathrm{type}[\mathrm{sID}] \neq$ "In"
19     return $\perp$                                   //no re-use
20 $(i,r) := (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
21 $\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[r]$
22 $(X, \sigma_i) := \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}]$
23 if $\mathsf{Ver}(\mathsf{pk}_r, (X,Y), \sigma_r) = 0$
24     return $\perp$
25 if $\mathrm{peerCorrupted}[\mathrm{sID}] =$ false and
   $\nexists \mathrm{sID}'$: $(\mathrm{resp}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}'])$
    $= (r, \text{"Re"}, (X, \sigma_i), (Y, \sigma_r))$
26     abort
27 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
28 $\textsc{Der}_{\mathrm{I}}'(\mathrm{sID}, Y)$
29 if $\exists K^* : \mathsf{H}[\mathrm{ctxt}, K^*, 1] = K$
30     $\mathrm{sKey}[\mathrm{sID}] := K$
31 elseif $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] = K$
32     $\mathrm{sKey}[\mathrm{sID}] := K$
33 else $K \xleftarrow{\$} \mathcal{K}$
34     $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] := K$
35     $\mathrm{sKey}[\mathrm{sID}] := K$
36 $\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}] := (Y, \sigma_r)$
37 return $\epsilon$

$\textsc{Session}_{\mathrm{R}}((i,r) \in [\mu]^2, (X, \sigma_i))$
38 if $\mathsf{Ver}(\mathsf{pk}_i, X, \sigma_i) = 0$
39     return $\perp$
40 $(\mathrm{sID}, Y) \xleftarrow{\$} \textsc{Session}_{\mathrm{R}}'(i, r, X)$
41 $\mathrm{cnt}_{\mathrm{S}}$++
42 $\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[i]$
43 if $\mathrm{peerCorrupted}[\mathrm{sID}] =$ false and
   $\nexists \mathrm{sID}'$: $(\mathrm{init}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'])$
    $= (i, \text{"In"}, (X, \sigma_i))$
44     abort
45 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) := (i,r)$
46 $\mathrm{type}[\mathrm{sID}] :=$ "Re"
47 $\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}] := (X, \sigma_i)$
48 $\sigma_r \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_r, (X, Y))$
49 $\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}] := (Y, \sigma_r)$
50 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
51 if $\exists K^* : \mathsf{H}[\mathrm{ctxt}, K^*, 1] = K$
52     $\mathrm{sKey}[\mathrm{sID}] := K$
53 elseif $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] = K$
54     $\mathrm{sKey}[\mathrm{sID}] := K$
55 else $K \xleftarrow{\$} \mathcal{K}$
56     $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] := K$
57     $\mathrm{sKey}[\mathrm{sID}] := K$
58 return $(Y, \sigma_r)$

$\mathsf{H}(\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r, K^*)$
59 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, X, \sigma_i, Y, \sigma_r)$
60 if $\mathsf{H}[\mathrm{ctxt}, K^*, \cdot] = K$
61     return $K$
62 $h := \perp$
63 if $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] = K$ and $\exists \mathrm{sID}$:
       $(\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]) = ((X, \sigma_i), (Y, \sigma_r))$
64     $\textsc{Der}_{\mathrm{I}}'(\mathrm{sID}, Y)$
65     if $\textsc{KVer}(\mathrm{sID}, K^*) = 1$
66         $\Sigma := (\mathrm{sID}, K^*)$                //attack for OW-HV
67         replace $(\perp, \perp)$ in $\mathsf{H}[\mathrm{ctxt}, \perp, \perp]$
               with $(K^*, 1)$
68         return $K$
69     else $h := 0$
70 $K \xleftarrow{\$} \mathcal{K}$
71 $\mathsf{H}[\mathrm{ctxt}, K^*, h] := K$
72 return $K$
```

**Fig. 13.** Reduction $\mathcal{C}$ against the $(t'', \varepsilon'', \mu, S', Q_V)$-OW-HV-security of KE. The OW-HV game provides oracles $\mathrm{O}' := \{\textsc{Session}_{\mathrm{I}}', \textsc{Session}_{\mathrm{R}}', \textsc{Der}_{\mathrm{I}}', \textsc{KVer}\}$. The adversary $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{H}\}$, where Reveal, Corr and Test are defined as in $G_2$ of Fig. 11. We highlight the most relevant codes with **bold** line numbers. The center dot '$\cdot$' in this figure means arbitrary value .

$$\leq \left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| + \left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}}$$

$$\leq 2\varepsilon' + \frac{\varepsilon''}{2} + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}},$$

and $t' \approx t, \quad Q_s \leq S, \quad Q'_{\mathrm{COR}} = Q_{\mathrm{COR}}, \quad t'' \approx t, \quad S' = S, \quad Q_V \leq Q_{\mathsf{H}}$.

□

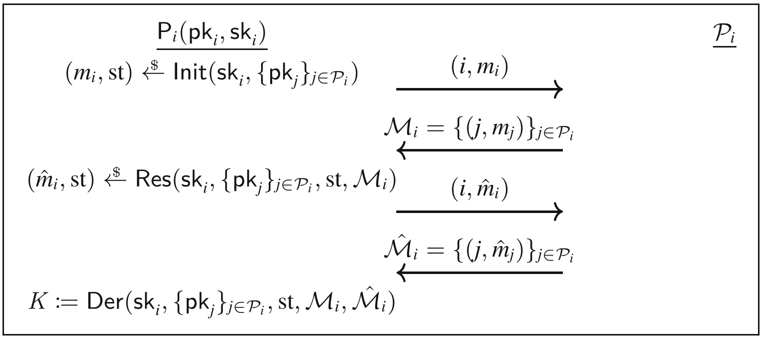
$\mathsf{P}_i(\mathsf{pk}_i, \mathsf{sk}_i)$
$\mathcal{P}_i$
$(m_i, \mathrm{st}) \overset{\$}{\leftarrow} \mathsf{Init}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i})$
$(i, m_i)$
$\mathcal{M}_i = \{(j, m_j)\}_{j \in \mathcal{P}_i}$
$(\hat{m}_i, \mathrm{st}) \overset{\$}{\leftarrow} \mathsf{Res}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i}, \mathrm{st}, \mathcal{M}_i)$
$(i, \hat{m}_i)$
$\hat{\mathcal{M}}_i = \{(j, \hat{m}_j)\}_{j \in \mathcal{P}_i}$
$K := \mathsf{Der}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i}, \mathrm{st}, \mathcal{M}_i, \hat{\mathcal{M}}_i)$

**Fig. 14.** Illustration of running a group authenticated key exchange from party $\mathsf{P}_i$'s point of view. All messages are broadcast to all parties, and every party runs all the algorithms .

## 6. An Extension: Tightly Secure Group Authenticated Key Exchange

### 6.1. *Security Model for Group Authenticated Key Exchange*

We consider two-round broadcast group authenticated key exchange protocols that are executed interactively between $\mu > 2$ parties. Each round corresponds to a messages broadcast. Formally, it is defined as $\mathsf{GAKE} = (\mathsf{Gen}_{\mathsf{GAKE}}, \mathsf{Init}, \mathsf{Res}, \mathsf{Der})$ consisting of four algorithms. It is visualized as in Fig. 14. We denote the set of potential participants by $\mathsf{P} = (\mathsf{P}_1, \ldots, \mathsf{P}_\mu)$. Before the protocol is executed for the first time, each party $\mathsf{P}_i \in \mathsf{P}$ runs the algorithm $\mathsf{Gen}_{\mathsf{GAKE}}(\mathsf{par})$ to generate his own long-term public and private keys $(\mathsf{pk}_i, \mathsf{sk}_i)$.

Our two-round $\mathsf{GAKE}$ protocol allows all parties in a group $\mathcal{Q} \subseteq \mathsf{P}$ to establish a common secret key. For a party $\mathsf{P}_i$, we say that $\mathcal{P}_i$ is the rest of the group from $\mathsf{P}_i$'s view, and we can write $\mathcal{Q} = \{\mathsf{P}_i\} \cup \mathcal{P}_i$. By a slight abuse of notation, we will often write $j \in \mathcal{P}_i$ instead of $\mathsf{P}_j \in \mathcal{P}_i$.

In the first round, each party $\mathsf{P}_i \in \mathcal{Q}$ starts the session sID by executing the initialization algorithm $\mathsf{Init}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i})$ which outputs a message $m_i$ and a state st. The party $\mathsf{P}_i$ broadcasts $(i, m_i)$ and keeps the internal state st.

In the second round, let $\mathcal{M}_i$ denote the set of all pairs $(j, m_j)$ received by $\mathsf{P}_i$ in the first round. Then, each party $\mathsf{P}_i \in \mathcal{Q}$ executes the response algorithm $\mathsf{Res}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i}, \mathrm{st}, \mathcal{M}_i)$ to compute a message $\hat{m}_i$ and an updated state st. As in the first round, $\mathsf{P}_i$ broadcasts $(i, \hat{m}_i)$ and keeps the state st.

In the final phase, let $\hat{\mathcal{M}}_i$ denote the set of all pairs $(j, \hat{m}_j)$ received by party $\mathsf{P}_i$ in the second round. To obtain the common group session key, each party $\mathsf{P}_i$ can execute $\mathsf{Der}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i}, \mathrm{st}, \mathcal{M}_i, \hat{\mathcal{M}}_i)$ which outputs the key $K$. An illustration is given in Fig. 14.

Similar to our two-party key exchange protocol, our security game is written in pseudocode. In our model, $\mathsf{GAKE}$ achieves forward secrecy and has both the explicit authentication and implicit key confirmation properties. In the group key exchange setting, explicit authentication means entity authentication for every message transmitted in the sense that each party can explicitly confirm that the initial message is issued by the actual owner of the associated public key. Moreover, the key confirmation property is also implicit for our $\mathsf{GAKE}$, where every party in a group $\mathcal{Q}$ is assured implicitly that

```
GAME IND-G-FS
for $n \in [\mu]$
    $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{GAKE}}(\mathsf{par})$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
for $\mathrm{sID}^* \in \mathcal{S}$:
    if $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$
        return 0                                    //session not fresh
    if $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
        return 0                                    //no valid attack
return $[\![b = b']\!]$

$\textsc{Session}_{\mathsf{I}}(i \in [\mu], \mathcal{P}_i \subseteq [\mu])$
$\mathrm{cnt}_{\mathrm{S}}$ ++
$\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
$\mathrm{owner}[\mathrm{sID}] := i$
$\mathrm{peer}[\mathrm{sID}] := \mathcal{P}_i$
$\mathcal{Q}[\mathrm{sID}] := \mathrm{peer}[\mathrm{sID}] \cup \{i\}$
$(m_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Init}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i})$
$\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}] := (i, m_i)$
$\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
return $(\mathrm{sID}, m_i)$

$\textsc{Session}_{\mathsf{R}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], \mathcal{M}_i)$
$(i, \mathcal{P}_i) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\mathcal{M}_i| \neq |\mathcal{P}_i|$
    return $\perp$                                  //all peers must have broadcasted
$\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}_i} \mathrm{corrupted}[j]$
$\mathcal{M}[\mathrm{sID}] := \mathcal{M}_i$
$(\hat{m}_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Res}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i}, \mathrm{state}[\mathrm{sID}], \mathcal{M}_i)$
$\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}] := (i, \hat{m}_i)$
$\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
return $\hat{m}_i$

$\textsc{Der}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], \hat{\mathcal{M}}_i)$
if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$
    return $\perp$
$(i, \mathcal{P}_i) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\hat{\mathcal{M}}_i| \neq |\mathcal{P}_i|$ return $\perp$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}_i} \mathrm{corrupted}[j]$
$\hat{\mathcal{M}}[\mathrm{sID}] := \hat{\mathcal{M}}_i$
$\mathcal{M}_i := \mathcal{M}[\mathrm{sID}]$
$K := \mathsf{Der}(\mathsf{sk}_i, \{\mathsf{pk}_j\}_{j \in \mathcal{P}_i}, \mathrm{state}[\mathrm{sID}], \mathcal{M}_i, \hat{\mathcal{M}}_i)$
$\mathrm{sKey}[\mathrm{sID}] := K$
return $\epsilon$

$\textsc{Reveal}(\mathrm{sID})$
$\mathrm{revealed}[\mathrm{sID}] := \mathbf{true}$
return $\mathrm{sKey}[\mathrm{sID}]$

$\textsc{Corr}(n \in [\mu])$
$\mathrm{corrupted}[n] := \mathbf{true}$
return $\mathsf{sk}_n$

$\textsc{Test}(\mathrm{sID})$
if $\mathrm{sID} \in \mathcal{S}$ return $\perp$      //already tested
if $\mathrm{sKey}[\mathrm{sID}] = \perp$ return $\perp$
$\mathcal{S} := \mathcal{S} \cup \{\mathrm{sID}\}$
$K_0^* := \mathrm{sKey}[\mathrm{sID}]$
$K_1^* \xleftarrow{\$} \mathcal{K}$
return $K_b^*$
```

**Fig. 15.** Game IND-G-FS for GAKE. The number of messages in the set $\mathcal{M}_i$ is denoted by $|\mathcal{M}_i|$, and $|\mathcal{P}_i|$ denotes the number of parties in $\mathcal{P}_i$ .

all members of the group will have the same session key. The security game is given in Figs. 15 and 16. Our model can be viewed as a careful extension of our two-party model to $\mu$ parties. Moreover, we note that Poettering et al. [41] proposed a general framework for defining security of GAKE protocols. To the best of our knowledge, our model can be viewed as a specified use case of their framework. For instance, we do not consider Expose queries to reveal the local session-state.

Execution environment. We consider $\mu$ parties $\mathsf{P} = (\mathsf{P}_1, \ldots, \mathsf{P}_\mu)$ with long-term key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$, $i \in [\mu]$. For each group key exchange, each party in a group $\mathcal{Q}$ has their own session with a unique identification number sID, and variables which are defined relative to sID:

- $\mathrm{owner}[\mathrm{sID}] \in [\mu]$ denotes the owner of the session.
- $\mathrm{peer}[\mathrm{sID}] \subseteq [\mu]$ denotes the peers of the session.
- $\mathcal{Q}[\mathrm{sID}]$ denotes all the participants of the session.
- $\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}]$ denotes the message sent by the owner during the first round.
- $\mathcal{M}[\mathrm{sID}]$ denotes the messages received by the owner during the first round.
- $\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]$ denotes the message sent by the owner during the second round.
- $\hat{\mathcal{M}}[\mathrm{sID}]$ denotes the messages received by the owner during the second round.
- state[sID] denotes the (secret) state information *i.e.* ephemeral secret keys.

- sKey[sID] denotes the session key.

Adversary model. Similar to the $\mathsf{AKE}$ security notion, we do not allow the adversary to register adversarially controlled parties by providing long-term public keys, and the adversary has access to oracles Corr and Reveal as described in Fig. 15. We use the following Boolean values to store which queries the adversary made:

- corrupted[$i$] denotes whether the long-term secret key of party $\mathsf{P}_i$ was given to the adversary.
- revealed[sID] denotes whether the group session key was given to the adversary.
- peerCorrupted[sID] denotes whether one of the peers in the group was corrupted and its long-term key was given to the adversary at the time when the session key was derived.

Matching sessions. Extending the definition of matching sessions from the two-party case, we define matching sessions in the $\mathsf{GAKE}$ setting as follows.

- **Matching Sessions**: Two sessions $\mathrm{sID}_i$, $\mathrm{sID}_j$ are matching if:

$$\begin{aligned}
&\mathrm{owner}[\mathrm{sID}_i] \neq \mathrm{owner}[\mathrm{sID}_j] && \text{(Different owners)}\\
&\mathcal{Q}[\mathrm{sID}_i] = \mathcal{Q}[\mathrm{sID}_j] && \text{(Identical participants)}\\
&\{\mathsf{Msg}_\mathsf{I}[\mathrm{sID}_i]\} \cup \mathcal{M}[\mathrm{sID}_i] && \\
&\ = \{\mathsf{Msg}_\mathsf{I}[\mathrm{sID}_j]\} \cup \mathcal{M}[\mathrm{sID}_j] && \text{(Identical messages in the first round)}\\
&\{\mathsf{Msg}_\mathsf{R}[\mathrm{sID}_i]\} \cup \hat{\mathcal{M}}[\mathrm{sID}_i] && \\
&\ = \{\mathsf{Msg}_\mathsf{R}[\mathrm{sID}_j]\} \cup \hat{\mathcal{M}}[\mathrm{sID}_j] && \text{(Identical messages in the second round)}
\end{aligned}$$

As in the $\mathsf{AKE}$ setting, our protocols in the full $\mathsf{GAKE}$ model will use signatures, and hence any successful no-match attack as described in [35] will lead to a signature forgery.

Test session. The adversary is given access to the test oracle Test. This oracle can be queried multiple times and depending on a randomly chosen bit $b \leftarrow_\$ \{0, 1\}$ (which is shared between all test queries), it outputs either a uniformly random key, or the specified session key.

### 6.2. *Verifiable Group Key Exchange*

To achieve tight security, we extend the verifiable key exchange from the two-party setting to $\mu$-parties. As for the regular two party $\mathsf{AKE}$, we construct our tightly secure group authenticated key exchange based on a verifiable (non-authenticated) group key exchange (GKE) that has One-Wayness against Honest and key Verification attacks (aka. $\mathsf{OW}$-$\mathsf{G}$-$\mathsf{HV}$ security). As in the two-party case, the adversary can perform passive attacks, or forward messages in a different order to create non-matching sessions, and check if a key corresponds to some honestly generated transcripts. We require that all the involved messages must be honestly generated by the security game and not by the adversary. A (non-authenticated) group key exchange (GKE) protocol consists of a tuple of algorithms $\mathsf{GKE} := (\mathsf{Init}, \mathsf{Res}, \mathsf{Der})$, where parties do not hold any public or private key and $\mathsf{Init}$ algorithms now take users' identities $(i, \mathcal{P}_i)$ as input.

```
FRESH(sID*)
$(i^*, \mathcal{Q}^*) := (\text{owner}[\text{sID}^*], \mathcal{Q}[\text{sID}^*])$
$\mathfrak{M}(\text{sID}^*) := \{\text{sID} \mid \text{owner}[\text{sID}] \neq i^* \ \wedge \ \mathcal{Q}[\text{sID}] = \mathcal{Q}^*$
        $\wedge \{\mathsf{Msg_I}[\text{sID}]\} \cup \mathcal{M}[\text{sID}] = \{\mathsf{Msg_I}[\text{sID}^*]\} \cup \mathcal{M}[\text{sID}^*]$
        $\wedge \{\mathsf{Msg_R}[\text{sID}]\} \cup \hat{\mathcal{M}}[\text{sID}] = \{\mathsf{Msg_R}[\text{sID}^*]\} \cup \hat{\mathcal{M}}[\text{sID}^*]\}$        //matching sessions
if revealed[sID*] or $(\exists \text{sID} \in \mathfrak{M}(\text{sID}^*) : \text{revealed}[\text{sID}] = \textbf{true})$
    return false        //$\mathcal{A}$ trivially learned the test session's key
if $\exists \text{sID} \in \mathfrak{M}(\text{sID}^*)$ s. t. $\text{sID} \in \mathcal{S}$
    return false        //$\mathcal{A}$ also tested a matching session
return true

VALID(sID*)
$(i^*, \mathcal{Q}^*) := (\text{owner}[\text{sID}^*], \mathcal{Q}[\text{sID}^*])$
$\mathfrak{M}(\text{sID}^*) := \{\text{sID} \mid \text{owner}[\text{sID}] \neq i^* \ \wedge \ \mathcal{Q}[\text{sID}] = \mathcal{Q}^*$
        $\wedge \{\mathsf{Msg_I}[\text{sID}]\} \cup \mathcal{M}[\text{sID}] = \{\mathsf{Msg_I}[\text{sID}^*]\} \cup \mathcal{M}[\text{sID}^*]$
        $\wedge \{\mathsf{Msg_R}[\text{sID}]\} \cup \hat{\mathcal{M}}[\text{sID}] = \{\mathsf{Msg_R}[\text{sID}^*]\} \cup \hat{\mathcal{M}}[\text{sID}^*]\}$        //matching sessions
for attack $\in$ Table 3
    if attack = true return true
return false
```

**Fig. 16.** Helper procedures FRESH and VALID for game IND-G-FS defined in Fig. 15. 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 3 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 .

**Table 3.** Table of attacks for adversaries against explicitly authenticated group key exchange protocols without ephemeral state reveals.

| | $\mathcal{A}$ gets (owner[sID*], $\mathcal{P}_i$ := peer[sID*]) | peerCorrupted[sID*] | $\lvert\mathfrak{M}(\text{sID}^*)\rvert$ |
|---|---|---|---|
| 0. | **multiple matching sessions** | – | $> \lvert\mathcal{P}_i\rvert$ |
| 1. | **(long-term, long-term)** | – | $= \lvert\mathcal{P}_i\rvert$ |
| 2. | **(long-term, long-term)** | **F** | $< \lvert\mathcal{P}_i\rvert$ |

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"

The OW-G-HV security is formally defined by Definition 6 with the security game OW-G-HV as in Fig. 17.

**Definition 6.** (*Group One-Wayness against Honest and Key Verification Attacks* (OW-G-HV)) A group key exchange protocol GKE is $(t, \varepsilon, \mu, S, Q_V)$-OW-G-HV-secure where $\mu$ is the number of users, $S$ is the number of sessions and $Q_V$ is the number of call to KVER, if for all adversaries $\mathcal{A}$ attacking the protocol in time at most $t$,

```
GAME OW-G-HV
$\mathrm{cnt}_\mathsf{S} := 0$   // total session counter
$(\mathrm{sID}^*, K^*) \leftarrow \mathcal{A}^{\mathrm{O}}([\mu])$
if $\mathrm{sID}^* > \mathrm{cnt}_\mathsf{S}$
    return $\bot$
return $\mathrm{KVER}(\mathrm{sID}^*, K^*)$

SESSION_I$(i \in [\mu], \mathcal{P}_i \subseteq [\mu])$
$\mathrm{cnt}_\mathsf{S}$ ++
$\mathrm{sID} := \mathrm{cnt}_\mathsf{S}$
$\mathrm{owner}[\mathrm{sID}] := i$
$\mathrm{peer}[\mathrm{sID}] := \mathcal{P}_i$
$\mathcal{Q}[\mathrm{sID}] := \mathrm{peer}[\mathrm{sID}] \cup \{i\}$
$(m_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Init}(i, \mathcal{P}_i)$
$\mathsf{Msg}_\mathsf{I}[\mathrm{sID}] := (i, m_i)$
$\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
return $(\mathrm{sID}, m_i)$

KVER$(\mathrm{sID}, K)$
return $[\![\mathrm{sKey}[\mathrm{sID}] = K]\!]$

SESSION_R$(\mathrm{sID}, \mathcal{M}_i)$
$(i, \mathcal{P}_i) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\mathcal{M}_i| \neq |\mathcal{P}_i|$
    return $\bot$
if $\exists (j, m') \in \mathcal{M}_i : \forall \mathrm{sID}' \in [\mathrm{cnt}_\mathsf{S}] : \mathsf{Msg}_\mathsf{I}[\mathrm{sID}'] \neq (j, m')$
    return $\bot$   // $(j, m')$ is not honest
$\mathcal{M}[\mathrm{sID}] := \mathcal{M}_i$
$(\hat{m}_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Res}(i, \mathcal{P}_i, \mathrm{state}[\mathrm{sID}], \mathcal{M}_i)$
$\mathsf{Msg}_\mathsf{R}[\mathrm{sID}] := (i, \hat{m}_i)$
$\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
return $\hat{m}_i$

DER$(\mathrm{sID}, \hat{\mathcal{M}}_i)$
if $\mathrm{sKey}[\mathrm{sID}] \neq \bot$
    return $\bot$
$(i, \mathcal{P}_i) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\hat{\mathcal{M}}_i| \neq |\mathcal{P}_i|$
    return $\bot$
if $\exists (j, \hat{m}') \in \hat{\mathcal{M}}_i : \forall \mathrm{sID}' \in [\mathrm{cnt}_\mathsf{S}] : \mathsf{Msg}_\mathsf{R}[\mathrm{sID}'] \neq (j, \hat{m}')$
    return $\bot$   // $(j, \hat{m}')$ is not honest
$\hat{\mathcal{M}}[\mathrm{sID}] := \hat{\mathcal{M}}_i$
$\mathcal{M}_i := \mathcal{M}[\mathrm{sID}]$
$K := \mathsf{Der}(i, \mathcal{P}_i, \mathrm{state}[\mathrm{sID}], \mathcal{M}_i, \hat{\mathcal{M}}_i)$
$\mathrm{sKey}[\mathrm{sID}] := K$
return $\epsilon$
```

**Fig. 17.** Game OW-G-HV for GKE. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_\mathsf{I}, \textsc{Session}_\mathsf{R}, \textsc{Der}, \textsc{KVer}\}$ .

we have:

$$\Pr[\mathsf{OW\text{-}G\text{-}HV}^{\mathcal{A}} \Rightarrow 1] \leq \varepsilon.$$

We require that a group key exchange protocol GKE has $\alpha$-bits of min-entropy, namely if for all messages $m'$ we have $\Pr[m = m'] \leq 2^{-\alpha}$, where $m$ is output by either Init or Res.

### 6.3. *Instantiation of* OW-G-HV *with Burmester–Desmedt*

We show that the Burmester–Desmedt group key exchange protocol [12] is OW-G-HV secure. We begin by describing the protocol in our framework, and then prove its security based on the strong computational Diffie–Hellman assumption.

Let $\mathsf{par} = (p, g, \mathbb{G})$ define a prime-order cyclic group $\mathbb{G} := \langle g \rangle$. We choose a group of users $\mathcal{Q}$ with $|\mathcal{Q}| = n$, and order the participants as $\mathsf{P}_1$ to $\mathsf{P}_n$ in a cycle. Messages $m_i$ and $\hat{m}_i$ are sent by $\mathsf{P}_i$. We then have $\mathsf{P}_{n+1} = \mathsf{P}_1$, and for the messages $m_{n+1}$ and $\hat{m}_{n+1}$ we have $m_{n+1} = m_1$ and $\hat{m}_{n+1} = \hat{m}_1$.

The Burmester–Desmedt protocol is described in Fig. 18, and for correctness we show that all parties compute the key

$$K = g^{r_1 r_2 + r_2 r_3 + \cdots r_{n-1} r_n + r_n r_1}. \tag{8}$$

```
Init($i, \{j\}_{j\in\mathcal{P}}$):
01 $\mathrm{st} := r_i \xleftarrow{\$} \mathbb{Z}_p$
02 $m_i := g^{r_i}$
03 return $(m_i, \mathrm{st})$

Res($i, \{j\}_{j\in\mathcal{P}}, \mathrm{st}, \mathcal{M}$):
04 $\hat{m}_i := (m_{i+1}/m_{i-1})^{\mathrm{st}}$
05 return $\hat{m}_i$

Der($i, \{j\}_{j\in\mathcal{P}}, \mathrm{st}, \mathcal{M}, \hat{\mathcal{M}}$):
06 $K := m_{i-1}^{n\cdot\mathrm{st}} \cdot \hat{m}_i^{n-1} \cdot \hat{m}_{i+1}^{n-2} \cdots \hat{m}_{i-2}$
```

**Fig. 18.** The Burmester–Desmedt protocol, $\mathsf{GKE_{BD}}$ .

Recall that for user $i$, we have $\mathrm{st} := r_i$. We define the following values:

$$\begin{aligned} A_{i-1} &:= m_{i-1}^{\mathrm{st}} = g^{r_{i-1}r_i} \\ A_i &:= m_{i-1}^{\mathrm{st}} \cdot \hat{m}_i = g^{r_i r_{i+1}} \\ A_{i+1} &:= m_{i-1}^{\mathrm{st}} \cdot \hat{m}_i \cdot \hat{m}_{i+1} = g^{r_{i+1}r_{i+2}} \\ &\vdots\ \vdots \\ A_{i-2} &:= m_{i-1}^{\mathrm{st}} \cdot \hat{m}_i \cdot \hat{m}_{i+1} \cdots \hat{m}_{i-2} = g^{r_{i-2}r_{i-1}}. \end{aligned}$$

It then follows that for the key computed in line 06 of Fig. 18, we have

$$K = m_{i-1}^{n\cdot\mathrm{st}} \cdot \hat{m}_i^{n-1} \cdot \hat{m}_{i+1}^{n-2} \cdots \hat{m}_{i-2} = A_{i-1}A_iA_{i+1}\cdots A_{i-2} = g^{r_1r_2+r_2r_3+\cdots r_{n-1}r_n+r_nr_1}.$$

**Lemma 2.** *Let* $\mathsf{GKE_{BD}}$ *be the Burmester–Desmedt group key exchange protocol as in Fig. 18. Then,* $\mathsf{GKE_{BD}}$ *has* $\alpha = \log_2 p$ *bits of min-entropy, and for every adversary* $\mathcal{A}$ *that breaks the* $(t, \varepsilon, \mu, S, Q_V)$*-security of* $\mathsf{GKE_{BD}}$*, there exists an adversary* $\mathcal{B}$ *which breaks the* $(t', \varepsilon', Q'_V)$*-security of* $\mathsf{StCDH}$ *with*

$$\varepsilon \le \varepsilon', \quad t \approx t', \quad Q'_V = Q_V + 1. \tag{9}$$

*Proof.* The entropy statement is again straightforward, since $r_i$ being drawn uniformly at random implies that both $m_i$ and $\hat{m}_i$ are uniformly random as well.

We now construct a simulator $\mathcal{B}$, which on input $(g^x, g^y)$ breaks the $\mathsf{CDH}$ assumption by simulating the $\mathsf{OW\text{-}G\text{-}HV}$ game to $\mathcal{A}$.

To simulate $\textsc{Session}_\mathsf{I}(i \in [\mu], \mathcal{P}_i \subseteq [\mu])$, $\mathcal{B}$ proceeds as in Fig. 17, but instead of running the $\mathsf{Init}$ algorithm in line 10, it does the following:

- if $i$ is odd, $\mathcal{B}$ draws an element $a_i \leftarrow_\$ \mathbb{Z}_p$ and sets and returns $m_i := g^x g^{a_i}$
- if $i$ is even, $\mathcal{B}$ draws an element $a_i \leftarrow_\$ \mathbb{Z}_p$ and sets and returns $m_i := g^y g^{a_i}$.

All $m_i$'s are uniformly distributed, exactly as in the original protocol.

To simulate $\textsc{Session}_\mathsf{R}$, note that $\mathcal{B}$ does not know the discrete logarithm of $m_i$'s, but it can compute $\hat{m}_i$ in the following way: If $i$ is even, $\mathcal{B}$ computes $\hat{m}_i := m_i^{a_{i+1}-a_{i-1}}$, since we have

$$\begin{aligned} \hat{m}_i &:= (m_{i+1}/m_{i-1})^{y+a_i} = (g^{x+a_{i+1}}/g^{x+a_{i-1}})^{y+a_i} = (g^{a_{i+1}-a_{i-1}})^{y+a_i} \\ &= (g^{y+a_i})^{a_{i+1}-a_{i-1}} = m_i^{a_{i+1}-a_{i-1}}. \end{aligned} \tag{10}$$

Simulation of $\hat{m}_i$ for odd $i$ is similar. Equation (10) shows that the simulated $\hat{m}_i$ are distributed the same as in the real distribution.

To simulate Der, $\mathcal{B}$ follows the steps in Fig. 17, but skips the key derivation in line 34 and leaves the corresponding session key empty. Since there are no session-key-reveal oracles in this game, $\mathcal{A}$ will not notice this and the simulation is perfect from $\mathcal{A}$'s viewpoint.

To simulate the KVer oracle on input (sID, $K$), for readability, we label $r_i := x + a_i$ for odd $i$ and $r_i := y + a_i$ for even $i$ and $m_i = g^{r_i}$ for all $i$. Recall that the derived session key in $\mathsf{GKE_{BD}}$ is $K = g^{r_1 r_2 + r_2 r_3 + \cdots r_{n-1} r_n + r_n r_1}$. We then write

$$g^{r_i r_{i+1}} = g^{(x+a_i)(y+a_{i+1})} = g^{(xy + x a_{i+1} + a_i(y+a_{i+1}))} = g^{xy} (g^x)^{a_{i+1}} (g^y)^{a_i} g^{a_i a_{i+1}}$$

for odd $i$, and

$$g^{r_i r_{i+1}} = g^{(y+a_i)(x+a_{i+1})} = g^{(xy + x a_i + a_{i+1}(y+a_i))} = g^{xy} (g^x)^{a_i} (g^y)^{a_{i+1}} g^{a_i a_{i+1}}$$

for even $i$. Note that all $a_i$'s are known. If $K$ is valid for an sID, we have

$$\begin{aligned} K &= g^{r_1 r_2 + r_2 r_3 + \cdots r_{n-1} r_n + r_n r_1} \\ &= \prod_{i=1}^{n} g^{r_i r_{i+1}} \\ &= \prod_{\substack{1 \le i \le n \\ i \equiv 1 \bmod 2}} g^{r_i r_{i+1}} \prod_{\substack{1 \le i \le n \\ i \equiv 0 \bmod 2}} g^{r_i r_{i+1}} \\ &= \prod_{\substack{1 \le i \le n \\ i \equiv 1 \bmod 2}} g^{xy} (g^x)^{a_{i+1}} (g^y)^{a_i} g^{a_i a_{i+1}} \prod_{\substack{1 \le i \le n \\ i \equiv 0 \bmod 2}} g^{xy} (g^x)^{a_i} (g^y)^{a_{i+1}} g^{a_i a_{i+1}} \\ &= g^{nxy} g^{\sum_{i=1}^{n} a_i a_{i+1}} \prod_{\substack{1 \le i \le n \\ i \equiv 1 \bmod 2}} (g^x)^{a_{i+1}} (g^y)^{a_i} \prod_{\substack{1 \le i \le n \\ i \equiv 0 \bmod 2}} (g^x)^{a_i} (g^y)^{a_{i+1}}. \end{aligned}$$

This implies that we can compute

$$\tilde{K} := \left( K \Big/ \left( g^{\sum_{i=1}^{n} a_i a_{i+1}} \prod_{\substack{1 \le i \le n \\ i \equiv 1 \bmod 2}} (g^x)^{a_{i+1}} (g^y)^{a_i} \prod_{\substack{1 \le i \le n \\ i \equiv 0 \bmod 2}} (g^x)^{a_i} (g^y)^{a_{i+1}} \right) \right)^{n^{-1}} . \quad (11)$$

If $K$ is valid for an sID, we have $\tilde{K} = g^{xy}$. Hence, $\mathcal{B}$ queries $\mathrm{DH}_x \left( g^y, \tilde{K} \right)$ to verify the key, and returns the answer. This completes the simulation.

If $\mathcal{A}$ is able to compute a valid session key, then $\mathcal{B}$ wins the $\mathsf{StCDH}$ game, and hence $\varepsilon \le \varepsilon'$. The running time of $\mathcal{B}$ is that of $\mathcal{A}$ plus one exponentiation for each $\mathrm{Session_I}$ and $\mathrm{Session_R}$ call, and 6 exponentiations and one inversion (disregarding the inversion of $n$, which is essentially free) for each call to KVer, since we can sum the various exponents together before we perform the exponentiations in the denominator. The total number

```
$\mathsf{Gen}_{\mathsf{GAKE}}(\mathsf{par})$:
$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{par})$
$\mathbf{return}\ (\mathsf{pk}, \mathsf{sk})$
$\mathsf{Res}(\mathsf{sk}_i, i, \mathcal{P}, \mathrm{st}, \mathcal{M}_i)$ :
$\mathcal{Q} := \{i\} \cup \mathcal{P}$
$\mathbf{parse}\ (\{m_j, \sigma_j\}_{j \in \mathcal{P}}) =: \mathcal{M}_i$
$\mathbf{for}\ j \in \mathcal{P}$
  $\mathbf{if}\ \mathsf{Ver}(\mathsf{pk}_j, m_j, \sigma_j) = 0$
    $\mathbf{return}\ \perp$
$\mathbf{parse}\ (\hat{\mathrm{st}}, m_i) =: \mathrm{st}$
$(\hat{m}_i, \hat{\mathrm{st}}') \xleftarrow{\$} \mathsf{Res}'(i, \mathcal{P}, \hat{\mathrm{st}}, \{m_j\}_{j \in \mathcal{P}})$
$\mathrm{st}' := (\mathrm{st}', m_i, \hat{m}_i)$
$\pi_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, (\{m_j\}_{j \in \mathcal{P}}, \hat{m}_i))$
$\mathbf{return}\ (\hat{m}_i, \pi_i, \mathrm{st}')$

$\mathsf{Init}(\mathsf{sk}_i, i, \mathcal{P})$ :
$\mathcal{Q} := \{i\} \cup \mathcal{P}$
$(m_i, \hat{\mathrm{st}}) \xleftarrow{\$} \mathsf{Init}'(i, \mathcal{P})$
$\mathrm{st} := (\mathrm{st}, m_i)$
$\sigma_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, m_i)$
$\mathbf{return}\ (m_i, \sigma_i, \mathrm{st})$
$\mathsf{Der}(\mathsf{sk}_i, i, \mathcal{P}, \mathrm{st}, \mathcal{M}_i, \hat{\mathcal{M}}_i)$ :
$\mathcal{Q} := \{i\} \cup \mathcal{P}$
$\mathbf{parse}\ (\{m_j, \sigma_j\}_{j \in \mathcal{P}}) =: \mathcal{M}_i$
$\mathbf{parse}\ (\{\hat{m}_j, \pi_j\}_{j \in \mathcal{P}}) =: \hat{\mathcal{M}}_i$
$\mathbf{parse}\ (\hat{\mathrm{st}}, m_i, \hat{m}_i) =: \mathrm{st}$
$\mathbf{for}\ j \in \mathcal{P}$
  $\mathbf{if}\ \mathsf{Ver}(\mathsf{pk}_j, m_j, \sigma_j) = 0\ \mathbf{or}$
  $\mathsf{Ver}(\mathsf{pk}_j, (\{m_j\}_{j \in \mathcal{P}}, \hat{m}_j), \pi_j) = 0$
    $\mathbf{return}\ \perp$
$K^* := \mathsf{Der}'(\mathsf{sk}_i, \mathcal{P}, \hat{\mathrm{st}}, \{m_j, \hat{m}_j\}_{j \in \mathcal{Q}})$
$\mathrm{ctxt} := (\mathcal{Q}, \{m_j, \hat{m}_j\}_{j \in \mathcal{Q}})$
$K := \mathsf{H}(\mathrm{ctxt}, K^*)$
$\mathbf{return}\ K$
```

**Fig. 19.** Generic construction of GAKE from SIG, GKE and a random oracle H .

of queries $Q'_V$ to $\mathrm{DH}_x$ is $Q'_V = Q_V + 1$, as we get one additional call to KVer when we verify the adversaries forgery. This completes the lemma. □

### 6.4. *Our Generic Transformation for GAKE*

Following the construction from Sect. 5, we construct an IND-G-FS-secure authenticated group key exchange protocol GAKE = ($\mathsf{Gen}_{\mathsf{GAKE}}$, Init, Res, Der) by combining a StCorrCMA-secure signature scheme SIG = (Gen, Sign, Ver), an OW-G-HV-secure group key exchange protocol GKE = (Init′, Res′, Der′), and a random oracle H. The construction is given in Fig. 19

**Theorem 3.** *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, Q_{\mathsf{H}}, Q_{\mathrm{COR}})$-IND-G-FS security of a protocol GAKE constructed as in Fig. 19, we can construct an adversary $\mathcal{B}$ that breaks the $(t', \varepsilon', \mu, Q_s, Q_{\mathsf{H}}, Q'_{\mathrm{COR}})$-StCorrCMA security of the underlying signature scheme SIG with $\alpha$ bits of key min-entropy, or breaks the $(t'', \varepsilon'', \mu, S', Q_V)$-OW-G-HV security of the underlying key exchange protocol $\Pi$ with $\beta$ bits of min-entropy, such that*

$$\varepsilon \leq 2\varepsilon' + \varepsilon'' + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}},$$
$$t' \approx t, \quad Q_s \leq S, \quad Q'_{\mathrm{COR}} = Q_{\mathrm{COR}},$$
$$t'' \approx t, \quad S = S', \quad Q_V \leq Q_{\mathsf{H}}.$$

*Proof.* We will prove this by using the following hybrid games, which are illustrated in Fig. 20.

Game $G_0$: This is the original IND-G-FS for the protocol GAKE. We assume that all long-term keys, and all messages generated by Init and Res are distinct. The security game aborts if a collision happens. Using the fact that SIG has $\alpha$-bits of key min-entropy and GKE has $\beta$-bits of message min-entropy, a collision in the keys happens with

```
GAME IND-G-FS
01 for $n \in [\mu]$
02    $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{GAKE}}(\mathsf{par})$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
05 for $\mathrm{sID}^* \in \mathcal{S}$:
06    if $\textsc{Fresh}(\mathrm{sID}^*) =$ false
07       return 0                                   //session not fresh
08    if $\textsc{Valid}(\mathrm{sID}^*) =$ false
09       return 0                                   //no valid attack
10 return $[\![b = b']\!]$

SESSION_I$(i \in [\mu], \mathcal{P} \subseteq [\mu])$
11 $\mathrm{cnt}_{\mathrm{S}}$ ++
12 $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
13 $\mathrm{owner}[\mathrm{sID}] := i$
14 $\mathrm{peer}[\mathrm{sID}] := \mathcal{P}$
15 $\mathcal{Q} := \mathrm{peer}[\mathrm{sID}] \cup \{i\}$
16 $(m_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Init}(\mathsf{sk}_i, \mathcal{P})$
17 $\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}] := (i, m_i)$
18 $\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
19 return $(\mathrm{sID}, m_i)$

SESSION_R$(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], \mathcal{M}_i)$
20 $(i, \mathcal{P}) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
21 $\mathcal{Q} := \{i\} \cup \mathcal{P}$
22 if $|\mathcal{M}_i| \neq |\mathcal{P}|$
23    return $\perp$                      //all peers must have broadcasted
24 parse $\{(j, m_j)\}_{j \in \mathcal{P}} =: \mathcal{M}_i$
25 $\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}} \mathrm{corrupted}[j]$
26 if $\mathrm{peerCorrupted}[\mathrm{sID}] =$ false                //$G_{1\text{-}2}$
27    for $j \in \mathcal{P}$                                     //$G_{1\text{-}2}$
28       if $\nexists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'_j])$
            $= (j, \mathcal{Q} \setminus \{j\}, (j, m_j))$                //$G_{1\text{-}2}$
29          $\mathsf{AbortSessR} =$ true                          //$G_{1\text{-}2}$
30          abort                                             //$G_{1\text{-}2}$
31 $(\hat{m}_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Res}(\mathsf{sk}_i, i, \mathcal{P}, \mathrm{state}[\mathrm{sID}], \mathcal{M}_i)$
32 $\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}] := (i, \hat{m}_i)$
33 $\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
34 $\mathcal{M}[\mathrm{sID}] := \mathcal{M}_i$
35 return $\hat{m}_i$

DER$(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], \hat{\mathcal{M}}_i)$
36 if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$
37    return $\perp$
38 $(i, \mathcal{P}) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
39 $\mathcal{Q} := \{i\} \cup \mathcal{P}$
40 if $|\hat{\mathcal{M}}_i| \neq |\mathcal{P}|$ return $\perp$
41 parse $\{j, m_j\}_{j \in \mathcal{P}} =: \mathcal{M}[\mathrm{sID}]$
42 parse $\{j, \hat{m}_j\}_{j \in \mathcal{P}} =: \hat{\mathcal{M}}_i$
43 $\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}} \mathrm{corrupted}[j]$
44 if $\mathrm{peerCorrupted}[\mathrm{sID}] =$ false                //$G_{1\text{-}2}$
45    for $j \in \mathcal{P}$                                     //$G_{1\text{-}2}$
46       if $\nexists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'_j], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}'_j])$
         $= (j, \mathcal{Q} \setminus \{j\}, (j, m_j), (j, \hat{m}_j))$      //$G_{1\text{-}2}$
47          $\mathsf{AbortDer} =$ true                            //$G_{1\text{-}2}$
48          abort                                             //$G_{1\text{-}2}$
49 $K := \mathsf{Der}(\mathsf{sk}_i, i, \mathcal{P}, \mathrm{state}[\mathrm{sID}], \mathcal{M}[\mathrm{sID}], \hat{\mathcal{M}}_i)$
50 $\mathrm{sKey}[\mathrm{sID}] := K$
51 return $\epsilon$

REVEAL(sID)
52 $\mathrm{revealed}[\mathrm{sID}] :=$ true
53 return $\mathrm{sKey}[\mathrm{sID}]$

CORR$(n \in [\mu])$
54 $\mathrm{corrupted}[n] :=$ true
55 return $\mathsf{sk}_n$

TEST(sID)
56 if $\mathrm{sID} \in \mathcal{S}$ return $\perp$                      //already tested
57 if $\mathrm{sKey}[\mathrm{sID}] = \perp$ return $\perp$
58 $\mathcal{S} := \mathcal{S} \cup \{\mathrm{sID}\}$
59 $K_0^* := \mathrm{sKey}[\mathrm{sID}]$                              //$G_{0\text{-}1}$
60 $K_0^* \xleftarrow{\$} \mathcal{K}$                                 //$G_2$
61 $K_1^* \xleftarrow{\$} \mathcal{K}$
62 return $K_b^*$
```

**Fig. 20.** Games $G_0$-$G_2$ .

probability at most $\mu^2/2^{\alpha+1}$, and a collision in the messages happens with probability at most $S^2/2^{\beta+1}$. Here, $\mu$ is the number of users and $S$ is the number of sessions. Thus, we have:

$$\Pr[\textsf{IND-G-FS}^{\mathcal{A}} \Rightarrow 1] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \frac{\mu^2}{2^{\alpha+1}} - \frac{S^2}{2^{\beta+1}}. \tag{12}$$

GAME $G_1$: In this game, SESSION$_{\mathsf{R}}$ and DER abort upon input a session id and a message set which do not correspond to a previously broadcast message set (*i.e.* all messages are honestly generated by using the given oracles; however, there may still be non-matching sessions), and all signatures with respect to each non-corrupted party in the group are valid. This step is to exclude the active attacks where an adversary creates its own message. This change is unnoticed by the adversary, since it requires him to forge at least one valid signature for the underlying $\mathsf{StCorrCMA}$ secure signature scheme. We will give a formal proof of the indistinguishability of $G_0$ and $G_1$ in Lemma 3. We denote the abort event as $\mathsf{AbortGAKE} := \mathsf{AbortSessR} \cup \mathsf{AbortDer}$, where $\mathsf{AbortSessR}$ and $\mathsf{AbortDer}$ correspond to the aborting event in line line 29 and line 47 of Fig. 20, respectively.

Since the only difference between $G_0$ and $G_1$ is the aborting events $\mathsf{AbortGAKE}$, using Lemma 3 we have

$$\begin{aligned}\Pr[G_1^{\mathcal{A}} \Rightarrow 1] &\geq \Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{AbortSessR}] \\ &\quad -\Pr[\mathsf{AbortDer}] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1] - 2\varepsilon'. \end{aligned} \tag{13}$$

Game $G_2$: Intuitively, since in $G_1$ an adversary $\mathcal{A}$ is not allowed to create its own message for active attacks against the protocol, $\mathcal{A}$ can either observe the protocol execution or forward the honestly generated messages in a different order. We will use the $\mathsf{OW\text{-}G\text{-}HV}$ security to tightly argue the indistinguishability of a real session key and a uniformly random one. Formally in $G_2$, the Test oracle always returns a uniformly random key, independent on the bit $b$. Since we already in $G_0$ assume that all messages generated by $\mathsf{Init}$ and $\mathsf{Res}$ are distinct, and we are in the random oracle model, the only way for $\mathcal{A}$ to compute a valid session key $K$ is to query the correct input. Therefore, by Lemma 4 we can reduce the difference between $G_2$ and $G_1$ to the $\mathsf{OW\text{-}G\text{-}HV}$ security of $\mathsf{GKE}$, and we have

$$\Pr[G_1^{\mathcal{A}} \Rightarrow 1] \geq \Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \varepsilon''. \tag{14}$$

In summary, we have

$$\begin{aligned}&\varepsilon \leq 2\varepsilon' + \varepsilon'' + \frac{\mu^2}{2^{\alpha+1}} + \frac{S^2}{2^{\beta+1}}, \\ &t' \approx t, \quad Q_s \leq S, \quad Q'_{\mathrm{COR}} = Q_{\mathrm{COR}}, t'' \approx t, \quad S = S', \quad Q_V \leq Q_{\mathsf{H}}.\end{aligned}$$

□

**Lemma 3.** *For every adversary $\mathcal{A}$ running in time $t_{0,1}$ that distinguishes $G_0$ from $G_1$ with probability $\varepsilon_{0,1}$, we can construct an adversary $\mathcal{B}$ against the* $(t', \varepsilon', \mu, Q_{\mathsf{H}}, Q'_{\mathrm{COR}})$-$\mathsf{StCorrCMA}$ *security of the underlying signature scheme* $\mathsf{SIG}$*, where*

$$t_{0,1} \approx t', \qquad \varepsilon_{0,1} \leq 2\varepsilon', \qquad Q'_{\mathrm{COR}} = Q_{\mathrm{COR}}.$$

*Proof.* The only difference between $G_0$ and $G_1$ is the aborting events $\mathsf{AbortSessR}$ and $\mathsf{AbortDer}$. To bound the probability of these, we build an adversary $\mathcal{B}$ against the $\mathsf{StCorrCMA}$ of the underlying signature scheme $\mathsf{SIG}$ as in Fig. 21. The adversary will successfully generate a valid forgery if and only if $\mathsf{AbortSessR}$ or $\mathsf{AbortDer}$ happens.

More precisely, if $\mathsf{AbortGAKE}$ is **true**, then the signatures in line 31 and in line 54 of Fig. 21 are valid forgeries against the $\mathsf{CorrCMA}$ security of $\mathsf{SIG}$. Here, we only prove the case where $\mathsf{AbortSessR} =$ **true**. The other case where $\mathsf{AbortDer} =$ **true** follows the same idea. Given the fact that $\mathsf{AbortSessR}$ happens, we have that for all $j \in \mathcal{P}$, $\mathsf{Ver}(\mathsf{pk}_j, m_j, \sigma_j) = 1$ and peerCorrupted[sID] $=$ **false**. Moreover, due to the criteria of line 30, there exists $j^* \in \mathcal{P}$ such that $(j^*, (m_{j^*}, \sigma_{j^*}))$ has never been output by $\mathrm{Session}_{\mathsf{I}}$. Therefore, $(m_{j^*}, \sigma_{j^*})$ is a valid forgery against the $\mathsf{CorrCMA}$ security of

```
$\mathcal{B}^{\textsc{Corr}',\textsc{Sign}'}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\mu)$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
for $\mathrm{sID}^* \in \mathcal{S}$:
    if $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$
        return 0                                  //session not fresh
    if $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
        return 0                                  //no valid attack
return $[\![\Sigma \in \mathsf{Win}_{\mathsf{StCorrCMA}}]\!]$            //break StCorrCMA

$\textsc{Session}_\mathsf{I}(i \in [\mu], \mathcal{P} \subseteq [\mu])$
$\mathrm{cnt}_\mathrm{S}$ ++
$\mathrm{sID} := \mathrm{cnt}_\mathrm{S}$
$\mathrm{owner}[\mathrm{sID}] := i$
$\mathrm{peer}[\mathrm{sID}] := \mathcal{P}$
$\mathcal{Q} := \mathrm{peer}[\mathrm{sID}] \cup \{i\}$
$(m_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Init}(\mathsf{sk}_i, \mathcal{P})$
$\sigma_i \xleftarrow{\$} \textsc{Sign}(i, m_i)$
$\mathsf{Msg}_\mathsf{I}[\mathrm{sID}] := (i, (m_i, \sigma_i))$
$\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
return $(\mathrm{sID}, m_i)$

$\textsc{Session}_\mathsf{R}(\mathrm{sID} \in [\mathrm{cnt}_\mathrm{S}], \mathcal{M}_i)$
$(i, \mathcal{P}) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\mathcal{M}_i| \neq |\mathcal{P}|$
    return $\perp$                        //all peers must have responded
$\mathcal{Q} := \{i\} \cup \mathcal{P}$
parse $\{(j, (m_j, \sigma_j))\}_{j \in \mathcal{P}} =: \mathcal{M}_i$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}} \mathrm{corrupted}[j]$
for $j \in \mathcal{P}$
    if $\mathsf{Ver}(\mathsf{pk}_j, m_j, \sigma_j) = 0$
        return $\perp$
if $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$
    for $j \in \mathcal{P}$
        if $\nexists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j], \mathsf{Msg}_\mathsf{I}[\mathrm{sID}'_j])$
          $= (j, \mathcal{Q} \setminus \{j\}, (j, (m_j, \sigma_j)))$
            $\Sigma := (\mathsf{pk}_j, m_j, \sigma_j)$              //valid forgery
            $\mathsf{AbortSessR} = \mathbf{true}$
            abort
$(\hat{m}_i, \mathrm{st}) \xleftarrow{\$} \mathsf{Res}(\mathsf{sk}_i, \mathcal{P}, \mathrm{state}[\mathrm{sID}], \mathcal{M}_i)$
$\pi_i \xleftarrow{\$} \textsc{Sign}(i, (\{m_j\}_{j \in \mathcal{Q}}, \hat{m}_i))$
$\mathsf{Msg}_\mathsf{R}[\mathrm{sID}] := (i, (\hat{m}_i, \pi_i))$
$\mathrm{state}[\mathrm{sID}] := \mathrm{st}$
$\mathcal{M}[\mathrm{sID}] := \mathcal{M}_i$
return $\hat{m}_i$

$\textsc{Der}(\mathrm{sID} \in [\mathrm{cnt}_\mathrm{S}], \hat{\mathcal{M}}_i)$
if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$
    return $\perp$
$(i, \mathcal{P}) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\hat{\mathcal{M}}_i| \neq |\mathcal{P}|$ return $\perp$
$\mathcal{Q} := \{i\} \cup \mathcal{P}$
parse $\{(j, (m_j, \sigma_j))\}_{j \in \mathcal{P}} =: \mathcal{M}[\mathrm{sID}], \{(j, (m_j, \pi_j))\}_{j \in \mathcal{P}} =: \hat{\mathcal{M}}_i$
parse $(i, (m_i, \sigma_i)) =: \mathsf{Msg}_\mathsf{I}[\mathrm{sID}]$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}} \mathrm{corrupted}[j]$
for $j \in \mathcal{P}$
    if $\mathsf{Ver}(\mathsf{pk}_j, (\{m_k\}_{k \in \mathcal{Q}}, \hat{m}_j), \pi_j) = 0$
        return $\perp$
if $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$
    for $j \in \mathcal{P}$
        if $\nexists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j], \mathsf{Msg}_\mathsf{I}[\mathrm{sID}'_j], \mathsf{Msg}_\mathsf{R}[\mathrm{sID}'_j])$
          $= (j, \mathcal{Q} \setminus \{j\}, (j, (m_j, \sigma_j)), (j, (\hat{m}_j, \pi_j)))$
            $\Pi := (\mathsf{pk}_j, (\{m_j\}_{j \in \mathcal{Q}}, \hat{m}_j), \pi_j)$          //valid forgery
            $\mathsf{AbortDer} := \mathbf{true}$
            abort
$K^* := \mathsf{Der}(\mathsf{sk}_i, \mathcal{P}, \mathrm{state}[\mathrm{sID}], \mathcal{M}[\mathrm{sID}], \hat{\mathcal{M}}_i)$
$\mathrm{ctxt} := (\mathcal{Q}, \mathcal{M}[\mathrm{sID}] \cup \{\mathsf{Msg}_\mathsf{I}[\mathrm{sID}]\}, \hat{\mathcal{M}}_i \cup \{\mathsf{Msg}_\mathsf{R}[\mathrm{sID}]\})$
$K := \mathsf{H}(\mathrm{ctxt}, K^*)$
$\mathrm{sKey}[\mathrm{sID}] := K$
return $\epsilon$

$\textsc{Corr}(n \in [\mu])$
$\mathrm{corrupted}[n] := \mathbf{true}$
$\mathsf{sk}_n \leftarrow \textsc{Corr}'(n)$
return $\mathsf{sk}_n$

$\mathsf{H}(\mathrm{ctxt}, K^*)$
if $\mathsf{H}[\mathrm{ctxt}, K^*] = K$
    return $K$
$K \xleftarrow{\$} \mathcal{K}$
$\mathsf{H}[\mathrm{ctxt}, K^*] := K$
return $K$
```

**Fig. 21.** Adversary $\mathcal{B}$ against the $(t', \varepsilon', \mu, Q_s, Q_{\mathrm{COR}})$-StCorrCMA of SIG. The StCorrCMA game provides oracles $\textsc{Sign}', \textsc{Corr}'$. The adversary $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_\mathsf{I}, \textsc{Session}_\mathsf{R}, \textsc{Der}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{H}\}$, where $\textsc{Reveal}$ and $\textsc{Test}$ remain the same as in Fig. 15. We highlight the most relevant codes with **bold** line numbers .

SIG, and we have

$$\Pr[\mathsf{AbortSessR}] \le \varepsilon'.$$

Similarly, we also have $\Pr[\mathsf{AbortDer}] \le \varepsilon'$. Overall, we have

$$t_{0,1} \approx t', \qquad \varepsilon_{0,1} \le 2\varepsilon', \qquad Q'_{\mathrm{COR}} = Q_{\mathrm{COR}}.$$

□

```
$\mathcal{B}^{\mathrm{O}'}(\mu)$
for $n \in [\mu]$
    $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{GAKE}}(\mathsf{par})$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
for $\mathrm{sID}^* \in \mathcal{S}$:
    if $\textsc{Fresh}(\mathrm{sID}^*) = $ false
        return 0                                    //session not fresh
    if $\textsc{Valid}(\mathrm{sID}^*) = $ false
        return 0                                    //no valid attack
return $[\![\Sigma \in \mathsf{Win}_{\mathsf{OW\text{-}G\text{-}HV}}]\!]$        //break OW-G-HV

$\textsc{Session}_{\mathsf{I}}(i \in [\mu], \mathcal{P} \subseteq [\mu])$
$(\mathrm{sID}, m_i) \xleftarrow{\$} \textsc{Session}'_{\mathsf{I}}(i, \mathcal{P})$
$\mathrm{owner}[\mathrm{sID}] := i$
$\mathrm{peer}[\mathrm{sID}] := \mathcal{P}$
$\mathcal{Q} := \mathcal{P} \cup \{i\}$
$\sigma_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, m_i)$
$\mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}] := (i, (m_i, \sigma_i))$
return $(\mathrm{sID}, m_i)$

$\textsc{Session}_{\mathsf{R}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathsf{S}}], \mathcal{M}_i)$
$(i, \mathcal{P}) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\mathcal{M}_i| \neq |\mathcal{P}|$
    return $\perp$                                  //all peers must have responded
parse $\{(j, (m_j, \sigma_j))\}_{j \in \mathcal{P}} := \mathcal{M}_i$
parse $(i, (m_i, \sigma_i)) =: \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}]$
$\mathcal{Q} := \{i\} \cup \mathcal{P}$
for $j \in \mathcal{P}$
    if $\mathsf{Ver}(\mathsf{pk}_j, m_j, \sigma_j) = 0$
        return $\perp$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}} \mathrm{corrupted}[j]$
if $\mathrm{peerCorrupted}[\mathrm{sID}] = $ false
    for $j \in \mathcal{P}$
        if $\nexists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'_j])$
          $= (j, \mathcal{Q} \setminus \{j\}, (j, m_j, \sigma_j))$
            $\mathsf{AbortSessR} = $ true
            abort
$(\mathrm{sID}, \hat{m}_i) \xleftarrow{\$} \textsc{Session}'_{\mathsf{R}}(\mathrm{sID}, \mathcal{M}_i)$
$\pi_i \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, (\{m_j\}_{j \in \mathcal{Q}}, \hat{m}_i))$
$\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}] := (i, (\hat{m}_i, \pi_i))$
$\mathcal{M}[\mathrm{sID}] := \mathcal{M}_i$
return $\hat{m}_i$

$\textsc{Der}(\mathrm{sID} \in [\mathrm{cnt}_{\mathsf{S}}], \hat{\mathcal{M}}_i)$
if $\mathrm{sKey}[\mathrm{sID}] \neq \perp$
    return $\perp$
$(i, \mathcal{P}) := (\mathrm{owner}[\mathrm{sID}], \mathrm{peer}[\mathrm{sID}])$
if $|\hat{\mathcal{M}}_i| \neq |\mathcal{P}|$ return $\perp$
parse $\{(j, m_j, \sigma_j)\}_{j \in \mathcal{P}} =: \mathcal{M}[\mathrm{sID}]$; $\{(j, \hat{m}_j, \pi_j)\}_{j \in \mathcal{P}} =: \hat{\mathcal{M}}_i$
parse $(i, (m_i, \sigma_i)) =: \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], (i, (\hat{m}_i, \pi_i)) =: \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]$
$\mathcal{Q} := \mathcal{P} \cup \{i\}$
for $k \in \mathcal{P}$
    if $\mathsf{Ver}(\mathsf{pk}_k, (\{m_j\}_{j \in \mathcal{Q}}, \hat{m}_j), \pi_j) = 0$
        return $\perp$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \bigvee_{j \in \mathcal{P}} \mathrm{corrupted}[j]$
if $\mathrm{peerCorrupted}[\mathrm{sID}] = $ false
    for $j \in \mathcal{P}$
        if $\nexists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j], \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'_j], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}'_j])$
      $= (j, \mathcal{Q} \setminus \{j\}, (j, (m_j, \sigma_j)), (j, (\hat{m}_j, \pi_j)))$
            $\mathsf{AbortDer} := $ true
            abort
$\mathrm{ctxt} := (\{\mathsf{pk}_j\}_{j \in \mathcal{Q}}, \mathcal{M}[\mathrm{sID}] \cup \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}], \hat{\mathcal{M}}_i \cup \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}])$
$\textsc{Der}'(\mathrm{sID}, \mathcal{M}[\mathrm{sID}], \hat{\mathcal{M}}_i)$
if $\exists K^*, K : \mathsf{H}[\mathrm{ctxt}, K^*, 1] = K$
    $\mathrm{sKey}[\mathrm{sID}] := K$
elseif $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] = K$
    $\mathrm{sKey}[\mathrm{sID}] := K$
else $K \xleftarrow{\$} \mathcal{K}$
    $\mathsf{H}[\mathrm{ctxt}, K^*, 0] := K$; $\mathrm{sKey}[\mathrm{sID}] := K$
$\hat{\mathcal{M}}[\mathrm{sID}] := \hat{\mathcal{M}}_i$
return $\epsilon$

$\mathsf{H}(\mathrm{ctxt}, K^*)$
$\mathrm{ctxt} := (\mathcal{Q}, \mathcal{M}_{\mathcal{Q}}, \hat{\mathcal{M}}_{\mathcal{Q}})$
if $\mathsf{H}[\mathrm{ctxt}, K^*, \cdot] = K$
    return $K$
$h := \perp$
for $j \in \mathcal{Q}$
    if $\mathsf{H}[\mathrm{ctxt}, \perp, \perp] = K$ and
    $\exists \mathrm{sID}'_j : (\mathrm{owner}[\mathrm{sID}'_j], \mathrm{peer}[\mathrm{sID}'_j]) = (j, \mathcal{Q} \setminus \{j\})$
        $\textsc{Der}'(\mathrm{sID}'_j, \mathcal{M}_{\mathcal{Q}} \setminus \mathsf{Msg}_{\mathsf{I}}[\mathrm{sID}'_j], \hat{\mathcal{M}}_{\mathcal{Q}} \setminus \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}'_j])$
        if $\textsc{KVer}(\mathrm{sID}'_j, K^*) = 1$
            $\Sigma := (\mathrm{sID}'_j, K^*)$               //attack for OW-G-HV
            replace $(\perp, \perp)$ in $\mathsf{H}[\mathrm{ctxt}, \perp, \perp]$
                with $(K^*, 1)$
            return $K$
    else $h := 0$
$K \xleftarrow{\$} \mathcal{K}$
$\mathsf{H}[\mathrm{ctxt}, K^*, h] := K$
return $K$
```

**Fig. 22.** Adversary $\mathcal{B}$ against the $(t'', \varepsilon'', \mu, S', Q_V)$-OW-G-HV of GKE. The OW-G-HV game provides oracles $\mathrm{O}' := \{\textsc{Session}'_{\mathsf{I}}, \textsc{Session}'_{\mathsf{R}}, \textsc{Der}', \textsc{KVer}\}$. The adversary $\mathcal{C}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathsf{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{H}\}$, where $\textsc{Reveal}, \textsc{Corr}, \textsc{Test}$ are defined as in the original IND-G-FS security game .

**Lemma 4.** *For every* PPT *adversary $\mathcal{A}$ running in time $t_{1,2}$ that distinguishes $G_1$ from $G_2$ with probability $\varepsilon_{1,2}$, we can construct an adversary $\mathcal{B}$ against $(t'', \varepsilon'', \mu, S', Q_V)$-*OW-G-HV *security of the underlying group key exchange protocol, where*

$$t_{1,2} \approx t'' \qquad \varepsilon_{1,2} \leq \varepsilon'' \qquad S = S'.$$

*Proof.* Notice that when $b = 1$, the Test oracle always returns a uniformly random key in both $G_2$ and $G_1$; therefore, the only difference between $G_2$ and $G_1$ occurs when

$b = 0$. Hence, we have $\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 1] = \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 1]$, and

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| = \frac{1}{2}\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0]\right|. \tag{15}$$

To bound Equation (15), we construct an adversary $\mathcal{B}$ that breaks the $(t'', \varepsilon'', \mu, S', Q_V)$ -OW-G-HV security of the underlying GKE as in Fig. 22.

Firstly, we remark that the output of $\textsc{Session}'_{\mathsf{I}}$, $\textsc{Session}'_{\mathsf{R}}$ and $\textsc{Der}'$ is distributed identically as in $G_1$. For all sessions that have finished computing a key without making the game abort, all messages must be honestly generated due to the abort conditions introduced in $G_1$, although they may be in a different order and there may be non-matching sessions. Hence, $\textsc{Session}_{\mathsf{I}}$, $\textsc{Session}_{\mathsf{R}}$ and $\textsc{Der}$ are perfectly simulated by $\textsc{Session}'_{\mathsf{I}}$, $\textsc{Session}'_{\mathsf{R}}$ and $\textsc{Der}'$ of the OW-G-HV game and the signing key.

We note that the random oracle H simulated by $\mathcal{B}$ has the same output distribution as in $G_1$. When $b = 0$ and line 72 is executed, we obtain a valid attack (sID, $K^*$) against the OW-G-HV security. In summary, we have

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1 \mid b = 0] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1 \mid b = 0] \le \varepsilon''.$$

□

## Acknowledgements

We thank the anonymous reviewers of CT-RSA 2021 for their many insightful suggestions to improve our paper. We are also grateful to the anonymous reviewers of Journal of Cryptology for their valuable comments to clarify our security model and make our security proofs more understandable. Parts of Pan's work were done, while he was supported by the Research Council of Norway under Project No. 324235.

**Funding** Open access funding provided by NTNU Norwegian University of Science and Technology (incl St. Olavs Hospital - Trondheim University Hospital).

# Appendices

## A Security of Schnorr in the Generic Group Model

We show the StCorrCMA security of Schnorr's signature scheme in the generic group model (GGM) which has been formally stated in Theorem 1. This section also gives a proof of the theorem.

We proceed as follows: Firstly, we propose a variant of the IDLOG assumption [32], CorrIDLOG, by introducing an additional corruption oracle. Secondly, by using a slightly different version of [32, Lemma 5.8], we prove that Schnorr's signature is tightly StCorrCMA-secure based on the CorrIDLOG assumption. Finally, we prove the hardness of CorrIDLOG.

Note that in [32] it has been proven that IDLOG tightly implies the multiuser security of Schnorr without corruptions, which does not necessary give us tight multiuser security with corruptions. However, our new CorrIDLOG assumption tightly implies the multiuser security of Schnorr *with* corruptions. We believe that our CorrIDLOG assumption is of independent interest.

Let par $= (p, g, \mathbb{G})$ be a set of system parameters. The CorrIDLOG assumption is defined as follow:

**Definition 7.** (CorrIDLOG) The CorrIDLOG problem is $(t, \varepsilon, \mu, Q_{\mathrm{CH}}, Q_{\mathrm{DL}})$-hard in par, if for all adversaries $\mathcal{A}$ interacting with $\mu$ users, running in time at most $t$ and making at most $Q_{\mathrm{CH}}$ queries to the challenge oracle Ch and $Q_{\mathrm{DL}}$ queries to the corruption oracle Dl, we have:

$$\Pr\left[ g^s \in \{X_i^{h_j} \cdot R_j \mid i \notin \mathcal{L}_{\mathcal{C}} \wedge j \in [Q_{\mathrm{CH}}]\} \;\middle|\; \begin{array}{l} \textbf{for } i \in [\mu] \\ \quad x_i \leftarrow_\$ \mathbb{Z}_p;\ X_i := g^{x_i} \\ s \leftarrow_\$ \mathcal{A}^{\textsc{Ch}(\cdot),\textsc{Dl}(\cdot)}(\{X_i\}_{i\in[\mu]}) \end{array} \right] \leq \varepsilon,$$

where on the $j$-th challenge query $\textsc{Ch}(R_j \in \mathbb{G})$ $(j \in [Q_{\mathrm{CH}}])$ Ch returns $h_j \leftarrow_\$ \mathbb{Z}_p$ to $\mathcal{A}$, and on a corruption query $\textsc{Dl}(i)$ for $i \in [\mu]$, Dl returns $x_i$ to $\mathcal{A}$ and adds $i$ into the corruption list $\mathcal{L}_{\mathcal{C}}$ (namely, $\mathcal{L}_{\mathcal{C}} := \mathcal{L}_{\mathcal{C}} \cup \{i\}$).

Before proving the hardness of CorrIDLOG in the GGM, Lemma 5 shows that CorrIDLOG tightly implies the StCorrCMA security of Schnorr in the random oracle model (without using the GGM). Note that this lemma does not contradict the impossibility result of [22], since our assumption is interactive. In fact, following the framework in [32, Section 3], one can easily prove that the standard DLOG assumption non-tightly implies the CorrIDLOG assumption in the standard model.

**Lemma 5.** (CorrIDLOG $\xrightarrow{tight}$ StCorrCMA) *If* CorrIDLOG *is* $(t, \varepsilon, \mu, Q_{\mathrm{CH}}, Q_{\mathrm{DL}})$*-hard in* par*, then Schnorr's signature* Schnorr *is* $(t', \varepsilon', \mu, Q_s, Q_{\mathrm{DL}}, Q_{\mathsf{H}})$-StCorrCMA *in the programmable random oracle model, where*

$$t' \approx t, \quad \varepsilon' \leq \varepsilon + \frac{Q_{\mathsf{H}} Q_s + 1}{p}, \quad Q_{\mathrm{CH}} = Q_{\mathsf{H}}.$$

*Proof.* This proof is straightforward by [32], but for completeness we prove it in details here. Let $\mathcal{A}$ be an adversary against StCorrCMA security. We construct $\mathcal{B}$ against CorrIDLOG (Fig. 23).

Firstly, we argue that $\mathcal{B}$ perfectly simulates the experiment StCorrCMA unless $\mathcal{B}$ aborts in line 14, namely $(R, m)$ collides with a previous hash query. Since $R$ is distributed uniformly at random, by the union bound the probability that $\mathcal{B}$ aborts in line 14 is bounded by $Q_{\mathsf{H}} Q_s / p$.

Secondly, we show that $\mathcal{B}$'s forgery $s^*$ is a valid CorrIDLOG forgery. Given the $(h^*, s^*)$ from $\mathcal{A}$, we have $R^* = g^{s^*} \cdot X_{i^*}^{-h^*}$ and $\textsc{Hash}(R^*, m^*) = h^*$. We make our argument in the following steps:

1. With high probability, there exists $((R^*, m^*), h^*) \in \mathcal{L}_{\mathsf{H}}$. Otherwise, it means $\mathcal{A}$ was able to guess the hash value of $(R^*, m^*)$ without querying Hash. This event is bounded by $1/p$.
2. If $((R^*, m^*), h^*)$ was added to $\mathcal{L}_{\mathsf{H}}$ by the signing oracle Sign, then Sign must have chosen an $s'$ such that $g^{s'} \cdot X_{i^*}^{-h^*} = R^* = g^{s^*} \cdot X_{i^*}^{-h^*}$, which means $s' = s^*$. However, if $(h^*, s^*)$ from $\mathcal{A}$ is a valid StCorrCMA forgery, then $s' = s^*$ cannot happen.

```
$\mathcal{B}(\{X_i\}_{\in[\mu]})$:                //CorrIDLOG adversary
for $i \in [\mu]$
    $\mathsf{pk}_i := X_i$
$(i^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Corr},\textsc{Sign}}(\{\mathsf{pk}_i\}_{i\in[\mu]})$
parse $(h^*, s^*) =: \sigma^*$
return $s^*$
$\textsc{Hash}(R, m)$ :
if $\exists h : ((R, m), h) \in \mathcal{L}_{\mathsf{H}}$
    return $h$
$h \xleftarrow{\$} \textsc{Ch}(R)$
$\mathcal{L}_{\mathsf{H}} := \mathcal{L}_{\mathsf{H}} \cup \{((R, m), h)\}$
return $h$

$\textsc{Sign}(i, m)$ :
parse $X_i =: \mathsf{pk}_i$
$s, h \xleftarrow{\$} \mathbb{Z}_p$
$R := g^s \cdot X_i^{-h}$
if $\exists h' : ((R, m), h') \in \mathcal{L}_{\mathsf{H}}$
    abort
$\mathcal{L}_{\mathsf{H}} := \mathcal{L}_{\mathsf{H}} \cup \{((R, m), h)\}$
$\sigma := (h, s)$
$\mathcal{L}_{\mathcal{S}} := \mathcal{L}_{\mathcal{S}} \cup \{(i, m, \sigma)\}$
return
$\textsc{Corr}(i)$ :
return $\textsc{Dl}(X_i)$
```

**Fig. 23.** Adversary $\mathcal{B}$ against the CorrIDLOG assumption .

3. Now $((R^*, m^*), h^*)$ can only be added to $\mathcal{L}_{\mathsf{H}}$ by the hashing oracle Hash. This is equivalent to $R^* = R_j$ and $h^* = h_j$ for some $j \in [Q_{\mathbb{G}}]$. Thus $g^{s^*} = R^* \cdot X_{i^*}^{h^*} = R_j \cdot X_{i^*}^{h_j}$, and $s^*$ is a valid attack in the CorrIDLOG security game.

This concludes the proof of Lemma 5. □

Combining Lemma 5 and Lemma 6 (namely, the generic hardness of CorrIDLOG), we can conclude the StCorrCMA security of Schnorr's signature in Theorem 1.

## *A.1 Generic Hardness of* CorrIDLOG

Generic Group Model. In the GGM for prime-order groups $\mathbb{G}$ [37,45], operations in $\mathbb{G}$ can only be carried out via black-box access to the group oracle $\mathrm{O}_{\mathbb{G}}(\cdot, \cdot)$, and adversaries only get non-random handles of the group elements. Since groups $(\mathbb{G}, \cdot)$ and $(\mathbb{Z}_p, +)$ are isomorphic, every element in $\mathbb{G}$ is internally identified as a $\mathbb{Z}_p$ element. To consistently simulate the group operations, the simulator maintains a list $\mathcal{L}_{\mathbb{G}}$ internally and a counter cnt that keeps track of the number of entries in $\mathcal{L}_{\mathbb{G}}$. $\mathcal{L}_{\mathbb{G}}$ contains entries of the form $(z(\boldsymbol{x}), C_z)$, where $z(\boldsymbol{x}) \in \mathbb{Z}_p[\boldsymbol{x}]$ represents a group element and the positive integer $C_z$ is its counter.
We assume $\mathcal{A}$ can make at most $Q_{\mathbb{G}}$ queries to $\mathrm{O}_{\mathbb{G}}$.

**Lemma 6.** *For any adversary $\mathcal{A}$ that $(t, \varepsilon, \mu, Q_{\mathrm{CH}}, Q_{\mathrm{DL}})$-breaks the* CorrIDLOG *assumption, we have*

$$\varepsilon \leq \frac{(Q_{\mathbb{G}} + \mu + 1)^2}{2p} + \frac{(\mu - Q_{\mathrm{DL}})}{p}.$$

We recall the Schwartz–Zippel Lemma that is useful for proving Lemma 6.

**Lemma 7.** (Schwartz–Zippel Lemma) *Let $f(x_1, \ldots, x_n)$ be a nonzero multivariant polynomial of maximum degree $d \geq 0$ over a field $\mathbb{F}$. Let $\mathcal{S}$ be a finite subset of $\mathbb{F}$ and $a_1, \ldots, a_n$ be chosen uniformly at random from $\mathcal{S}$. Then, we have*

$$\Pr[f(a_1, \ldots, a_n) = 0] \leq \frac{d}{|\mathcal{S}|}.$$

*Proof of Lemma 6.* $\mathcal{A}$ is an adversary against the CorrIDLOG assumption. $\mathcal{B}$ is simulator that simulates the CorrIDLOG security game in the GGM and interacts with $\mathcal{A}$. The simulation is described in Fig. 24

$\mathcal{B}$ simulates the CorrIDLOG game in a symbolic way using degree-1 polynomials. The internal list $\mathcal{L}_{\mathbb{G}}$ stores the entries of the form $(f(\boldsymbol{x}), C_{f(\boldsymbol{x})})$, where $f(\boldsymbol{x}) \in \mathbb{Z}_p[x_1, \ldots, x_\mu]$ is a degree-1 polynomial and $C_{f(\boldsymbol{x})} \in \mathbb{N}$ identifies which entry it is. $\mathcal{B}$ also keeps track of the size of $\mathcal{L}_{\mathbb{G}}$ by cnt. After $\mathcal{A}$ outputs an attack, all the variables $(x_1 \ldots x_\mu)$ will be assigned a value $(a_1, \ldots, a_\mu) \leftarrow_{\$} \mathbb{Z}_p^\mu$ chosen uniformly at random.

```
$\mathcal{B}$:                                                         //CorrIDLOG in the GGM
$\mathcal{L}_{\mathbb{G}} := \{(1, C_1 := 1)\}$
for $i \in [\mu]$
    $a_i \xleftarrow{\$} \mathbb{Z}_p$
    $C_{x_i} := i + 1$
    $\mathcal{L}_{\mathbb{G}} := \mathcal{L}_{\mathbb{G}} \cup \{(x_i, C_{x_i})\}$
    $\mathsf{pk}_i := C_{x_i}$
cnt $:= \mu + 1$                                                        //tracking the size of $\mathcal{L}_{\mathbb{G}}$
$\vec{x} := (x_1, \ldots, x_\mu)$
$\vec{a} := (a_1, \ldots, a_\mu)$
$s^* \xleftarrow{\$} \mathcal{A}^{\mathrm{O}}(\{\mathsf{pk}_i\}_{i \in [\mu]})$
if $\exists (f_1(\vec{x}), C_1), (f_2(\vec{x}), C_2) \in \mathcal{L}_{\mathbb{G}}$ :
$f_1(\vec{x}) \neq f_2(\vec{x}) \wedge f_1(\vec{a}) = f_2(\vec{a})$
    $\mathsf{Bad}_{\mathbb{G}} := 1$; abort
for $(C^*, h^*) \in \mathcal{L}_{\mathrm{CH}}$
    fetch $(r^*(\vec{x}), C^*) \in \mathcal{L}_{\mathbb{G}}$
    if $\exists i^* \in [\mathrm{cnt}] \setminus \mathcal{L}_{\mathcal{C}} : s^* = a_{i^*} \cdot h^* + r^*(\vec{a})$
        return 1
return 0

$\mathrm{O}_{\mathbb{G}}(C_1, C_2)$:                                    //Group operation
if $(C_1, C_2) \notin [\mathrm{cnt}]^2$
    return $\perp$
fetch $(f_1(\vec{x}), C_1), (f_2(\vec{x}), C_2) \in \mathcal{L}_{\mathbb{G}}$
$z(\vec{x}) := f_1(\vec{x}) + f_2(\vec{x})$
if $\exists C_z \in [\mathrm{cnt}] : (z(\vec{x}), C_z) \in \mathcal{L}_{\mathbb{G}}$
    return $C_z$
else
    cnt ++
    $C_z :=$ cnt
    $\mathcal{L}_{\mathbb{G}} := \mathcal{L}_{\mathbb{G}} \cup \{(z(\vec{x}), C_z)\}$
    return $C_z$

CHALL($C$):                                        //$k$-th query ($k \in [Q_{\mathrm{CH}}]$)
if $C \notin [\mathrm{cnt}]$
    return $\perp$
else
    $h_k \xleftarrow{\$} \mathbb{Z}_p$
    $\mathcal{L}_{\mathrm{CH}} := \mathcal{L}_{\mathrm{CH}} \cup \{(C, h_k)\}$
    return $h_k$

DL($i$):                                                   //Corruption oracle
$\mathcal{L}_{\mathcal{C}} := \mathcal{L}_{\mathcal{C}} \cup \{i\}$
return $a_i$
```

**Fig. 24.** $\mathcal{B}$ simulates the CorrIDLOG security game in the GGM and interacts with $\mathcal{A}$. The adversary $\mathcal{A}$ has access to the oracles $\mathrm{O} := (\mathrm{O}_{\mathbb{G}}, \textsc{Chall}, \textsc{Dl})$ .

We remark that $\mathcal{B}$ perfectly simulates the CorrIDLOG security game in the GGM if none of the distinct polynomials $z_i$ and $z_j$ stored in $\mathcal{L}_{\mathbb{G}}$ collide when evaluating on the random vector $\boldsymbol{a}$ over $\mathbb{Z}_p$. Applying the union bound over all pairs of distinct polynomials in $\mathcal{L}_{\mathbb{G}}$, we have:

$$\Pr[\mathsf{Bad}_{\mathbb{G}}] := \Pr_{\boldsymbol{a} \leftarrow_{\$} \mathbb{Z}_p^\mu}[\exists (i, j) \in [\mathrm{cnt}]^2 : z_i(\boldsymbol{x}) \neq z_j(\boldsymbol{x}) \wedge z_i(\boldsymbol{a}) = z_j(\boldsymbol{a})]$$
$$\leq \binom{Q_{\mathbb{G}} + \mu + 1}{2} \cdot \frac{1}{p} \leq \frac{(Q_{\mathbb{G}} + \mu + 1)^2}{2p},$$

where the factor $\frac{1}{p}$ comes from Lemma 7 and the fact that $\mathcal{L}_{\mathbb{G}}$ contains only degree-1 polynomials and $(a_1, \ldots, a_\mu)$ is chosen uniformly at random from $\mathbb{Z}_p^\mu$.
We give an upper bound of the success probability of $\mathcal{A}$ as follows:

$$\varepsilon \leq \Pr[\mathsf{Bad}_{\mathbb{G}}] + \Pr_{\boldsymbol{a} \leftarrow_{\$} \mathbb{Z}_p^\mu}[\exists i^* \in [\mu] \setminus \mathcal{L}_{\mathcal{C}} : s^* = a_{i^*} h^* + r^*(\boldsymbol{a})]$$
$$\leq \frac{(Q_{\mathbb{G}} + \mu + 1)^2}{2p} + \frac{(\mu - Q_{\mathrm{DL}})}{p}.$$

The second term $\frac{(\mu - Q_{\mathrm{DL}})}{p}$ comes from the fact that for each $i^* \in [\mu] \setminus \mathcal{L}_{\mathcal{C}}$ $\mathcal{A}$ has no information about $x_{i^*}$. Thus for a fixed $i^* \in [\mu] \setminus \mathcal{L}_{\mathcal{C}}$, we get that $x_{i^*} h^* + r^*(\boldsymbol{x}) - s^*$ is a degree-1 polynomial, and by Lemma 7

$$\Pr_{\boldsymbol{a} \leftarrow_{\$} \mathbb{Z}_p^\mu}[s^* = a_{i^*} h^* + r^*(\boldsymbol{a})] \leq \frac{1}{p}.$$

By the union bound, we have

$$\Pr_{\boldsymbol{a} \leftarrow_{\$} \mathbb{Z}_p^\mu}[\exists i^* \in [\mu] \setminus \mathcal{L}_{\mathcal{C}} : s^* = a_{i^*} h^* + r^*(\boldsymbol{a})] \leq \frac{\mu - Q_{\mathrm{DL}}}{p}.$$

□

## References

[1] M. Abdalla, M. Bellare, P. Rogaway, The oracle Diffie-Hellman assumptions and an analysis of DHIES, in Naccache, D. (ed.) *CT-RSA 2001. LNCS*, vol. 2020 (Springer, Heidelberg, 2001), pp. 143–158

[2] C. Bader, D. Hofheinz, T. Jager, E. Kiltz, Y. Li, Tightly-secure authenticated key exchange, in Dodis, Y., Nielsen, J.B. (eds.) *TCC 2015, Part I. LNCS*, vol. 9014 (Springer, Heidelberg, 2015), pp. 629–658

[3] M. Bellare, W. Dai, The multi-base discrete logarithm problem: Tight reductions and non-rewinding proofs for Schnorr identification and signatures, in Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) *INDOCRYPT 2020. LNCS*, vol. 12578 (Springer, Heidelberg, 2020), pp. 529–552

[4] M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) *ACM CCS 93* (ACM Press, 1993), pp. 62–73

[5] M. Bellare, P. Rogaway, Entity authentication and key distribution, in Stinson, D.R. (ed.) *CRYPTO'93. LNCS*, vol. 773 (Springer, Heidelberg, 1994), pp. 232–249

[6] M. Bellare, P. Rogaway, The security of triple encryption and a framework for code-based game-playing proofs, in Vaudenay, S. (ed.) *EUROCRYPT 2006. LNCS*, vol. 4004 (Springer, Heidelberg, 2006), pp. 409–426

[7] F. Bergsma, T. Jager, J. Schwenk, One-round key exchange with strong security: An efficient and generic construction in the standard model, in Katz, J. (ed.) *PKC 2015. LNCS*, vol. 9020 (Springer, Heidelberg, 2015), pp. 477–494

[8] D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.Y. Yang, High-speed high-security signatures, in Preneel, B., Takagi, T. (eds.) *CHES 2011. LNCS*, vol. 6917 (Springer, Heidelberg, 2011), pp. 124–142

[9] E. Bresson, O. Chevassut, D. Pointcheval, Provably authenticated group Diffie-Hellman key exchange—the dynamic case, in: Boyd, C. (ed.) *ASIACRYPT 2001. LNCS*, vol. 2248 (Springer, Heidelberg, 2001), pp. 290–309

[10] E. Bresson, O. Chevassut, D. Pointcheval, Dynamic group Diffie-Hellman key exchange under standard assumptions, in Knudsen, L.R. (ed.) *EUROCRYPT 2002. LNCS*, vol. 2332 (Springer, Heidelberg, 2002), pp. 321–336

[11] E. Bresson, O. Chevassut, D. Pointcheval, J.J. Quisquater, Provably authenticated group Diffie-Hellman key exchange, in Reiter, M.K., Samarati, P. (eds.) *ACM CCS 2001* (ACM Press, 2001), pp. 255–264

[12] M. Burmester, Y. Desmedt, A secure and efficient conference key distribution system (extended abstract), in: Santis, A.D. (ed.) *EUROCRYPT'94. LNCS*, vol. 950 (Springer, Heidelberg, 1995), pp. 275–286

[13] D. Cash, E. Kiltz, V. Shoup, The twin Diffie-Hellman problem and applications, in Smart, N.P. (ed.) *EUROCRYPT 2008. LNCS*, vol. 4965 (Springer, Heidelberg, 2008), pp. 127–145

[14] K. Cohn-Gordon, C. Cremers, K. Gjøsteen, H. Jacobsen, T. Jager, Highly efficient key exchange protocols with optimal tightness, in Boldyreva, A., Micciancio, D. (eds.) *CRYPTO 2019, Part III.* LNCS, vol. 11694 (Springer, Heidelberg, 2019), pp. 767–797

[15] H. Davis, F. Günther, Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols, in Sako, K., Tippenhauer, N.O. (eds.) *ACNS 21, Part II. LNCS*, vol. 12727 (Springer, Heidelberg, 2021), pp. 448–479

[16] C. de Saint Guilhem, M. Fischlin, B. Warinschi, Authentication in key-exchange: Definitions, relations and composition, in: Jia, L., Küsters, R. (eds.) *CSF 2020 Computer Security Foundations Symposium* (IEEE Computer Society Press, 2020), pp. 288–303

[17] D. Diemert, K. Gellert, T. Jager, L. Lyu, More efficient digital signatures with tight multi-user security, in Garay, J. (ed.) *PKC 2021, Part II. LNCS*, vol. 12711 (Springer, Heidelberg, 2021), pp. 1–31

[18] D. Diemert, T. Jager, On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments, *J. Cryptol.* **34**(3), 30 (2021)

[19] W. Diffie, M.E. Hellman, New directions in cryptography, *IEEE Trans. Inf. Theory* **22**(6), 644–654 (1976)

[20] W. Diffie, P.C. van Oorschot, M.J. Wiener, Authentication and authenticated key exchanges, *Designs Codes Cryptography* **2**(2), 107–125 (1992)

[21] M. Fischlin, F. Günther, B. Schmidt, B. Warinschi, Key confirmation in key exchange: A formal treatment and implications for TLS 1.3, in *2016 IEEE Symposium on Security and Privacy* (IEEE Computer Society Press, 2016), pp. 452–469

[22] N. Fleischhacker, T. Jager, D. Schröder, On tight security proofs for Schnorr signatures, in P. Sarkar, T. Iwata (eds.) *ASIACRYPT 2014, Part I. LNCS*, vol. 8873 (Springer, Heidelberg, 2014), pp. 512–531

[23] S.D. Galbraith, J. Malone-Lee, N.P. Smart, Public key signatures in the multi-user setting, *Inf. Process. Lett.* **83**(5), 263–266 (2002). https://doi.org/10.1016/S0020-0190(01)00338-6

[24] K. Gjøsteen, T. Jager, Practical and tightly-secure digital signatures and authenticated key exchange, in Shacham, H., Boldyreva, A. (eds.) *CRYPTO 2018, Part II. LNCS*, vol. 10992 (Springer, Heidelberg, 2018), pp. 95–125

[25] M.C. Gorantla, C. Boyd, J.M. González Nieto, Modeling key compromise impersonation attacks on group key exchange protocols, in Jarecki, S., Tsudik, G. (eds.) *PKC 2009. LNCS*, vol. 5443 (Springer, Heidelberg, 2009), pp. 105–123

[26] D. Harkins, D. Carrel, The internet key exchange (IKE). RFC 2409 (1998). https://www.ietf.org/rfc/rfc2409.txt

[27] D. Hofheinz, E. Kiltz, The group of signed quadratic residues and applications, in Halevi, S. (ed.) *CRYPTO 2009. LNCS*, vol. 5677 (Springer, Heidelberg, 2009), pp. 637–653

[28] T. Jager, E. Kiltz, D. Riepel, S. Schäge, Tightly-Secure Authenticated Key Exchange, Revisited. In: Eurocrypt 2021 (2021). https://ia.cr/2020/1279

[29] T. Jager, F. Kohlar, S. Schäge, J. Schwenk, On the security of TLS-DHE in the standard model, in Safavi-Naini, R., Canetti, R. (eds.) *CRYPTO 2012. LNCS*, vol. 7417 (Springer, Heidelberg, 2012), pp. 273–293

[30] T. Jager, F. Kohlar, S. Schäge, J. Schwenk, Authenticated confidential channel establishment and the security of TLS-DHE, *J. Cryptol.* **30**(4), 1276–1324 (2017)

[31] J. Katz, M. Yung, Scalable protocols for authenticated group key exchange, in Boneh, D. (ed.) *CRYPTO 2003. LNCS*, vol. 2729 (Springer, Heidelberg, 2003), pp. 110–125

[32] E. Kiltz, D. Masny, J. Pan, Optimal security proofs for signatures from identification schemes, in Robshaw, M., Katz, J. (eds.) *CRYPTO 2016, Part II. LNCS*, vol. 9815 (Springer, Heidelberg, 2016), pp. 33–61

[33] H. Krawczyk, SIGMA: The “SIGn-and-MAc” approach to authenticated Diffie-Hellman and its use in the IKE protocols, in Boneh, D. (ed.) *CRYPTO 2003. LNCS*, vol. 2729 (Springer, Heidelberg, 2003), pp. 400–425

[34] B.A. LaMacchia, K. Lauter, A. Mityagin, Stronger security of authenticated key exchange, in Susilo, W., Liu, J.K., Mu, Y. (eds.) *ProvSec 2007. LNCS*, vol. 4784 (Springer, Heidelberg, 2007), pp. 1–16

[35] Y. Li, S. Schäge, No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial, in Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) *ACM CCS 2017* (ACM Press, 2017), pp. 1343–1360

[36] X. Liu, S. Liu, D. Gu, J. Weng, Two-pass authenticated key exchange with explicit authentication and tight security, in Moriai, S., Wang, H. (eds.) *ASIACRYPT 2020, Part II. LNCS*, vol. 12492 (Springer, Heidelberg, 2020), pp. 785–814

[37] U.M. Maurer, Abstract models of computation in cryptography (invited paper), in Smart, N.P. (ed.) *10th IMA International Conference on Cryptography and Coding. LNCS*, vol. 3796 (Springer, Heidelberg, 2005), pp. 1–12

[38] J. Pan, C. Qian, M. Ringerud, Signed diffie-hellman key exchange with tight security, in Paterson, K.G. (ed.) *CT-RSA 2021. LNCS*, vol. 12704 (Springer, Heidelberg, 2021), pp. 201–226

[39] J. Pan, M. Ringerud, Signatures with tight multi-user security from search assumptions, in L. Chen, N. Li, K. Liang, S.A. Schneider (eds.) *ESORICS 2020, Part II. LNCS*, vol. 12309 (Springer, Heidelberg, 2020), pp. 485–504

[40] PKCS #1: RSA cryptography standard. RSA Data Security, Inc. (1991)

[41] B. Poettering, P. Rösler, J. Schwenk, D. Stebila, SoK: Game-based security models for group key exchange, in Paterson, K.G. (ed.) *CT-RSA 2021. LNCS*, vol. 12704 (Springer, Heidelberg, 2021), pp. 148–176

[42] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard (2018). https://tools.ietf.org/html/rfc8446

[43] P. Rösler, C. Mainka, J. Schwenk, More is less: On the end-to-end security of group chats in signal, whatsapp, and threema, in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 415–429 (2018)

[44] C.P. Schnorr, Efficient signature generation by smart cards *J. Cryptol.* **4**(3), 161–174 (1991)

[45] V. Shoup, Lower bounds for discrete logarithms and related problems, in Fumy, W. (ed.) *EUROCRYPT'97. LNCS*, vol. 1233 (Springer, Heidelberg, 1997), pp. 256–266

[46] Y. Xiao, R. Zhang, H. Ma, Tightly secure two-pass authenticated key exchange protocol in the CK model, in Jarecki, S. (ed.) *CT-RSA 2020. LNCS*, vol. 12006 (Springer, Heidelberg, 2020), pp. 171–198