Key Exchange with Tight (Full) Forward Secrecy via Key Confirmation

Jiaxin Pan[1(✉)], Doreen Riepel[2], and Runzhi Zeng[3]

[1] University of Kassel, Kassel, Germany
jiaxin.pan@uni-kassel.de
[2] University of California San Diego, La Jolla, USA
driepel@ucsd.edu
[3] Norwegian University of Science and Technology, Trondheim, Norway
runzhi.zeng@ntnu.no

Abstract. Weak forward secrecy (wFS) of authenticated key exchange (AKE) protocols is a passive variant of (full) forward secrecy (FS). A natural mechanism to upgrade from wFS to FS is the use of key confirmation messages which compute a message authentication code (MAC) over the transcript. Unfortunately, Gellert, Gjøsteen, Jacobson and Jager (GGJJ, CRYPTO 2023) show that this mechanism inherently incurs a loss proportional to the number of users, leading to an overall non-tight reduction, even if wFS was established using a tight reduction.

Inspired by GGJJ, we propose a new notion, called one-way verifiable weak forward secrecy (OW-VwFS), and prove that OW-VwFS can be transformed *tightly* to FS using key confirmation in the random oracle model (ROM). To implement our generic transformation, we show that several tightly wFS AKE protocols additionally satisfy our OW-VwFS notion tightly. We highlight that using the recent lattice-based protocol from Pan, Wagner, and Zeng (CRYPTO 2023) can give us the first lattice-based tightly FS AKE via key confirmation in the classical random oracle model. Besides this, we also obtain a Decisional-Diffie-Hellman-based protocol that is considerably more efficient than the previous ones.

Finally, we lift our study on FS via key confirmation to the quantum random oracle model (QROM). While our security reduction is overall non-tight, it matches the best existing bound for wFS in the QROM (Pan, Wagner, and Zeng, ASIACRYPT 2023), namely, it is square-root- and session-tight. Our analysis is in the multi-challenge setting, and it is more realistic than the single-challenge setting as in Pan et al.

Keywords: Authenticated key exchange · forward secrecy · key confirmation · tight security · (quantum) random oracles

1 Introduction

Forward secrecy (FS) is an essential security requirement for authenticated key exchange (AKE) protocols. It states that even if an active adversary corrupts a

M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14657, pp. 59–89, 2024.
https://doi.org/10.1007/978-3-031-58754-2_3

user's long-term secret key, all session keys agreed before should remain secret to the adversary. A weaker form of FS is called weak FS (wFS), where an adversary is not allowed to perform active attacks, namely, it does not actively interfere with the protocol transcripts of the session that it attacks.

Key confirmation is simple and arguably the most efficient way in achieving FS and has been used in many works, e.g., [10,15,24]. Essentially, it generically transforms an AKE protocol with wFS to FS. More precisely, two parties firstly run a wFS AKE protocol to agree on a session key k, and then they exchange key confirmation messages derived from k. These messages are usually message authentication codes (MAC) on the protocol transcripts using k as the MAC key. Apart from key confirmation, one can use a digital signature scheme to sign a passively secure key exchange protocol as in the signed Diffie-Hellman protocol [19,27] to provide FS. Considering that using a MAC or hash function is much more efficient than digital signatures, the signature-based approach is often inefficient and less desirable.

Security Models for AKE. Defining the security for AKE protocols is a complex task, and there are many different security models for AKE (e.g., [3,7,25]). In this paper, we consider active adversaries that can modify, drop, or inject some messages. Moreover, they may adaptively corrupt users' long-term secret keys via Corr oracle and reveal session keys via Reveal oracle. Some of the models even allow adversaries to learn ephemeral states (which are usually randomness in generating protocol messages) via Rev-State oracle. We formalize key secrecy via Test, where an adversary $\mathcal{A}$ chooses a *fresh* session, receives either a real or random key for it, and shall distinguish between the two. We consider the single-bit guessing, multi-challenge security, namely, $\mathcal{A}$ can query Test multiple times and each time Test responds using the same bit in deciding real or random. Composability for this notion was initially proven for password-based key exchange [1], and we refer to [21] for further discussion on why this is the realistic and meaningful notion. For forward secrecy, keys of these Test-sessions must be computed before Corr is queried to either parties of a Test-session. Depending on the type of forward secrecy, freshness is defined differently. If it is wFS, then $\mathcal{A}$ must perform only passive attacks on this fresh session. Otherwise, $\mathcal{A}$ can perform active attacks, for instance, modify or inject some messages.

Security Loss for FS via Key Confirmation. The complexity of AKE models makes it challenging to prove security of an AKE protocol, in particular, giving tight security proofs for AKE. The security of modern cryptographic protocols is often proven by reductions. A reduction $\mathcal{R}$ uses an adversary $\mathcal{A}$ against protocol Π to break the security of the underlying primitive P. By doing so, we can conclude the concrete security bound, $\varepsilon_{\mathcal{A}} \leq \ell \cdot \varepsilon_{\mathcal{R}}$, where $\varepsilon_{\mathcal{A}}$ and $\varepsilon_{\mathcal{R}}$ are the success probability of $\mathcal{A}$ and $\mathcal{R}$, respectively. ℓ is called the security loss. Assuming $\mathcal{A}$ and $\mathcal{R}$ have roughly the same running time, if ℓ is a small constant, we say protocol Π has tight security, and non-tight security, otherwise. A tight security reduction is highly desirable, since it allows protocols to be instantiated with optimal parameters without compensation for the security loss.

A natural question to ask is whether the key confirmation approach preserves the tightness of the underlying wFS AKE. Due to its high efficiency, it would be ideal to have an affirmative answer to this question, since it means that we do not need to increase the security parameter of the wFS AKE to compensate any security loss.

Intuitively, there should not be a tightness loss when going from wFS to FS, which was even falsely claimed by the work of Cohn-Gordon et al. [10] previously. At CRYPTO 2023, Gellert, Gjøsteen, Jacobsen, and Jager (GGJJ) [18] identified a flaw in [10] and proposed a fix by using a selective variant of wFS (called selective key secrecy in [18]). The selective wFS is essentially the same as wFS, except that an adversary $\mathcal{A}$ has to select a user of which $\mathcal{A}$ will not corrupt the long-term secret key. Unfortunately, when we construct a reduction $\mathcal{R}$ to prove FS based on this selective wFS, $\mathcal{R}$ has to guess the non-corrupted user, which leads to a security loss of $O(\mu)$ where μ is the maximal number of users. This security loss is proven to be inherent (and thus optimal) in [18] when starting from a wFS AKE with key indistinguishability.

However, a linear loss in the number of users is undesirable, since in the real world the number of users can be massive. According to the impossibility result in [10], it seems inherent to have this security loss. Hence, it motivates us to propose a different modularization that potentially requires strong security for the underlying wFS AKE in achieving tight FS.

1.1 Our Contribution I: Tight Forward Secrecy via Key Confirmation

We revise the security proof for the wFS-to-FS transformation.

Tight FS from Verifiable wFS. We propose a new variant of wFS, called One-Wayness against key Verification attacks and weak Forward Secrecy (OW-VwFS). In the OW-VwFS security game, an adversary has the same capability as in the usual wFS game, but additionally it can verify whether a session key is the valid one of a particular session. Hence, the adversary capability of OW-VwFS is stronger than that of wFS and it is the main reason why we bypass the optimality result from Gellert et al. [18]. In terms of security goals, OW-VwFS is weaker than wFS, namely, OW-VwFS only requires an adversary cannot compute the session key of a fresh session, while wFS requires a session key to be indistinguishable from a random key.

Using key confirmation, we prove that OW-VwFS *tightly* implies FS in the random oracle model. Our transformation is the same as the standard wFS-to-FS transformation, but ours preserves the tightness of the underlying OW-VwFS protocol, and it enables tight FS in contrast to the selective notion in [18]. An important consequence of our work is that the future AKE design can aim at OW-VwFS, since its transformation to FS is the same as the standard wFS-to-FS one, but tightness-preserving. Moreover, our analysis considers security against (ephemeral) state reveals. Such a strong form of attacks was not considered in the work of Gellert et al. [18], which is why we bypass their impossibility.

Constructing (Tightly) Verifiable wFS. Furthermore, we show that several tightly wFS protocols satisfy our new OW-VwFS notion tightly, in particular, the lattice-based protocol of Pan, Wagner, and Zeng [30][1]. Subsequently, this yields the *first* AKE protocol with tight FS from lattices.

Essentially, we show that a One-Way Checkable against Chosen-Ciphertext Attacks (OW-ChCCA) [30] secure key encapsulation mechanism (KEM) tightly implies a OW-VwFS AKE protocol. Once again, our analysis allows adversaries to reveal ephemeral state in the AKE protocol. Roughly speaking, the OW-ChCCA game is a multi-user, multi-challenge variant of the standard IND-CCA game: Besides the oracles provided by the standard IND-CCA security, it allows adversaries to corrupt some of the user's decryption keys and decrypt some of the challenge ciphertexts, and, most importantly, it allows an adversary to check if a key is valid with respect to a ciphertext. The adversary goal is to invert a fresh challenge ciphertext. As shown in [30], we can construct OW-ChCCA KEM tightly from the Decisional Diffie-Hellman (DDH) and Learning-With-Errors (LWE) assumptions, respectively. As a technical note, our proof requires only a slightly weaker version of OW-ChCCA, where adversaries are not allowed to ask for a decryption, but to verify whether a ciphertext can be decapsulated.

Efficiency Comparison among DDH-based Protocols. Besides having the first lattice-based AKE with tight FS, we also obtain the most efficient DDH-based protocol against state reveal attacks. In Table 1, we compare efficiency among well-known DDH-based AKE with tight or "optimal" tight FS (namely, with security loss $O(\mu)$) to show the practicality of our work. Our estimation focuses on communication and computation complexity for both parties to agree on a session key. For computation complexity, we only count the number of exponentiations, since they are the most costly operations. For concrete efficiency, we instantiate the protocols at 128-bit security and assume that the number of users $\mu \approx 2^{30}$. This is about the number of monthly active users in a social media app[2]. We instantiate the fully tight protocols with a NIST P256 curve and "optimal" tight ones with a NIST P384 (since they require a 158-bit hard DLog assumption). Our benchmarks for an exponentiation in a P256 and P384 are 0.5 ms and 1 ms, using Apple M1 Max, 32GB of RAM and macOS Ventura 13.3.1 (a).

We observe that the DDH-based non-committing KEM in [21] is tightly OW-ChCCA secure (cf. [30, Footnote 1]). Our analysis shows that the wFS JKRS in [21] is tightly OW-VwFS, and after adding key confirmation to the wFS JKRS in [21] we get $\mathsf{JKRS}_{\mathsf{KC}}$ that is tightly FS. According to Table 1, our tight security proofs allow one to implement $\mathsf{JKRS}_{\mathsf{KC}}$ with about 30% shorter transcripts and 50% faster speed than the one with the "optimal", non-tight proofs in [18] at 128-bit security. Considering security against State Reveals, $\mathsf{JKRS}_{\mathsf{KC}}$ is the

[1] Their lattice-based protocol is almost tight (similar to [9]), since it needs to lose a factor of $O(\lambda)$ to the LWE assumption, where λ is the security parameter. We call it tight as well, but specify the concrete loss in our theorems and proofs.

[2] Cf. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/.

Table 1. Comparison of Diffie-Hellman-based AKE protocols with (tight or "optimal" tight) FS. Concrete efficiency is estimated for 128-bit security. "$\mathsf{JKRS}_{\mathsf{KC}}$ [18]" is transforming the implicitly authenticated JKRS [21] via key confirmation. We estimate its efficiency, according to the "optimal", non-tight security bound by Gellert et al. [18]. The last row is the same construction as the second last one, but with our tight security proof (cf. Theorem 4). In the upper arrows, schemes are using signatures, and we estimate the concrete bytes with the most efficient signature scheme in [13]. **Comm.** counts values exchanged during the protocol execution. $\mathbb{G}$ counts the number of group elements, H the number of hashes or MACs, 'Sign' the number of signatures, and 'other' the additional data in bits. **Bytes** counts total data in bytes by instantiating $\mathbb{G}$ with NIST P256 or P384 (for the non-tight $\mathsf{JKRS}_{\mathsf{KC}}$). **Exp.** counts the total numbers of exponentiation (which is the most costly computation in an AKE protocol) from both parties in agreeing a session key, and **Time** is the estimated time of computing those exponentiation in milliseconds.

Protocol	Comm. ($\mathbb{G}, H$, Sign, other)	Bytes	Exp.	Time (ms)	#Msg.	State Reveal	Security loss
TLS 1.3 [11,14]	(2, 2, 2, 512)	384	32	16	3	no	$O(1)$
GJ [19]	(2, 1, 2, 0)	288	32	16	3	no	$O(1)$
LLGW [26]	(3, 0, 2, 0)	288	35	17.5	2	no	$O(1)$
JKRS [21]	(5, 1, 1, 0)	288	29	14.5	2	yes	$O(1)$
PQR [27]	(2, 0, 2, 0)	256	32	16	2	no	$O(1)$
$\mathsf{CCGJJ}_{\mathsf{KC}}$ [10]	(2,2,0,0)	160	8	8	3	no	$O(\mu)$
$\mathsf{JKRS}_{\mathsf{KC}}$ [18]	(5,2,0,0)	304	15	15	3	yes	$O(\mu)$
$\mathsf{JKRS}_{\mathsf{KC}}$ (Ours)	(5,2,0,0)	224	15	7.5	3	yes	$O(1)$

most efficient DDH-based protocol, due to our tight security proofs. It is worth mentioning that the $\mathsf{CCGJJ}_{\mathsf{KC}}$ has shorter protocol transcripts, but it is insecure under State Reveals.

Interestingly, although the signature-based JKRS uses relatively inefficient primitives as signatures, its tight security proof allows an instantiation that is slightly more efficient than the non-tight, signature-less JKRS (namely, $\mathsf{JKRS}_{\mathsf{KC}}$ with proofs in [18]).

Relation to the Work of Gellert et al. [18]. We circumvent the impossibility result of Gellert et al. [18] by using a different wFS notion, OW-VwFS, and random oracles. As discussed earlier, the key checking oracle makes our notion stronger. The security definition in [18] does not have such an oracle and thus their impossibility result does not apply to our proof. At the same time, we opted for the weakest definition which allows a tight reduction (i.e., one-wayness and also no Reveal oracle), which makes our definition and that of [18] incomparable (neither implies the other). Moreover, their impossibility is in the standard model, while ours is in the random oracle model. For these reasons, our results do not contradict the impossibility result in [18], but rather provides an alternative way to prove security while enabling full tightness.

1.2 Our Contribution II: Forward Secrecy via Key Confirmation in the QROM

Our second contribution is proposing the first security proof for FS via key confirmation in the quantum random oracle model (QROM) [6], where a quantum adversary can have quantum access to the hash function. Our analysis considers the KEM-based AKE protocol (via key confirmation) and assumes a Multi-User, Multi-Challenge Chosen-Ciphertext Attacks (MUC-CCA) KEM and a Multi-Challenge CCA (MC-CCA) KEM. The main reason of doing so is that we do not know how to tightly prove OW-VwFS implies FS in the QROM, since it will trigger the Oneway-to-Hiding Lemma [32] and lead to a square-root-loss such as $\sqrt{\varepsilon}$, where ε is the advantage of breaking the underlying KEM. We still think that our tight lattice-based protocol in the classical ROM is interesting, since it is the first protocol with tight FS from post-quantum assumptions. Of course, one may alternatively rephrase our analysis in the classical ROM with the suitable KEMs, but it may lower the readability. More importantly, our OW-VwFS notion is more generic and gives more freedom to designers to construct their OW-VwFS protocols that will lead to FS in a tightness-preserving manner.

Our security bound in the QROM is unfortunately non-tight. More precisely, ignoring the statistically negligible terms, **our security bound for FS** in the QROM is

$$\varepsilon_{\mathrm{FS}}^{\mathsf{our}} \leq O(\mu) \cdot \varepsilon_{\text{MC-CCA}} + O(1) \cdot \varepsilon_{\text{MUC-CCA}}. \tag{1}$$

where μ is the number of users, $\varepsilon_{\text{MC-CCA}}$ is the advantage of MC-CCA, and $\varepsilon_{\text{MUC-CCA}}$ is that of MUC-CCA. It matches the best known bound for wFS in the QROM proposed by Pan, Wagner, and Zeng (PWZ) [31]. In this sense our FS bound preserves the tightness of the KEM-based AKE protocol with wFS. It is worth mentioning that, as shown in [31], we can tightly instantiate MC-CCA and MUC-CCA from the LWE assumption in the QROM.

We also improve PWZ's analysis in the sense that their analysis considers only one single Test query, but our security bound (as stated in Eq. (1)) is established in the context of multiple challenges, where an adversary is allowed to query Test multiple times. The multi-challenge setting is more realistic and well-established for public-key primitives [2,10,16,17], since in the real world, an adversary usually wants to attack multiple instances of a primitives. Although the security bound of PWZ can be extended to the multi-Test setting with a multiplicative factor t (which is the number of Test-queries), ours does not need to lose such a factor. In practice, t can be up to the total number of established sessions, which can be much larger than the number of users. We stress that the analysis of PWZ is only for wFS, and transforming it to FS, it may lose another multiplicative factor μ by applying the analysis of GGJJ [18]. Hence, combining the analysis of PWZ and GGJJ leads to a bound for FS in the QROM as

$$\varepsilon_{\mathrm{FS}}^{\mathsf{PWZ\ \&\ GGJJ}} \leq O(\mu^2 t) \cdot \varepsilon_{\text{MC-CCA}} + O(\mu t) \cdot \varepsilon_{\text{MUC-CCA}}. \tag{2}$$

Strictly speaking, the bound above is only an estimation and not theoretically sound, since the analysis of GGJJ is in the classical setting. Their bound may

change, if an adversary can query the hash function or key derivation function with a quantum state.

More Related Work in the QROM. Another work on the KEM-based AKE protocol in the QROM is due to Hövelmanns, Kiltz, Schäge, and Unruh [20], and it has a square-root-loss, namely, its security bound is

$$O(S^2 + S \cdot \mu) \cdot \left(\varepsilon_{\mathrm{CPA}} + \sqrt{Q \cdot \varepsilon_{\mathrm{CPA}}}\right), \tag{3}$$

where S, μ, and Q are the numbers of total sessions, users, and random oracle queries, respectively, and $\varepsilon_{\mathrm{CPA}}$ is the advantage of breaking the underlying CPA secure PKE. Similar to the work of PWZ, Eq. (3) is only for wFS and in the single-Test setting. Upgrading to FS in the multi-Test setting requires an additional multiplicative loss in μt. It is usually less desirable to have the square-root-loss as in Eq. (3), since it reduces the security guarantee of the underlying PKE in half.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. For a finite set $\mathcal{S}$, we denote the sampling of a uniform random element x by $x \xleftarrow{\$} \mathcal{S}$. 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 $y := \mathcal{A}(x)$ 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 stated differently. 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 [4] 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 random coins in game G. More generically, we write $\Pr[\mathsf{Event} : G]$ to denote the probability that Event happens in the game G. If the context is clear, we simply write it as $\Pr[\mathsf{Event}]$. Within a procedure, "**abort**" means that we terminate the run of an adversary $\mathcal{A}$.

3 Three-Message Authenticated Key Exchange

We recall the AKE security model from [21] and adapt it to three-message protocols. A three-message key exchange protocol $\mathsf{AKE} := (\mathsf{Setup}, \mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathsf{I}}, \mathsf{Init}_{\mathsf{R}}, \mathsf{Der}_{\mathsf{I}}, \mathsf{Der}_{\mathsf{R}})$ consists of five algorithms which are executed interactively by two parties as shown in Fig. 1.

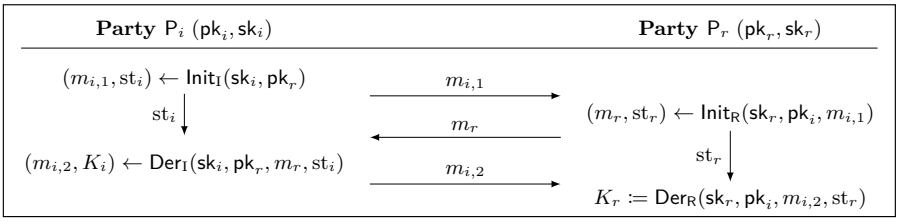

Fig. 1. Running a three-message AKE protocol between two parties.

Setup is the setup algorithm for the system parameters. We denote the party which initiates the session by P_i and the party which responds to the session by P_r. The key generation algorithm $\mathsf{Gen}_\mathsf{AKE}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ for one party. The initiator's initialization algorithm Init_I inputs the initiator's long-term secret key sk_i and the responder's long-term public key pk_r, and outputs a message $m_{i,1}$ and the initiator's state st_i. The responder's initialization algorithm Init_R inputs the responder's long-term secret key sk_r and the initiator's long-term public key pk_i, and outputs a message m_r and the responder's state st_r. The initiator's derivation algorithm Der_I takes as input sk_i, pk_r, a message m_r and the state st_i. It computes the final message $m_{i,2}$ and a session key K. The responder's derivation algorithm Der_R takes as input the sk_r, pk_i, a message $m_{i,2}$ and the state st_r. It computes the session key K. Here K can be $\perp$ meaning that the session is rejected during the execution. Correctness of an AKE protocol states that an honest execution between two parties should yield the same session key.

Definition 1 (Correctness of three-message AKE). *Let* $\mathsf{AKE} := (\mathsf{Setup}, \mathsf{Gen}_\mathsf{AKE}, \mathsf{Init}_\mathsf{I}, \mathsf{Init}_\mathsf{R}, \mathsf{Der}_\mathsf{I}, \mathsf{Der}_\mathsf{R})$ *be a three-message AKE protocol. We say* AKE *is* ρ*-correct if*

$$\Pr\left[K_i = K_r \neq \perp : \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_\mathsf{AKE}(\mathsf{par}), (\mathsf{pk}_r, \mathsf{sk}_r) \leftarrow \mathsf{Gen}_\mathsf{AKE}(\mathsf{par}), \\ (m_{i,1}, \mathrm{st}_i) \leftarrow \mathsf{Init}_\mathsf{I}(\mathsf{sk}_i, \mathsf{pk}_r), \\ (m_r, \mathrm{st}_r) \leftarrow \mathsf{Init}_\mathsf{R}(\mathsf{sk}_r, \mathsf{pk}_i, m_{i,1}), \\ (m_{i,2}, K_i) \leftarrow \mathsf{Der}_\mathsf{I}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st}_i), \\ K_r := \mathsf{Der}_\mathsf{R}(\mathsf{sk}_r, \mathsf{pk}_i, m_{i,2}, \mathrm{st}_r) \end{array}\right] \geq \rho \ ,$$

where the probability is taken over the randomness of $\mathsf{Setup}, \mathsf{Gen}_\mathsf{AKE}, \mathsf{Init}_\mathsf{I}, \mathsf{Init}_\mathsf{R}$, *and* Der_I.

We give a security game written in pseudocode focusing on (full) forward secrecy, rather than implicit or explicit authentication. We refer readers to [12] for more details on different types of authentication for key exchange protocols, and their connections to forward secrecy in [18].

Execution Environment. We consider μ 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

GAME IND-FS, [IND-FS-St]

00 **for** $n \in [\mu]$
01 $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{AKE}}$
02 $b \xleftarrow{\$} \{0,1\}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
04 **for** $\mathrm{sID}^* \in \mathcal{S}_{\mathtt{test}}$
05 **if** Fresh(sID^*) = **false** //session not fresh
06 **or** Valid(sID^*) = **false** //no valid attack
07 **return** b
08 **return** $[\![b = b']\!]$

Session$_{\mathrm{R}}((i,r) \in [\mu]^2, m_{i,1})$
09 $\mathrm{cnt}_{\mathrm{S}}$ ++
10 $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
11 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
12 $\mathtt{Type}[\mathrm{sID}] :=$ "Re"
13 $(m_r, \mathrm{st}_r) \leftarrow \mathsf{Init}_{\mathrm{R}}(\mathsf{sk}_r, \mathsf{pk}_i, m_{i,1})$
14 $(\mathtt{Msg}_{\mathsf{I},1}[\mathrm{sID}], \mathtt{Msg}_{\mathrm{R}}[\mathrm{sID}]) := (m_{i,1}, m_r)$
15 $\mathrm{st}[\mathrm{sID}] := \mathrm{st}_r$
16 **return** (sID, m_r)

Der$_{\mathrm{R}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_{i,2})$
17 **if** $\mathtt{SK}[\mathrm{sID}] \neq \bot$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "Re"
18 **return** $\bot$ //no re-use
19 $(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
20 $\mathrm{st}_r := \mathtt{ST}[\mathrm{sID}]$
21 $\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[i]$
22 $K := \mathsf{Der}_{\mathrm{R}}(\mathsf{sk}_r, \mathsf{pk}_i, m_{i,2}, \mathrm{st}_r)$
23 **if** $K \neq \bot$
24 $\mathtt{SK}[\mathrm{sID}] := K$
25 **else**
26 $\mathtt{SK}[\mathrm{sID}] :=$ "reject"
27 $\mathtt{Msg}_{\mathsf{I},2}[\mathrm{sID}] := m_{i,2}$
28 **return** ε

[Rev-State(sID)
29 $\mathtt{revST}[\mathrm{sID}] :=$ **true**
30 **return** $\mathtt{ST}[\mathrm{sID}]$]

Session$_{\mathrm{I}}((i,r) \in [\mu]^2)$
31 $\mathrm{cnt}_{\mathrm{S}}$ ++
32 $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
33 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
34 $\mathtt{Type}[\mathrm{sID}] :=$ "In"
35 $(m_{i,1}, \mathrm{st}_i) \leftarrow \mathsf{Init}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r)$
36 $(\mathtt{Msg}_{\mathsf{I},1}[\mathrm{sID}], \mathtt{ST}[\mathrm{sID}]) := (m_{i,1}, \mathrm{st}_i)$
37 **return** $(\mathrm{sID}, m_{i,1})$

Der$_{\mathrm{I}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_r)$
38 **if** $\mathtt{SK}[\mathrm{sID}] \neq \bot$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "In"
39 **return** $\bot$ //no re-use
40 $(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
41 $\mathrm{st}_i := \mathtt{ST}[\mathrm{sID}]$
42 $\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[r]$
43 $(m_{i,2}, K) \leftarrow \mathsf{Der}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st}_i)$
44 $(\mathtt{Msg}_{\mathrm{R}}[\mathrm{sID}], \mathtt{Msg}_{\mathsf{I},2}[\mathrm{sID}]) := (m_r, m_{i,2})$
45 **if** $K \neq \bot$
46 $\mathtt{SK}[\mathrm{sID}] := K$
47 **else**
48 $\mathtt{SK}[\mathrm{sID}] :=$ "reject"
49 **return** $m_{i,2}$

Reveal(sID)
50 $\mathtt{revSK}[\mathrm{sID}] :=$ **true**
51 **return** $\mathtt{SK}[\mathrm{sID}]$

Corr$(n \in [\mu])$
52 $\mathtt{cor}[n] :=$ **true**
53 **return** sk_n

Test(sID)
54 **if** $\mathrm{sID} \in \mathcal{S}_{\mathtt{test}}$ **return** $\bot$ //already tested
55 **if** $\mathtt{SK}[\mathrm{sID}] \in \{\bot,$ "reject"$\}$ **return** $\bot$
56 $\mathcal{S}_{\mathtt{test}} := \mathcal{S}_{\mathtt{test}} \cup \{\mathrm{sID}\}$
57 $K_0^* := \mathtt{SK}[\mathrm{sID}]$
58 $K_1^* \xleftarrow{\$} \mathcal{K}$
59 **return** K_b^*

Fig. 2. Games IND-FS and IND-FS-St for AKE. Rev-State is only available in IND-FS-St. In IND-FS, $\mathcal{A}$ has access to oracles $\mathrm{O} := \{$Session$_{\mathrm{I}}$, Session$_{\mathrm{R}}$, Der$_{\mathrm{I}}$, Der$_{\mathrm{R}}$, Reveal, Corr, Test$\}$. In IND-FS-St, $\mathcal{A}$ has access to the oracles in IND-FS and the Rev-State oracle. Helper procedures Fresh and Valid are defined in Fig. 3. If there exists any test session which is neither fresh nor valid, the game will return b.

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:

- $\mathtt{Init}[\mathrm{sID}] \in [\mu]$ denotes the initiator of the session.
- $\mathtt{Resp}[\mathrm{sID}] \in [\mu]$ denotes the responder of the session.
- $\mathtt{Type}[\mathrm{sID}] \in \{$"In", "Re"$\}$ denotes the session's view, i. e. whether the initiator or the responder computes the session key.

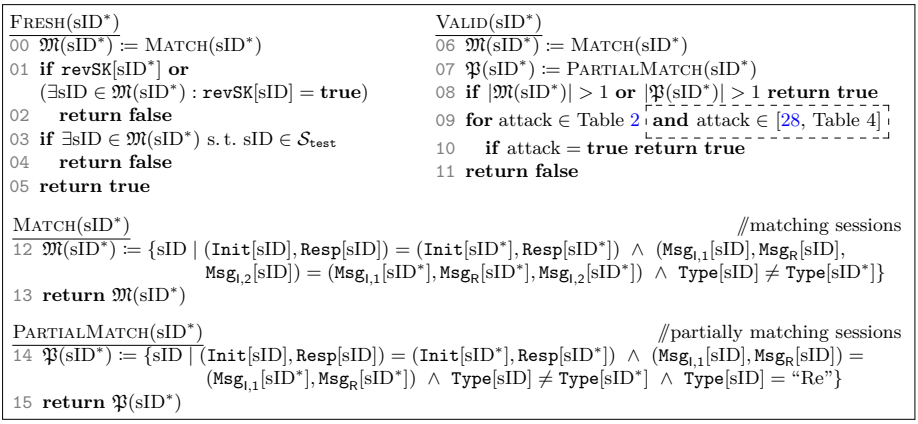

Fig. 3. Helper procedures FRESH and VALID for games IND-FS and IND-FS-St defined in Fig. 2. Procedure FRESH checks if the adversary performed some trivial attack. In procedure VALID, each attack is evaluated by the set of variables in attack tables and checks if an allowed attack was performed. The attack table for IND-FS is shown in Table 2 and the table for game IND-FS-St is given in [28, Table 4], where the latter includes session-state reveal attacks. 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.

- $\mathtt{Msg}_{\mathsf{I},1}[\mathrm{sID}]$ denotes the first message that was computed by the initiator.
- $\mathtt{Msg}_{\mathsf{R}}[\mathrm{sID}]$ denotes the message that was computed by the responder.
- $\mathtt{Msg}_{\mathsf{I},2}[\mathrm{sID}]$ denotes the final message that was computed by the initiator.
- $\mathtt{ST}[\mathrm{sID}]$ denotes the (secret) state information, i. e. ephemeral secret keys.
- $\mathtt{SK}[\mathrm{sID}]$ denotes the session key. If the session terminates without a valid session key, we set this variable to the special string "reject".

To establish a session between two parties, the adversary is given access to oracles $\textsc{Session}_{\mathsf{I}}$ and $\textsc{Session}_{\mathsf{R}}$, where the first one starts a session of type "In" and the second one of type "Re". In order to complete a session, oracles $\textsc{Der}_{\mathsf{I}}$ and $\textsc{Der}_{\mathsf{R}}$ have to be queried. The adversary has also access to oracles CORR, REVEAL and REV-STATE to obtain secret information. (The latter is only available if state reveal attacks are considered.) We use the following boolean values to keep track of which queries the adversary made:

- $\mathtt{cor}[n]$ denotes whether the long-term secret key of party P_n was given to the adversary.
- $\mathtt{peerPreCor}[\mathrm{sID}]$ denotes whether the peer of the session was corrupted and its long-term key was given to the adversary *before* the owner's session key was computed, which is important for forward security.
- $\mathtt{revST}[\mathrm{sID}]$ denotes whether the session state was given to the adversary.
- $\mathtt{revSK}[\mathrm{sID}]$ denotes whether the session key was given to the adversary.

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, the messages sent and received are the same they are of different types (cf. line 12 in Fig. 3).
- **Partially Matching Session**: A session sID has a *partially matching* session sID′ if the same parties are involved, the messages sent and received are the same without considering the last message and they are of different types, where sID′ is of type "Re" (cf. line 14 in Fig. 3).

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 queries, and store test sessions in a set $\mathcal{S}_{\mathtt{test}}$. For each test session, we require that the adversary does not issue queries such that the session key can be trivially computed. In Fig. 3 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. For game IND-FS-St, we capture this with attacks listed in our full paper [28, Table 4]. For game IND-FS, we use Table 2 to capture valid attacks.

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. We capture this as part of the validity property (cf. line 08). For an honest run of the protocol, the underlying min-entropy ensures that this attack will only happen with negligible probability.

We define validity of different attack strategies in Table 2, using variables to indicate which queries the adversary may (not) make. The purpose is to make our proofs precise by listing all the possible and non-trivial attacks. Attacks covered in the IND-FS model capture *forward secrecy* (FS) and *key compromise impersonation* (KCI) attacks. We provide a more detailed description of Table 2 and the full table for IND-FS-St in our full paper [28, Appendix B]. 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 2 (Key Indistinguishability of AKE). *We define games* IND-FS *and* IND-FS-St *as in Figs. 2 and 3. We say* AKE *is* $(t, \varepsilon, \mu, S, T, Q_{\mathrm{Cor}})$-IND-FS-*secure resp.* $(t', \varepsilon', \mu, S, T, Q_{\mathrm{Cor}}, Q_{\mathrm{St}})$-IND-FS-St-*secure if for all adversaries* $\mathcal{A}$ *attacking the protocol in time* t *resp.* t' *with* μ *users,* S *sessions,* T *test queries,* Q_{Cor} *corruptions, and* Q_{St} *state reveals, we have*

$$\left|\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] - \frac{1}{2}\right| \leq \varepsilon \quad \textit{resp.} \quad \left|\Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] - \frac{1}{2}\right| \leq \varepsilon' \ .$$

Table 2. Table of attacks for adversaries against three-message protocols with FS. 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". This table is obtained from [28, Table 3] by excluding all trivial attacks.

	$\mathcal{A}$ gets (Initiator, Responder)	$\mathtt{cor}[i^*]$	$\mathtt{cor}[r^*]$	$\mathtt{peerPreCor}[\mathrm{sID}^*]$	$\mathrm{Type}[\mathrm{sID}^*]$	$\lvert\mathfrak{M}(\mathrm{sID}^*)\rvert$	$\lvert\mathfrak{K}(\mathrm{sID}^*)\rvert$
1	**(long-term, long-term)**	–	–	–	"In"	–	1
2	**(long-term, long-term)**	–	–	–	"Re"	1	–
5	**(long-term, long-term)**	–	–	**F**	"In"	–	0
6	**(long-term, long-term)**	–	–	**F**	"Re"	0	–

Note that if there exists a session which is neither fresh nor valid, the game outputs the bit b, which implies that $\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] = \frac{1}{2}$ or $\Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] = \frac{1}{2}$, 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 Authenticated Key Exchange

To build a tightly secure three-message AKE protocol with key confirmation from a two-message AKE protocol, we define two security notions of the two-message protocol: The first one is One-Way against key Verification attacks and weak Forward Secrecy, or OW-VwFS for short, and the second one is OW-VwFS with state-reveal attacks, or OW-VwFS-St for short.

We define the syntax of a two-message key exchange protocol in a similar fashion as the three-message AKE. Let $\mathsf{AKE}' := (\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_{\mathrm{I}}, \mathsf{Init}'_{\mathsf{R}}, \mathsf{Der}'_{\mathrm{I}})$, where Setup', Gen' and $\mathsf{Init}'_{\mathrm{I}}$ are defined exactly as in the three-message protocol. instead of a state, the responder's algorithm $\mathsf{Init}'_{\mathsf{R}}$ computes a session key K. The initiator's algorithm $\mathsf{Der}'_{\mathrm{I}}$ does not output a second message, but only the session key. Correctness is defined similarly to the three-message case.

Definition 3 (Correctness of two-message AKE). *Let* $\mathsf{AKE}' := (\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_{\mathrm{I}}, \mathsf{Init}'_{\mathsf{R}}, \mathsf{Der}'_{\mathrm{I}})$ *be an AKE protocol. We say* AKE' *is* ρ*-correct if*

$$\Pr\left[K_i = K_r \neq \perp : \begin{array}{l} \mathsf{par}' \leftarrow \mathsf{Setup}'(1^\lambda), \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}'(\mathsf{par}), (\mathsf{pk}_r, \mathsf{sk}_r) \leftarrow \mathsf{Gen}'(\mathsf{par}), \\ (m_i, \mathrm{st}_i) \leftarrow \mathsf{Init}'_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r), \\ (m_r, K_r) \leftarrow \mathsf{Init}'_{\mathsf{R}}(\mathsf{sk}_r, \mathsf{pk}_i, m_i), \\ K_i := \mathsf{Der}'_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st}_i) \end{array}\right] \geq \rho,$$

where the probability is taken over the randomness of Setup', Gen', $\mathsf{Init}'_{\mathrm{I}}$, *and* $\mathsf{Init}'_{\mathsf{R}}$.

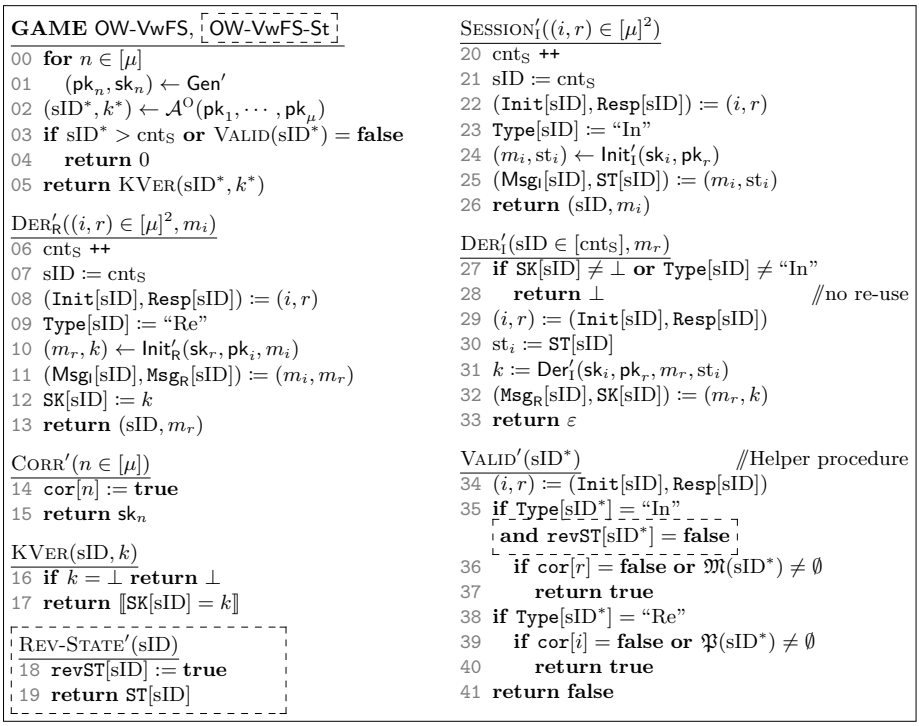

Fig. 4. Games OW-VwFS (without dashed boxes) and OW-VwFS-St (including dashed boxes) for AKE′. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}'_I, \textsc{Der}'_R, \textsc{Der}'_I, \textsc{Corr}', \textsc{KVer}\}$. In OW-VwFS-St, $\mathcal{A}$ also has access to $\textsc{Rev-State}'$. In two-message AKE, responder sessions do not have state. So, $\textsc{Rev-State}'(\mathrm{sID})$ will return $\perp$ if sID is a responder session. Further, partially matching session is defined as $\mathfrak{P}(\mathrm{sID}^*) := \{\mathrm{sID} \mid \mathtt{Type}[\mathrm{sID}] =$ "In" $\wedge\ (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) = (\mathtt{Init}[\mathrm{sID}^*], \mathtt{Resp}[\mathrm{sID}^*]) \wedge \mathsf{Msg}_I[\mathrm{sID}] = \mathsf{Msg}_I[\mathrm{sID}^*]\}$.

OW-VwFS is similar to the standard weak forward secrecy, but an adversary is additionally allowed to check if a key corresponds to some generated transcripts. The security notion OW-VwFS-St, based on OW-VwFS, allows the adversary to reveal session states. Moreover, these two security notions do not have Reveal and Test oracles. Our notion is motivated by the one-wayness against honest and key verification attacks in [27], but it is stronger in the sense that it allows active attacks. These are formally defined by Definition 4 with security games OW-VwFS and OW-VwFS-St as in Fig. 4.

Definition 4 (OW-VwFS and OW-VwFS-St security). *A two-message authenticated key exchange protocol* AKE′ *is* $(t, \varepsilon, \mu, S, Q_{\mathrm{COR}}, Q_{Ver})$-OW-VwFS *secure resp.* $(t', \varepsilon', \mu, S, Q_{\mathrm{COR}}, Q_{Ver}, Q_{\mathrm{ST}})$-OW-VwFS-St *secure, where* μ *is the number of users,* S *is the number of sessions,* Q_{Ver} *is the number of calls to* KVer *and* Q_{ST} *is the number of calls to* $\textsc{Rev-State}'$*, if for all adversaries* $\mathcal{A}$ *attacking the protocol in time at most* t *resp.* t'*, we have*

$$\Pr[\text{OW-VwFS}^{\mathcal{A}}_{\text{AKE}'} \Rightarrow 1] \leq \varepsilon \quad \textit{resp.} \quad \Pr[\text{OW-VwFS-St}^{\mathcal{A}}_{\text{AKE}'} \Rightarrow 1] \leq \varepsilon' \ .$$

Valid attacks are defined via $\textsc{Valid}'$. For the session sID^* for which the adversary aims to compute the session key, we basically allow two types of attacks: If there is a (partially) matching session, then both parties may be corrupted. Otherwise, the adversary must not corrupt the peer of the session. Additionally for the model with state reveal attacks, the state for sID^* must not be revealed in any case.

Min-Entropy. We require that public keys have γ *bits of min-entropy*, i. e., for all $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}'$, $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}'$, we have $\Pr[\mathsf{pk}_0 = \mathsf{pk}_1] \leq 2^{-\gamma}$. Similarly, we require that messages have α *bits of min-entropy*, i. e., for all messages m' we have $\Pr[m = m'] \leq 2^{-\alpha}$, where m is output by either $\mathsf{Init}'_\mathsf{I}$ or $\mathsf{Init}'_\mathsf{R}$.

5 AKE with Key Confirmation

We now build a three-message AKE protocol AKE_KC with key confirmation from a two-message AKE protocol AKE' and three hash functions $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R}, \mathsf{H}$. An overview is given in Fig. 5. Hash functions G_I, G_R and H are defined as follows: $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$, where λ is the length of key confirmation tags and $\mathcal{K}$ is the key space of AKE_KC.[3]

Let $\mathsf{AKE}' = (\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_\mathsf{I}, \mathsf{Init}'_\mathsf{R}, \mathsf{Der}'_\mathsf{I})$. We define AKE_KC as follows: Setup, $\mathsf{Gen}_\mathsf{AKE}$, Init_I will be the same as Setup', Gen' and $\mathsf{Init}'_\mathsf{I}$, respectively. Init_R first runs $\mathsf{Init}'_\mathsf{R}$ to obtain the responder's message m_r and the key k of AKE', where the latter is used to derive the final session key and key confirmation messages. In particular, the responder first computes the key confirmation tag $\pi_r := \mathsf{G}_\mathsf{R}(k, \mathrm{ctxt})$, where ctxt is defined as the two parties' public keys and the initial messages (cf. Fig. 5). It then also computes the expected key confirmation tag π'_i and session key K' using G_I and H on the same input. It sends (m_r, π_r) to the initiator and keeps (π'_i, K') as state. The initiator runs Der_I which is defined as follows: First, it runs Der'_I to get k and then performs the same computations as the responder to compute key confirmation tags π_i, π'_r and the final session key K. It accepts K if $\pi'_r = \pi_r$ and sends π_i as the final message. The responder's derivation algorithm Der_R checks whether the key confirmation tag is valid, i. e., $\pi_i = \pi'_i$, and if this is the case it sets the session key to K'.

Whenever an equality check fails or the underlying algorithms of AKE' return $\perp$, the parties terminate the session, i. e., they *reject*, and return $\perp$.

Correctness. The correctness of AKE_KC follows directly from the correctness of AKE'. In particular, if AKE' is $(1-\delta)$-correct, then so is AKE_KC.

Security. We prove IND-FS security of AKE_KC based on OW-VwFS security of AKE' and modeling G_I, G_R and H as random oracles.

Theorem 1. *Let* AKE' *be* $(1-\delta)$*-correct and have public keys with* γ *bits of entropy and messages with* α *bits of entropy. Let* AKE_KC *be as defined in Fig. 5, where* $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$ *and* $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$

[3] We define three different hash functions here which allows us to model them as independent random oracles. When instantiating the hash functions with the same function, one would need to use appropriate domain separation.

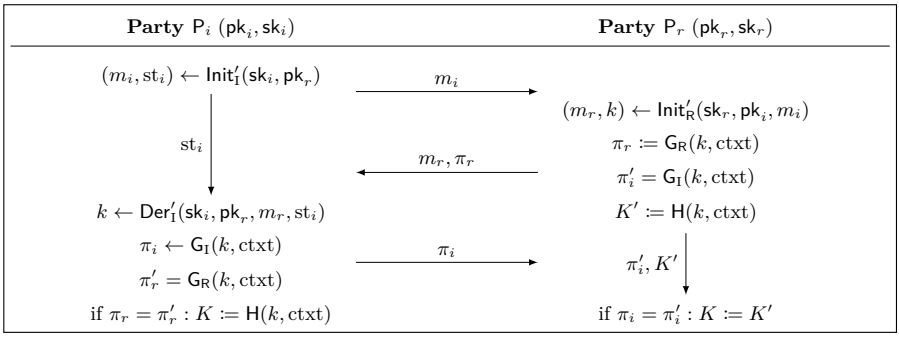

Fig. 5. AKE protocol AKE_KC from AKE' and key confirmation. The context is defined as $\mathrm{ctxt} \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$. G_I, G_R and H are independent random oracles.

are modelled as random oracles. For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}})$-IND-FS-security of AKE_KC, there exists an adversary $\mathcal{B}$ that breaks the $(t', \varepsilon', \mu, S, Q_{\mathrm{COR}}, Q_{Ver})$-OW-VwFS security of AKE' with $t' \approx t$ and

$$\varepsilon \leq \varepsilon' + 2S \cdot \delta + (S + S^2) \cdot 2^{-\lambda} + \mu^2 \cdot 2^{-\gamma} + S(S + Q_{\mathsf{G}_\mathrm{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_\mathsf{H}) \cdot 2^{-\alpha} \ ,$$

where $Q_{\mathsf{G}_\mathrm{I}}, Q_{\mathsf{G}_\mathsf{R}}$ and Q_H are the number of queries to random oracles G_I, G_R and H and $Q_{Ver} \leq S + Q_{\mathsf{G}_\mathrm{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_\mathsf{H}$.

The idea of the proof is that we can simulate the key confirmation tags and session keys without knowing the key k of the underlying two-message protocol as long as it has not been queried to (one of) the random oracles. For this we have to keep track of whether the adversary trivially knows k because the session is not fresh anymore. We can handle this case and still simulate correctly using KVER oracle. The only way to win the game is to compute k for a fresh and valid session, thus breaking one-wayness of the underlying protocol. We now prove the theorem formally.

Proof. Let $\mathcal{A}$ be an adversary against IND-FS security of AKE_KC. We consider the sequence of games G_0-G_3 in Figs. 6 and 7.

GAME G_0. The first game G_0 is the original IND-FS security game, however we exclude that public keys or messages collide (which means that if such events happen, then the game will abort and return a random bit). This also includes the key confirmation tags. Thus we get

$$\Pr[\mathsf{G}_0^\mathcal{A} \Rightarrow 1] \leq \Pr[\mathsf{IND\text{-}FS}^\mathcal{A}_{\mathsf{AKE}_\mathsf{KC}} \Rightarrow 1] + \mu^2 \cdot 2^{-\gamma} + S^2 \cdot 2^{-\alpha} + S^2 \cdot 2^{-\lambda}.$$

Note that this means there can be at most one (partially) matching session for each session.

GAME G_1. In game G_1, we want to ensure that π_r, π_i and K have not been queried to the respective random oracle before they are determined. Note that

GAMES G_0-G_3

00 **for** $n \in [\mu]$
01 $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}'$
02 $b \xleftarrow{\$} \{0,1\}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
04 **for** $\mathrm{sID}^* \in \mathcal{S}_{\mathsf{test}}$
05 **if** $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$ **or** $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
06 **return** b
07 **return** $[\![b = b']\!]$

$\textsc{Session}_{\mathsf{R}}((i,r) \in [\mu]^2, m_i)$

08 cnt_S ++
09 $\mathrm{sID} \coloneqq \mathrm{cnt}_S$
10 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) \coloneqq (i,r)$
11 $\mathtt{Type}[\mathrm{sID}] \coloneqq$ "Re"
12 $(m_r, k) \leftarrow \mathsf{Init}'_{\mathsf{R}}(\mathsf{sk}_r, \mathsf{pk}_i, m_i)$
13 **if** $k = \bot$
14 $\mathtt{SK}[\mathrm{sID}] \coloneqq$ "reject"
15 **return** $\bot$
16 $\pi_r \coloneqq \mathsf{G}_{\mathsf{R}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_0
17 $\pi_i \coloneqq \mathsf{G}_{\mathsf{I}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_0
18 $K \coloneqq \mathsf{H}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_0
19 **if** $\exists k'$ s. t. $\mathsf{G}_{\mathsf{R}}[k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \neq \bot$ // G_1-G_3
or $\mathsf{G}_{\mathsf{I}}[k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \neq \bot$ // G_1-G_3
or $\mathsf{H}[k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \neq \bot$ // G_1-G_3
20 $\mathsf{BadEntropy} \coloneqq \mathbf{true}$; **abort** // G_1-G_3
21 $\pi_r \xleftarrow{\$} \{0,1\}^\lambda$, $\pi_i \xleftarrow{\$} \{0,1\}^\lambda$, $K \xleftarrow{\$} \mathcal{K}$ // G_1-G_3
22 $\mathsf{G}_{\mathsf{R}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_r$ // G_1
23 $\mathsf{G}_{\mathsf{I}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_i$ // G_1
24 $\mathsf{H}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq K$ // G_1
25 $\mathtt{ctxt}[\mathrm{sID}] \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_2-G_3
26 $\mathtt{k}[\mathrm{sID}] \coloneqq k$ // G_2-G_3
27 **if** $\exists \mathrm{sID}'$ s. t. $\mathtt{ctxt}[\mathrm{sID}'] = (\mathsf{pk}_i, \mathsf{pk}_r, m_i, \bot)$ **or** $\mathtt{cor}[i] = \mathbf{false}$ // G_2-G_3
28 $\mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_r$ // G_2-G_3
29 $\mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_i$ // G_2-G_3
30 $\mathsf{H}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq K$ // G_2-G_3
31 **else** // G_2-G_3
32 $\mathsf{G}_{\mathsf{R}}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_r$ // G_2-G_3
33 $\mathsf{G}_{\mathsf{I}}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_i$ // G_2-G_3
34 $\mathsf{H}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq K$ // G_2-G_3
35 $(\mathsf{Msg}_{\mathsf{I},1}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}]) \coloneqq (m_i, (m_r, \pi_r))$
36 $\mathtt{ST}[\mathrm{sID}] \coloneqq (\pi_i, K)$
37 **return** $(\mathrm{sID}, (m_r, \pi_r))$

$\textsc{Session}_{\mathsf{I}}((i,r) \in [\mu]^2)$

38 cnt_S ++
39 $\mathrm{sID} \coloneqq \mathrm{cnt}_S$
40 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) \coloneqq (i,r)$
41 $\mathtt{Type}[\mathrm{sID}] \coloneqq$ "In"
42 $(m_i, \mathrm{st}_i) \leftarrow \mathsf{Init}'_{\mathsf{I}}(\mathsf{sk}_i, \mathsf{pk}_r)$
43 $(\mathsf{Msg}_{\mathsf{I},1}[\mathrm{sID}], \mathtt{ST}[\mathrm{sID}]) \coloneqq (m_i, \mathrm{st}_i)$
44 $\mathtt{ctxt}[\mathrm{sID}] \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, m_i, \bot)$ // G_2-G_3
45 **return** (sID, m_i)

$\textsc{Der}_{\mathsf{I}}(\mathrm{sID} \in [\mathrm{cnt}_S], (m_r, \pi_r))$

46 **if** $\mathtt{SK}[\mathrm{sID}] \neq \bot$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "In" **return** $\bot$
47 $(i,r) \coloneqq (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
48 $\mathrm{st}_i \coloneqq \mathtt{ST}[\mathrm{sID}]$
49 $\mathtt{peerPreCor}[\mathrm{sID}] \coloneqq \mathtt{cor}[r]$
50 $k \coloneqq \mathsf{Der}'_{\mathsf{I}}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st}_i)$
51 **if** $k = \bot$
52 $\mathtt{SK}[\mathrm{sID}] \coloneqq$ "reject"
53 **return** $\bot$
54 **if** $\pi_r \neq \mathsf{G}_{\mathsf{R}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_0-G_1
55 $\mathtt{SK}[\mathrm{sID}] \coloneqq$ "reject" // G_0-G_1
56 **return** $\bot$ // G_0-G_1
57 $\pi_i \coloneqq \mathsf{G}_{\mathsf{I}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_0-G_1
58 $K \coloneqq \mathsf{H}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_0-G_1
59 $\mathtt{k}[\mathrm{sID}] \coloneqq k$ // G_2-G_3
60 Replace $\bot$ in $\mathtt{ctxt}[\mathrm{sID}]$ with m_r // G_2-G_3
61 **if** $\exists \mathrm{sID}'$ s. t. $\mathtt{ctxt}[\mathrm{sID}'] = \mathtt{ctxt}[\mathrm{sID}]$ // G_2-G_3
62 **if** $\pi_r \neq \mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$ // G_2-G_3
63 $\mathtt{SK}[\mathrm{sID}] \coloneqq$ "reject" // G_2-G_3
64 **return** $\bot$ // G_2-G_3
65 $\pi_i \coloneqq \mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$ // G_2-G_3
66 $K \coloneqq \mathsf{H}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$ // G_2-G_3
67 **else** // G_2-G_3
68 **if** $\mathsf{G}_{\mathsf{R}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] = \pi_r$ // G_2-G_3
69 **if** $\mathtt{cor}[r] = \mathbf{false}$ // G_3
70 $\mathsf{QueryRO} \coloneqq \mathbf{true}$; **abort** // G_3
71 $\pi_i \coloneqq \mathsf{G}_{\mathsf{I}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_2-G_3
72 $K \coloneqq \mathsf{H}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ // G_2-G_3
73 **else** // G_2-G_3
74 $\mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \xleftarrow{\$} \{0,1\}^\lambda$ // G_2-G_3
75 **if** $\pi_r = \mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$ // G_2-G_3
76 $\mathsf{RandKC} \coloneqq \mathbf{true}$; **abort** // G_2-G_3
77 $\mathtt{SK}[\mathrm{sID}] \coloneqq$ "reject" // G_2-G_3
78 **return** $\bot$ // G_2-G_3
79 $(\mathsf{Msg}_{\mathsf{R}}[\mathrm{sID}], \mathsf{Msg}_{\mathsf{I},2}[\mathrm{sID}]) \coloneqq (m_r, \pi_i)$
80 $\mathtt{SK}[\mathrm{sID}] \coloneqq K$
81 **return** π_i

Fig. 6. Games G_0-G_3 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}_{\mathsf{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathsf{I}}, \textsc{Der}_{\mathsf{R}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{G}_{\mathsf{I}}, \mathsf{G}_{\mathsf{R}}, \mathsf{H}\}$. Helper procedures $\textsc{Fresh}$ and $\textsc{Valid}$ are defined in Fig. 3.

all three values will be determined in $\textsc{Session}_{\mathsf{R}}$ when m_r and k are computed. Thus, whenever $\textsc{Session}_{\mathsf{R}}$ is queried, we check whether there already exists a query $(k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ to G_{R}, G_{I} or H for some k' (line 19). If this is the case, we raise flag $\mathsf{BadEntropy}$ and abort. If $\mathsf{BadEntropy}$ is not raised, we draw fresh values for π_r, π_i and K and explicitly assign them to the corresponding

```
DER_R(sID ∈ [cnt_S], π_i)
00 if SK[sID] ≠ ⊥ or Type[sID] ≠ "Re"
01   return ⊥
02 (i, r) := (Init[sID], Resp[sID])
03 (π'_i, K') := ST[sID]
04 peerPreCor[sID] := cor[i]
05 if π_i ≠ π'_i                                   // G0-G1
06   SK[sID] := "reject"                           // G0-G1
07   return ⊥                                      // G0-G1
08 K := K'                                         // G0-G1
09 if ∃sID' s.t. ctxt[sID'] = ctxt[sID]            // G2-G3
10   if π_i ≠ G_I[⋄, pk_i, pk_r, m_i, m_r]         // G2-G3
11     SK[sID] := "reject"                         // G2-G3
12     return ⊥                                    // G2-G3
13   K := H[⋄, pk_i, pk_r, m_i, m_r]               // G2-G3
14 else                                            // G2-G3
15   if G_I[k, pk_i, pk_r, m_i, m_r] = π_i         // G2-G3
16     if cor[i] = false                           // G3
17       QueryRO := true; abort                    // G3
18     K := H(k, pk_i, pk_r, m_i, m_r)             // G2-G3
19   else                                          // G2-G3
20     G_I[⋄, pk_i, pk_r, m_i, m_r] ←$ {0,1}^λ     // G2-G3
21     if π_i = G_I[⋄, pk_i, pk_r, m_i, m_r]       // G2-G3
22       RandKC := true; abort                     // G2-G3
23     SK[sID] := "reject"                         // G2-G3
24     return ⊥                                    // G2-G3
25 (Msg_{I,2}[sID], SK[sID]) := (π_i, K)
26 return ε

G_R(k, pk_i, pk_r, m_i, m_r)
27 if G_R[⋄, pk_i, pk_r, m_i, m_r] = π ≠ ⊥          // G2-G3
28   S := {sID | ctxt[sID] = (pk_i, pk_r, m_i, m_r)} // G2-G3
29   for sID ∈ S // note |S| ≤ 2                    // G2-G3
30     if k[sID] = k                               // G2-G3
31       QueryRO := true; abort                    // G3
32       return π                                  // G2-G3
33 elseif G_R[⊕, pk_i, pk_r, m_i, m_r] = π ≠ ⊥      // G2-G3
34   Find sID s.t. ctxt[sID] = (pk_i, pk_r, m_i, m_r) // G2-G3
35   if k[sID] = k                                 // G2-G3
36     Replace ⊕ with k                            // G2-G3
37     return π                                    // G2-G3
38 if G_R[k, pk_i, pk_r, m_i, m_r] = π ≠ ⊥
39   return π
40 π ←$ {0,1}^λ
41 G_R[k, pk_i, pk_r, m_i, m_r] := π
42 return π

CORR(n ∈ [μ])
43 cor[n] := true
44 return sk_n

TEST(sID)
45 if sID ∈ S_test return ⊥
46 if SK[sID] ∈ {⊥, "reject"} return ⊥
47 S_test := S_test ∪ {sID}
48 K*_0 := SK[sID]
49 K*_1 ←$ K
50 return K*_b
```

Fig. 7. Oracles for games G_0-G_3 for the proof of Theorem 1. G_I and H are defined analogously to G_R.

entry of the respective random oracle (lines 21–24). We make this explicit here to prepare for the next step where we have to do a case distinction. Note that G_0 and G_1 are the same, except if $\mathsf{BadEntropy}$ is raised. Thus,

$$|\Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{BadEntropy}] \leq S(Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{G}_\mathrm{I}} + Q_\mathsf{H}) \cdot 2^{-\alpha},$$

where we bound the event by the entropy of AKE'. The message m_r is computed by the game directly before we check for this event. We then use the union bound over the maximum number of sessions S.

GAME G_2. In game G_2, we want to compute π_r, π_i and K without using k explicitly and prepare for the reduction to $\mathsf{OW\text{-}VwFS}$. For this, we have to make a distinction between fresh and non-fresh sessions. We add two additional variables $\mathtt{ctxt}[\mathrm{sID}]$ and $\mathtt{k}[\mathrm{sID}]$ for each session which store the context and the session key of the underlying AKE'. When $\textsc{Session}_\mathsf{R}$ is queried, we no longer assign the random oracle entries $[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$. Instead, we use a special placeholder symbol for the key k. In particular, if the session is still fresh and valid (i. e., there exists a session with a matching context up to this point, or the intended peer is not (yet) corrupted), we use the symbol $\diamond$ (lines 28–30). Otherwise, in case the session is not valid, we use the symbol $\oplus$ (lines 32–34). This distinction will be necessary to patch the random oracle correctly. Note that an adversary might be able to compute the correct key k for a non-valid session.

We describe how the random oracles are patched below. First, we explain how to change $\mathsf{Der}_{\mathrm{I}}$ and $\mathsf{Der}_{\mathsf{R}}$ accordingly. For each query to $\mathsf{Der}_{\mathrm{I}}$, we first update the context with the message m_r that was used to query the oracle. Then we check whether there exists a potential partnered session with the same context (line 61). In this case we know the corresponding values π_r, π_i and K which are stored with the symbol $\diamond$. We check whether the tag π_r is correct (if not, the session rejects) and assign the session key (lines 62–66). If there is no other session with the same context, we have to make another case distinction. In the first case, we use k explicitly to check whether there has been a query to the random oracle G_{R} such that the tag π_r matches (line 68). In this case, we proceed normally. Looking ahead, this will be a critical point in the next game modification. If there exists no such query to G_{R}, then the game makes the query and chooses a tag uniformly at random (line 74). If this tag is the same as the one provided by the adversary, we raise flag RandKC and abort (line 76). Otherwise, the session simply rejects. We modify $\textsc{Der}_{\mathsf{R}}$ in the exact same way, except that we are now looking at π_i (Fig. 7, lines 09–24).

Before bounding RandKC, we explain the simulation of random oracles as described in Fig. 7. We explain G_{R} in more detail. (G_{I} and H are modeled in exactly the same way). For each query $(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$, we first check for entries with the special symbol. More specifically, if there exists an entry with the given context and the symbol $\diamond$, we look for the sID with this context. Note that there can be at most two sessions (one of type "In" and one of type "Re" which will be matching sessions), which we capture by computing a set $\mathcal{S}$ containing the corresponding sID(s) (line 29). If the key k of the random oracle query corresponds to the one stored in $\mathtt{k}[\mathrm{sID}]$[4], then G_{R} simply outputs the stored value π (line 32). We do the same for special symbol $\oplus$ (line 34, note that here sID is always unique), except that we also update the entry accordingly, i. e., replace $\oplus$ with k if $k = \mathtt{k}[\mathrm{sID}]$ (line 36). This way, the simulation is consistent with the other oracles.

Overall, the two games only differ when flag RandKC is raised. Note that we can bound the probability that a tag is valid without the random oracle being queried by the length of the tag. Union bound over S session gives us

$$|\Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{RandKC}] \leq S \cdot 2^{-\lambda} + S \cdot \delta.$$

Game G_3. In the final game G_3, we raise flag $\mathsf{QueryRO}$ if the adversary ever queries the random oracle on a key k of a fresh session. Depending on the order of queries, this event can occur for different oracles: $\textsc{Der}_{\mathrm{I}}$ (Fig. 6, line 70), $\textsc{Der}_{\mathsf{R}}$ (Fig. 7, line 17) or one of the random oracles (Fig. 7, line 31). First, we look at sessions that do not have a matching session with the same context. For queries to oracle $\textsc{Der}_{\mathrm{I}}$ we check the validity of π_r in line 68. If the peer r is not corrupted,

[4] Since we cannot check correctness efficiently in the reduction which we will build in the next step, we explicitly perform the test here for all sessions in $\mathcal{S}$. However, if $\mathcal{S}$ indeed contains two sessions, then by correctness, this key (and thus the outcome) will be the same.

then the session is still fresh and valid. Thus, we raise $\mathsf{QueryRO}$ if there has been a query to G_R on the correct key and context such that the output is indeed π_r. We proceed similarly for responder sessions when $\textsc{Der}_\mathsf{R}$ is queried, checking for queries to G_I. This means that all sessions where the peer is uncorrupted and no session with a matching context exist will reject (or abort).

We now look at sessions that have a session with matching context and whose relevant random oracle entries are marked with $\diamond$. Whenever one of the random oracles is queried, we check whether the key matches the one stored in $\mathtt{k}[\mathrm{sID}]$ (as described earlier) and if this is the case, we also raise $\mathsf{QueryRO}$ and abort.

We claim that

$$|\Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{QueryRO}] \leq \varepsilon' + S \cdot \delta.$$

Before proving the claim, note that in G_3 we have $\Pr[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1] = 1/2$. For this, observe that all sessions must have a (partially) matching session and that random oracle H is never queried on k for any of those sessions. Thus, the session key is indistinguishable from a uniformly random key.

Bounding Event $\mathsf{QueryRO}$. We now describe an adversary $\mathcal{B}$ against $\mathsf{OW\text{-}VwFS}$ security of the underlying AKE' to bound event $\mathsf{QueryRO}$. A pseudocode description is given in Fig. 8. The idea is that whenever $\mathcal{A}$ queries one of the random oracles on the underlying key k of a fresh and valid session (either in order to forge a key confirmation tag or to distinguish the actual session key), we can use this to break $\mathsf{OW\text{-}VwFS}$ security of AKE', where the verification oracle $\textsc{KVer}$ is used to simulate the random oracles consistently.

We now describe $\mathcal{B}$ in more detail. It gets as input μ public keys and forwards them to $\mathcal{A}$. $\mathcal{B}$ simulates queries to oracle $\textsc{Session}_\mathrm{I}$ in a straightforward way by querying its own oracle $\textsc{Session}'_\mathrm{I}$ which returns (sID, m_i). After assigning the corresponding variables, $\mathcal{B}$ forwards the output to $\mathcal{A}$. Queries to $\textsc{Session}_\mathsf{R}$ are simulated as in game G_3. $\mathcal{B}$ first queries $\textsc{Der}'_\mathsf{R}$ to receive (sID, m_r). Instead of checking whether $k = \bot$, $\mathcal{B}$ checks whether $m_r = \bot$. If this is the case, it rejects and outputs $\bot$. Otherwise, it proceeds as described in G_3, preparing random oracle assignments by assigning fresh values to π_i, π_r and K and returning $(\mathrm{sID}, (m_r, \pi_r))$.

When $\mathcal{A}$ queries $\textsc{Der}_\mathrm{I}$, $\mathcal{B}$ queries $\textsc{Der}'_\mathrm{I}$. $\mathcal{B}$ will not be able to explicitly check whether the session key was computed successfully, however, we will argue that the simulation is consistent by correctness of AKE' and the validity of the key confirmation tag. Thus, $\mathcal{B}$ directly proceeds as described in G_3. Whenever there exists a session with the same context, then the key confirmation tag must be the same as the one computed by that session, up to correctness of AKE'. Thus the simulation is perfect except with probability $S \cdot \delta$. Whenever there exists no (partially) matching session, $\mathcal{B}$ needs to check whether G_R was already queried on the correct k. For this it checks all random oracle queries that have output π_r provided by $\mathcal{A}$. If such a query exists, it will be unique since we excluded collisions in the first game. $\mathcal{B}$ checks whether the respective key of the query is the correct key using its oracle $\textsc{KVer}$. If this is the case, we further distinguish two cases, based on whether the session still qualifies for a valid test session

$\mathcal{B}^{\text{SESSION}'_{\mathsf{I}},\text{DER}'_{\mathsf{R}},\text{DER}'_{\mathsf{I}},\text{CORR}',\text{KVER}}(\mathsf{pk}_1,\cdots,\mathsf{pk}_\mu)$
00 $b' \leftarrow \mathcal{A}^{\text{O}}(\mathsf{pk}_1,\cdots,\mathsf{pk}_\mu)$
01 **return** $\bot$

$\text{SESSION}_{\mathsf{R}}((i,r)\in[\mu]^2, m_i)$
02 $(\text{sID}, m_r) \leftarrow \text{DER}'_{\mathsf{R}}((i,r), m_i)$
03 $(\texttt{Init}[\text{sID}], \texttt{Resp}[\text{sID}]) \coloneqq (i,r)$
04 $\texttt{Type}[\text{sID}] \coloneqq$ "Re"
05 **if** $m_r = \bot$
06 $\quad \texttt{SK}[\text{sID}] \coloneqq$ "reject"
07 $\quad$ **return** $\bot$
08 **if** $\exists k'$ s. t. $\mathsf{G}_{\mathsf{R}}[k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \neq \bot$
or $\mathsf{G}_{\mathsf{I}}[k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \neq \bot$
or $\mathsf{H}[k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \neq \bot$
09 $\quad$ **abort**
10 $\pi_r \xleftarrow{\$} \{0,1\}^\lambda$, $\pi_i \xleftarrow{\$} \{0,1\}^\lambda$, $K \xleftarrow{\$} \mathcal{K}$
11 $\texttt{ctxt}[\text{sID}] \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$
12 **if** $\exists \text{sID}'$ s. t. $\texttt{ctxt}[\text{sID}'] = (\mathsf{pk}_i, \mathsf{pk}_r, m_i, \bot)$
or $\texttt{cor}[i] =$ **false**
13 $\quad \mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_r$
14 $\quad \mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_i$
15 $\quad \mathsf{H}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq K$
16 **else**
17 $\quad \mathsf{G}_{\mathsf{R}}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_r$
18 $\quad \mathsf{G}_{\mathsf{I}}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq \pi_i$
19 $\quad \mathsf{H}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \coloneqq K$
20 $(\mathsf{Msg}_{\mathsf{I}}[\text{sID}], \mathsf{Msg}_{\mathsf{R}}[\text{sID}]) \coloneqq (m_i, (m_r, \pi_r))$
21 **return** $(\text{sID}, (m_r, \pi_r))$

$\text{REVEAL}(\text{sID})$
22 $\texttt{revSK}[\text{sID}] \coloneqq$ **true**
23 **return** $\texttt{SK}[\text{sID}]$

$\text{CORR}(n \in [\mu])$
24 $\texttt{cor}[n] \coloneqq$ **true**
25 $\mathsf{sk}_n \leftarrow \text{CORR}'(n)$
26 **return** sk_n

$\text{SESSION}_{\mathsf{I}}((i,r)\in[\mu]^2)$
27 $(\text{sID}, m_i) \leftarrow \text{SESSION}'_{\mathsf{I}}(i,r)$
28 $(\texttt{Init}[\text{sID}], \texttt{Resp}[\text{sID}]) \coloneqq (i,r)$
29 $\texttt{Type}[\text{sID}] \coloneqq$ "In"
30 $\mathsf{Msg}_{\mathsf{I},1}[\text{sID}] \coloneqq m_i$
31 $\texttt{ctxt}[\text{sID}] \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, m_i, \bot)$
32 **return** (sID, m_i)

$\text{DER}_{\mathsf{I}}(\text{sID}, (m_r, \pi_r))$
33 **if** $\texttt{SK}[\text{sID}] \neq \bot$ **or** $\texttt{Type}[\text{sID}] \neq$ "In"
34 $\quad$ **return** $\bot$
35 $(i,r) \coloneqq (\texttt{Init}[\text{sID}], \texttt{Resp}[\text{sID}])$
36 $\texttt{peerPreCor}[\text{sID}] \coloneqq \texttt{cor}[r]$
37 $\text{DER}'_{\mathsf{I}}(\text{sID}, m_r)$
38 Replace $\bot$ in $\texttt{ctxt}[\text{sID}]$ with m_r
39 **if** $\exists \text{sID}'$ s. t. $\texttt{ctxt}[\text{sID}'] = \texttt{ctxt}[\text{sID}]$
40 $\quad$ **if** $\pi_r \neq \mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$
41 $\quad\quad \texttt{SK}[\text{sID}] \coloneqq$ "reject"
42 $\quad\quad$ **return** $\bot$
43 $\quad \pi_i \coloneqq \mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$
44 $\quad K \coloneqq \mathsf{H}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$
45 **else**
46 $\quad$ **if** $\exists k$ s. t. $\mathsf{G}_{\mathsf{R}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] = \pi_r$
and $\text{KVER}(k, \text{sID})$
47 $\quad\quad$ **if** $\texttt{cor}[r] =$ **false**
48 $\quad\quad\quad$ Stop with (sID, k)
49 $\quad\quad \pi_i \coloneqq \mathsf{G}_{\mathsf{I}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$
50 $\quad\quad K \coloneqq \mathsf{H}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$
51 $\quad$ **else**
52 $\quad\quad \mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \xleftarrow{\$} \{0,1\}^\lambda$
53 $\quad\quad$ **if** $\pi_r = \mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$ **abort**
54 $\quad\quad \texttt{SK}[\text{sID}] \coloneqq$ "reject"
55 $\quad\quad$ **return** $\bot$
56 $(\mathsf{Msg}_{\mathsf{R}}[\text{sID}], \mathsf{Msg}_{\mathsf{I},2}[\text{sID}]) \coloneqq (m_r, \pi_i)$
57 $\texttt{SK}[\text{sID}] \coloneqq K$
58 **return** π_i

Fig. 8. Adversary $\mathcal{B}$ against OW-VwFS. $\mathcal{A}$ has access to oracles $\text{O} \coloneqq \{\text{SESSION}_{\mathsf{I}}, \text{SESSION}_{\mathsf{R}}, \text{DER}_{\mathsf{I}}, \text{DER}_{\mathsf{R}}, \text{REVEAL}, \text{CORR}, \text{TEST}, \mathsf{G}_{\mathsf{I}}, \mathsf{G}_{\mathsf{R}}, \mathsf{H}\}$. Helper procedures FRESH and VALID are defined in Fig. 3. Oracles DER_{R}, TEST and G_{R} are defined in Fig. 9, and G_{I}, H are defined analogously.

or not. If the peer of the session has not been corrupted yet, then this is a valid session and $\mathcal{B}$ outputs (sID, k) as solution in its own game. Otherwise, it proceeds. Oracle DER_{R} is simulated similarly, looking at G_{I} instead of G_{R}.

Oracle REVEAL and CORR can be simulated in a straightforward way. The latter requires $\mathcal{B}$ to query its own oracle CORR'. Queries to TEST will always return the real session key. Note that this is a perfect simulation since session keys are perfectly hidden unless QueryRO happens in which case $\mathcal{B}$ stops because it breaks OW-VwFS security.

It remains to describe the simulation of random oracles. In Fig. 9 we give a description of G_{R}. G_{I} and H are simulated in the same way. As in G_3, $\mathcal{B}$ first checks whether there exists an entry with the special symbol $\diamond$. If this is the case, it finds the corresponding sID and uses the KVER oracle to check whether the key k provided by $\mathcal{A}$ belongs to this session. Since $\diamond$ is used to mark sessions

$\underline{\text{DER}_{\mathsf{R}}(\text{sID}, \pi_i)}$
00 **if** $\mathtt{SK}[\text{sID}] \neq \bot$ **or** $\mathtt{Type}[\text{sID}] \neq$ "Re" **return** $\bot$
01 $(i, r) := (\mathtt{Init}[\text{sID}], \mathtt{Resp}[\text{sID}])$
02 $\mathtt{peerPreCor}[\text{sID}] := \mathtt{cor}[i]$
03 **if** $\exists \text{sID}'$ s. t. $\mathtt{ctxt}[\text{sID}'] = \mathtt{ctxt}[\text{sID}]$
04 **if** $\pi_i \neq \mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$
05 $\mathtt{SK}[\text{sID}] :=$ "reject"
06 **return** $\bot$
07 $K := \mathsf{H}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$
08 **else**
09 **if** $\exists k$ s. t. $\mathsf{G}_{\mathsf{I}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] = \pi_i$ **and** $\text{KVER}(k, \text{sID})$
10 **if** $\mathtt{peerPreCor}[\text{sID}] =$ **false**
11 Stop with (sID, k)
12 $K := \mathsf{H}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$
13 **else**
14 $\mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] \xleftarrow{\$} \{0,1\}^{\lambda}$
15 **if** $\pi_i = \mathsf{G}_{\mathsf{I}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$ **abort**
16 $\mathtt{SK}[\text{sID}] :=$ "reject"
17 **return** $\bot$
18 $(\mathtt{Msg}_{\mathsf{I},2}[\text{sID}], \mathtt{SK}[\text{sID}]) := (m_{i,2}, K)$
19 **return** ε

$\underline{\mathsf{G}_{\mathsf{R}}(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)}$
20 **if** $\mathsf{G}_{\mathsf{R}}[\diamond, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] = \pi \neq \bot$
21 $\mathcal{S} := \{\text{sID} \mid \mathtt{ctxt}[\text{sID}] = (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)\}$
22 **for** $\text{sID} \in \mathcal{S}$
23 **if** $\text{KVER}(\text{sID}, k)$
24 Stop with (sID, k)
25 **elseif** $\mathsf{G}_{\mathsf{R}}[\oplus, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] = \pi \neq \bot$
26 Find sID s. t. $\mathtt{ctxt}[\text{sID}] = (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$
27 **if** $\text{KVER}(\text{sID}, k)$
28 Replace $\oplus$ with k
29 **return** π
30 **if** $\mathsf{G}_{\mathsf{R}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] = \pi \neq \bot$
31 **return** π
32 $\pi \xleftarrow{\$} \{0,1\}^{\lambda}$
33 $\mathsf{G}_{\mathsf{R}}[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r] := \pi$
34 **return** π

$\underline{\text{TEST}(\text{sID})}$
35 **if** $\text{sID} \in \mathcal{S}_{\mathtt{test}}$ **return** $\bot$
36 **if** $\mathtt{SK}[\text{sID}] \in \{\bot, \text{"reject"}\}$ **return** $\bot$
37 $\mathcal{S}_{\mathtt{test}} := \mathcal{S}_{\mathtt{test}} \cup \{\text{sID}\}$
38 **return** $\mathtt{SK}[\text{sID}]$

Fig. 9. Oracles DER_{R}, TEST and G_{R} for adversary $\mathcal{B}$. G_{I} and H are defined analogously to G_{R}.

that have a (partially) matching session, $\mathcal{B}$ can always use this key to win the OW-VwFS game. If there is no entry with $\diamond$ but one with $\oplus$, $\mathcal{B}$ again queries the KVER oracle, but this time it updates the corresponding entry with the correct key (if KVER returns **true**). This way, $\mathcal{B}$ can perfectly simulate non-test sessions. If none of these cases happen or KVER has returned **false**, then $\mathcal{B}$ proceeds as usual by lazy sampling.

This concludes the description of $\mathcal{B}$. Note that if QueryRO happens in game G_3, i. e., there exists a random oracle query for a fresh and valid session with correct key k, then $\mathcal{B}$ wins game OW-VwFS. We get $\Pr[\mathsf{QueryRO}] \leq \varepsilon' + S \cdot \delta$.

Further, note that $\mathcal{B}$ issues at most $(S + Q_{\mathsf{G}_{\mathsf{I}}} + Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{H}})$ to KVER since we have excluded collisions of tags in the first game. The number of queries to all other oracles is preserved. This completes the proof of Theorem 1.

AKE WITH KEY CONFIRMATION AGAINST STATE REVEAL. Based on $\mathsf{AKE}_{\mathsf{KC}}$, we build a three-message AKE protocol $\mathsf{AKE}_{\mathsf{stKC}}$ that is secure against state-reveal attacks (cf. Definition 2). Since $\mathsf{AKE}_{\mathsf{stKC}}$ has a similar structure with $\mathsf{AKE}_{\mathsf{KC}}$, we follow the notations used in defining $\mathsf{AKE}_{\mathsf{KC}}$ (cf. Fig. 5). An overview of $\mathsf{AKE}_{\mathsf{stKC}}$ is given in Fig. 10.

$\mathsf{AKE}_{\mathsf{stKC}}$ uses the state-encryption technique [21,30] to protect session states. Concretely, let $\mathsf{G}_{\mathsf{stI}} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \to \{0,1\}^{d_{\mathsf{I}}}$ and $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \to \{0,1\}^{d_{\mathsf{R}}}$ be two hash functions. We assume that any initiator session state of the underlying two-message AKE protocol AKE' can be encoded as a d_{I}-bit string and $d_{\mathsf{R}} = 2\lambda$ (the length of key confirmation tag plus the length of session key derived by AKE'). $\mathsf{AKE}_{\mathsf{stKC}}$ proceeds the same as $\mathsf{AKE}_{\mathsf{KC}}$ except that (1) the long-term secret key of user i in $\mathsf{AKE}_{\mathsf{stKC}}$ also include a uniformly random key

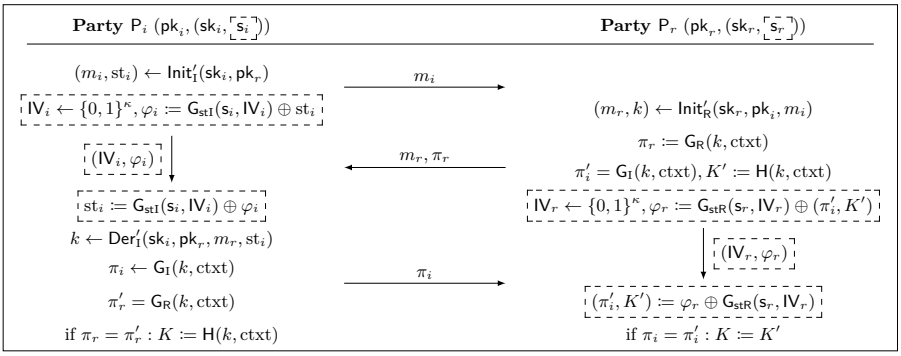

Fig. 10. AKE protocol $\mathsf{AKE}_{\mathsf{stKC}}$ from AKE', key confirmation, and state encryption. The context is defined as $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$. G_{I}, G_{R}, and H are independent random oracles. Dashed parts show how we use the state encryption technique to protect the session states. $\mathsf{G}_{\mathsf{stI}}$ and $\mathsf{G}_{\mathsf{stR}}$ are independent random oracles used for state encryption.

$\mathsf{s}_i \in \{0,1\}^\kappa$, and (2) each session will sample a one-time key IV uniformly at random and encrypt the session state of $\mathsf{AKE}_{\mathsf{KC}}$ via XORing with the one-time pad $\mathsf{G}_{\mathsf{stI}}(\mathsf{s}_i, \mathsf{IV})$. Now the session state (that the adversary can reveal in the state-reveal AKE model) is (IV, φ). Dashed parts in Fig. 10 shows how this technique works.

Correctness. Similar to $\mathsf{AKE}_{\mathsf{KC}}$, the correctness of $\mathsf{AKE}_{\mathsf{stKC}}$ follows directly from the correctness of AKE'. If AKE' is $(1-\delta)$-correct, then so is $\mathsf{AKE}_{\mathsf{stKC}}$.

Security. In Theorem 2, we prove IND-FS-St security of $\mathsf{AKE}_{\mathsf{KC}}$ based on OW-VwFS-St security of AKE' and modeling G_{I}, G_{R}, H, $\mathsf{G}_{\mathsf{stI}}$, and $\mathsf{G}_{\mathsf{stR}}$ as random oracles. Here we sketch the proof idea. By using the state encryption technique, the adversary cannot learn the unencrypted states of the underlying two-message AKE, unless it reveals the encrypted session state and corrupts the owner of the session. But this makes the session invalid and thus, it cannot be tested. Therefore, for valid sessions, state-reveal queries do not give any advantage to the adversary, and thus we can use the proof idea of Theorem 1. The full proof of Theorem 2 is postponed to our full version [28, Appendix C].

Theorem 2. *Let* AKE' *be* $(1-\delta)$*-correct and have public keys with* γ *bits of entropy and messages with* α *bits of entropy. Let* $\mathsf{AKE}_{\mathsf{stKC}}$ *be as defined in Fig. 5, where* $\mathsf{G}_{\mathsf{I}}, \mathsf{G}_{\mathsf{R}} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$, $\mathsf{G}_{\mathsf{stI}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_{\mathsf{I}}}$*, and* $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_{\mathsf{R}}}$ *are modeled as random oracles. For every adversary* $\mathcal{A}$ *that breaks the* $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{\mathrm{ST}})$-IND-FS-St*-security of* $\mathsf{AKE}_{\mathsf{stKC}}$*, there exists an adversary* $\mathcal{B}$ *that breaks the* $(t', \varepsilon', \mu, S, Q_{\mathrm{COR}}, Q_{Ver}, S)$-OW-VwFS-St *security of* AKE' *with*

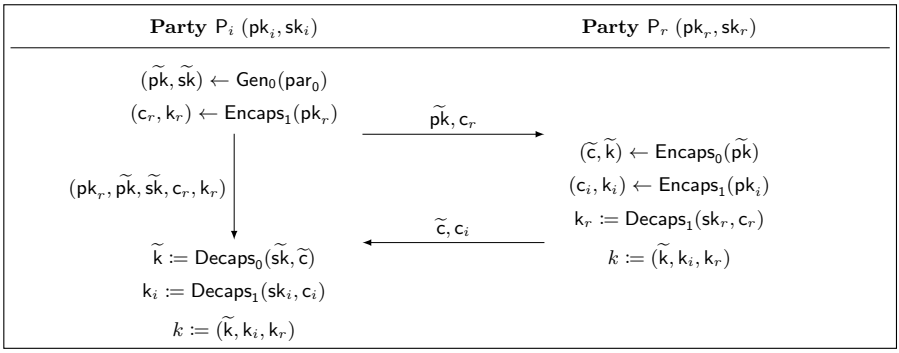

Fig. 11. AKE protocol $\mathsf{AKE}'_{\mathsf{kem}}$ from KEM schemes KEM_1, KEM_0.

$t' \approx t$ *and*

$$\varepsilon \leq \varepsilon' + 2S \cdot \delta + (\mu^2 + S^2 + \mu Q_{\mathsf{G}_{\mathsf{stI}}} + 2SQ_{\mathsf{G}_{\mathsf{stI}}}) \cdot 2^{-\kappa} + \mu^2 \cdot 2^{-\gamma} + (S + S^2) \cdot 2^{-\lambda} + (Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{G}_{\mathsf{I}}} + Q_{\mathsf{H}} + S) \cdot S \cdot 2^{-\alpha},$$

where Q_{h} *is the number of queries to the respective random oracle* h *and* $Q_{Ver} \leq S + Q_{\mathsf{G}_{\mathsf{I}}} + Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{H}}$.

6 Applying Our Results to Existing Protocols

We first show how to construct verifiable AKE from KEMs which gives us tight AKE with key confirmation and perfect forward secrecy from lattices and DDH. The advantage is that we do not have to consider random oracles and the proofs are comparably simpler than those of full AKE security. We then show how we can recover the optimal tightness bound for the CCGJJ protocol [10] using our modular transformation rather than that of [18].

6.1 AKE from KEMs

We provide results for KEM-based AKE secure without and with state reveal, where the former allows for weaker assumptions. The protocol, denoted by $\mathsf{AKE}'_{\mathsf{kem}}$, to which we want to apply our compiler from the previous section is given in Fig. 11. Each party holds long-term keys of a KEM scheme KEM_1 and in each session, an ephemeral key using KEM_0 is exchanged. The session key then simply consists of three KEM keys. We also denote its variant with key confirmation by $\mathsf{AKE}_{\mathsf{kem}}$ (i. e., combining Fig. 11 with Fig. 5) and the one resisting state reveals by $\mathsf{AKE}_{\mathsf{st,kem}}$ (i. e., combining Fig. 11 with Fig. 10).

One-Way Security of KEM. Depending on whether the KEM is used for long-term keys or ephemeral keys and whether state reveals are allowed,

we need a different variant of one-way security: multi-user one-way security under plaintext checking and ciphertext validity attacks without (OW-PCVA) or with corruptions (OW-PCVA-C), and with corruptions and reveal queries (OW-PCVA-CR). These notions are weaker variants of OW-ChCCA security from Pan, Wagner and Zeng [30] and are tightly implied by OW-ChCCA. The formal security definitions are given in our full version [28, Appendix A].

Analysis of $\mathsf{AKE}'_{\mathsf{kem}}$ and $\mathsf{AKE}_{\mathsf{kem}}$. We prove that $\mathsf{AKE}'_{\mathsf{kem}}$ protocol is a secure verifiable AKE protocol. When not considering state reveals, we can use weaker assumptions, namely OW-PCVA and OW-PCVA-C. We also prove security with state-reveals which uses the definition of OW-PCVA-CR security. We then apply Theorem 1 resp. Theorem 2 to obtain $\mathsf{AKE}_{\mathsf{kem}}$ which has full forward secrecy.

Correctness and Entropy. Let KEM_0 be $(1-\delta_0)$-correct and have public keys with γ_0 bits of entropy and messages with α_0 bits of entropy. Let KEM_1 be $(1-\delta_1)$-correct and have public keys with γ_1 bits of entropy and messages with α_1 bits of entropy. Then $\mathsf{AKE}'_{\mathsf{kem}}$ is $(1-\delta_0-2\delta_1)$-correct. Further, $\mathsf{AKE}'_{\mathsf{kem}}$ has public keys with γ_1 bits of entropy and messages with at least $\min(\gamma_0, \alpha_0, \alpha_1 - 1)$ bits of entropy.

We now establish OW-VwFS and OW-VwFS-St security of $\mathsf{AKE}'_{\mathsf{kem}}$ and defer the proofs to our full paper [28, Appendix D.1].

Lemma 1. *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{Ver})$-OW-VwFS security of $\mathsf{AKE}'_{\mathsf{kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver})$-OW-PCVA security of KEM_0 and $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{\mathrm{COR}})$-OW-PCVA-C security of KEM_1 with $t_1 \approx t_2 \approx t$ and $\varepsilon \leq \varepsilon_1 + \varepsilon_2$.*

Lemma 2. *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{Ver}, Q_{\mathrm{ST}})$-OW-VwFS-St security of $\mathsf{AKE}'_{\mathsf{kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver}, Q_{\mathrm{ST}})$-OW-PCVA-C security of KEM_0 and $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{\mathrm{COR}}, Q_{\mathrm{ST}})$-OW-PCVA-CR security of KEM_1 with $t_1 \approx t_2 \approx t$ and $\varepsilon \leq \varepsilon_1 + \varepsilon_2$.*

We now add key confirmation to $\mathsf{AKE}'_{\mathsf{kem}}$ as described in Fig. 5 resp. Fig. 10. The following theorem then follows from combining Theorem 1 with Lemma 1 resp. Theorem 2 with Lemma 2.

Theorem 3. *Let KEM_0 be $(1-\delta_0)$-correct and have public keys with γ_0 bits of entropy and messages with α_0 bits of entropy. Let KEM_1 be $(1-\delta_1)$-correct and have public keys with γ_1 bits of entropy and messages with α_1 bits of entropy. Let $\mathsf{AKE}_{\mathsf{kem}}$ resp. $\mathsf{AKE}_{\mathsf{st,kem}}$ be defined as described above by combining Fig. 11 with Fig. 5 resp. Fig. 10, where $\mathsf{G}_{\mathsf{I}}, \mathsf{G}_{\mathsf{R}} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$, $\mathsf{G}_{\mathsf{stI}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_{\mathrm{I}}}$ and $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_{\mathrm{R}}}$ are modeled as random oracles. Let Q_{h} be the number of queries to the respective random oracle h.*

For any $\mathcal{A}$ against the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}})$-IND-FS-security of $\mathsf{AKE}_{\mathsf{kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver})$-OW-PCVA security

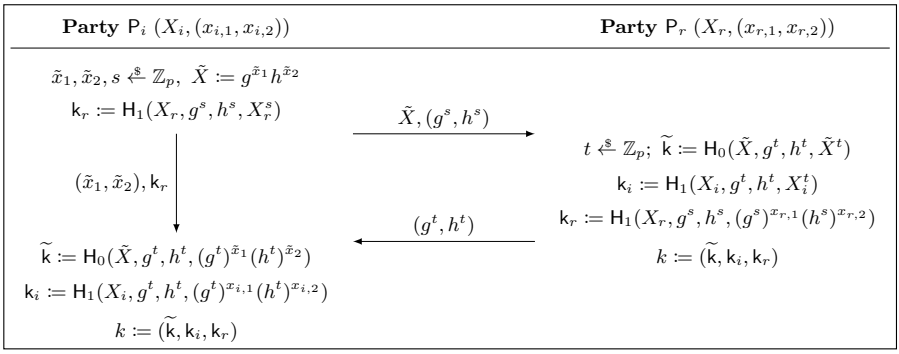

Fig. 12. AKE protocol JKRS. H_0, H_1 are independent random oracles. Protocol $\mathsf{JKRS}_{\mathsf{KC}}$ is obtained by adding the transformation from Fig. 10.

of KEM_0 *and* $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{\mathrm{COR}})$-OW-PCVA-C *security of* KEM_1*, where* $Q_{Ver} \leq S + Q_{\mathsf{G_I}} + Q_{\mathsf{G_R}} + Q_{\mathsf{H}}$*, with* $t_1 \approx t_2 \approx t$ *and*

$$\varepsilon \leq \varepsilon_1 + \varepsilon_2 + 2S \cdot (\delta_0 + 2\delta_1) + (S + S^2) \cdot 2^{-\lambda} + \mu^2 \cdot 2^{-\gamma_1} + S(S + Q_{\mathsf{G_I}} + Q_{\mathsf{G_R}} + Q_{\mathsf{H}}) \cdot (2^{-\gamma_0} + 2^{-\alpha_0} + 2^{-\alpha_1 + 1}) \ .$$

Further, for every adversary $\mathcal{A}$ *that breaks the* $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{\mathrm{ST}})$-IND-FS-St-*security of* $\mathsf{AKE}_{\mathsf{st},\mathsf{kem}}$*, there exist adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *that break* $(t_1, \varepsilon_1, S, S, S, Q_{Ver}, Q_{\mathrm{ST}})$-OW-PCVA-C *security of* KEM_0 *and* $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{\mathrm{COR}}, Q_{\mathrm{ST}})$-OW-PCVA-CR *security of* KEM_1*, where* $Q_{Ver} \leq S + Q_{\mathsf{G_I}} + Q_{\mathsf{G_R}} + Q_{\mathsf{H}}$*, with* $t_1 \approx t_2 \approx t$ *and*

$$\varepsilon \leq \varepsilon_1 + \varepsilon_2 + 2S \cdot (\delta_0 + 2\delta_1) + (\mu^2 + S^2 + \mu Q_{\mathsf{G_{stI}}} + 2S Q_{\mathsf{G_{stI}}}) \cdot 2^{-\kappa} + \mu^2 \cdot 2^{-\gamma_1} + (S + S^2) \cdot 2^{-\lambda} + S(Q_{\mathsf{G_R}} + Q_{\mathsf{G_I}} + Q_{\mathsf{H}} + S) \cdot (2^{-\gamma_0} + 2^{-\alpha_0} + 2^{-\alpha_1 + 1}) \ .$$

Instantiation with Non-Committing KEM. We can use a non-committing KEM as defined in [21] to instantiate a verifiable AKE protocol very efficiently, e. g., from DDH (cf. protocol JKRS in Fig. 12). We can easily show that a non-committing KEM implies OW-PCVA-CR security of that KEM. In our full version [28, Appendix A], we recall the formal definition of NC-CCA security for KEMs from [21] and show the implication. Adding key confirmation as described in Fig. 10 then yields protocol $\mathsf{JKRS}_{\mathsf{KC}}$.

Security of $\mathsf{JKRS}_{\mathsf{KC}}$. We now establish security of protocol $\mathsf{JKRS}_{\mathsf{KC}}$. Since the JKRS protocol is perfectly correct, so is $\mathsf{JKRS}_{\mathsf{KC}}$. Further, public keys and messages have $\log(p)$ bits entropy. Security is based on the DDH assumption which asks to distinguish between (g^x, g^y, g^{xy}) and (g^x, g^y, g^z) for $x, y, z \xleftarrow{\$} \mathbb{Z}_p$.

Theorem 4. *Let* $\mathsf{JKRS}_{\mathsf{KC}}$ *be defined as in Fig. 12, where* $\mathsf{G_I}, \mathsf{G_R} : \{0,1\}^* \to \{0,1\}^\lambda$*,* $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$*,* $\mathsf{H}_0 : \{0,1\}^* \to \mathsf{KEM}_0.\mathcal{K}$*,* $\mathsf{H}_1 : \{0,1\}^* \to \mathsf{KEM}_1.\mathcal{K}$*,*

$\mathsf{G}_{\mathsf{stI}} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \to \{0,1\}^{d_{\mathrm{I}}}$, *and* $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \to \{0,1\}^{d_{\mathrm{R}}}$ *are modeled as random oracles.*

For every adversary $\mathcal{A}$ *that breaks the* $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{\mathrm{ST}})$-$\mathsf{IND\text{-}FS\text{-}St}$-*security of* $\mathsf{JKRS}_{\mathsf{KC}}$*, there exists an adversary* $\mathcal{B}$ *that breaks* (t', ε')-DDH *with* $t' \approx t$ *and*

$$\varepsilon \leq \varepsilon' + (\mu^2 + S^2 + \mu Q_{\mathsf{G}_{\mathsf{stI}}} + 2SQ_{\mathsf{G}_{\mathsf{stI}}}) \cdot 2^{-\kappa} + \mu^2 \cdot 2^{-\log(p)} + (S + S^2) \cdot 2^{-\lambda} + S(Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{G}_{\mathrm{I}}} + Q_{\mathsf{H}} + Q_{\mathsf{H}_0} + Q_{\mathsf{H}_1} + S + 1) \cdot 2^{-\log(p)} \ ,$$

where Q_{h} *is the number of queries to the respective random oracle* h*.*

The theorem follows from [28, Theorem 6], Lemma 2 and Theorem 2 in combination with [21, Theorem 5], where the latter deals with the optimization that only one ciphertext is sent in the second round.

Instantiation from Lattices. We can also instantiate the KEM-based verifiable AKE protocol using lattices assumptions. The scheme $\mathsf{KEM}_{\mathsf{LWE}}$ described in [30, Section 3] satisfies $\mathsf{OW\text{-}ChCCA}$ security which implies $\mathsf{OW\text{-}PCVA\text{-}CR}$ security. This gives us an AKE protocol with key confirmation from LWE secure in the random oracle model.

6.2 The CCGJJ Protocol and Its Isogeny-Based Variant

It is easy to see that the core protocol from Cohn-Gordon et al. (CCGJJ) [10] is a verifiable AKE protocol, ignoring the session key hash. For completeness, we provide a formal treatment in our full paper [28, Appendix F].

Isogeny-based AKE. The isogeny-based AKE protocol which was independently analyzed by de Kock, Gjøsteen and Veroni [23] and Kawashima et al. [22] follows the same blueprint as the CCGJJ protocol, relying on the group action structure of CSIDH [8] rather than prime-order groups. Thus, we also get an AKE protocol with key confirmation from isogenies, based on the same assumptions as the analysis in [22,23]. This is particularly interesting because the only group action based and tightly-secure signature scheme supporting adaptive corruptions [29] is rather inefficient.

7 KEM-Based AKE with Key Confirmation in the QROM

We analyze FS via key confirmation in the quantum random oracle model (QROM). Following [31], we use IND-CCA-secure KEMs in the multi-user, multi challenge settings as building blocks. By the key confirmation technique, we lift the result of Pan, Wagner, and Zeng [31] to FS in the QROM. The work of Pan, Wagner, and Zeng only achieves weak FS in the QROM. Our result not only preserves the security loss of their protocol, but also achieves FS and allows multiple Test queries with single challenge bit, while [31] allows at most one single Test query.

The MC-IND-CCA and MUC-IND-CCA security definitions of KEMs are given in our full paper [28, Appendix A]. We use notations introduced in Sect. 5 to present our protocol $\mathsf{AKE}_{\mathsf{kem}}$. Let KEM_1 and KEM_0 be two KEM schemes and $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$, and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ be hash functions, where λ is the length of key confirmation tags and $\mathcal{K}$ is the key space of $\mathsf{AKE}_{\mathsf{kem}}$. An overview of our AKE construction $\mathsf{AKE}_{\mathsf{kem}}$ is given in Fig. 13. $\mathsf{AKE}_{\mathsf{kem}}$ is essentially the KEM-based AKE protocol in [31] adding key confirmation, namely, it is obtained from combining Fig. 11 and Fig. 5.

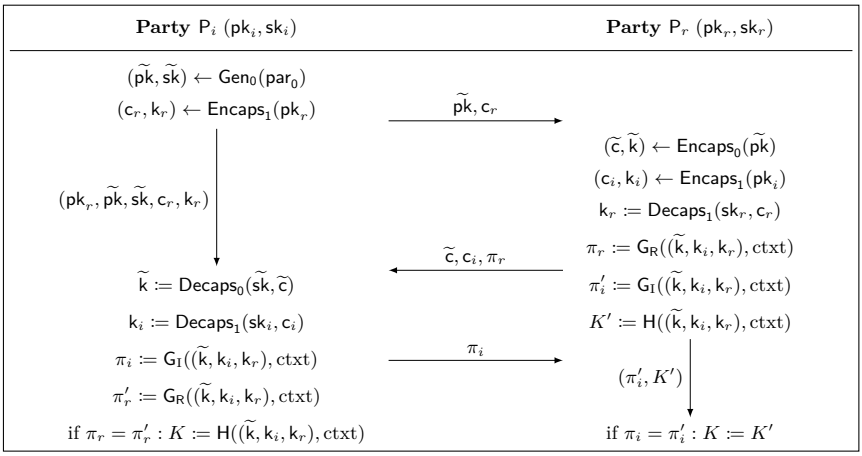

Fig. 13. AKE protocol $\mathsf{AKE}_{\mathsf{kem}}$ from KEM schemes KEM_1, KEM_0, and key confirmation. The context is defined as $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$. G_I, G_R, and H are independent random oracles.

Correctness. The correctness of $\mathsf{AKE}_{\mathsf{kem}}$ is due to KEM_1 and KEM_0. Each session of $\mathsf{AKE}_{\mathsf{kem}}$ includes two ciphertexts of KEM_1 and one ciphertext of KEM_0. If KEM_1 is $(1-\delta_1)$-correct and KEM_0 is $(1-\delta_0)$-correct, then by the union bound, $\mathsf{AKE}_{\mathsf{kem}}$ is $(1 - 2\delta_1 - \delta_0)$-correct.

Security. We prove IND-FS security of $\mathsf{AKE}_{\mathsf{kem}}$ based on the MC-IND-CCA security of KEM_1, the MUC-IND-CCA security of KEM_0, and modeling $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R}$, and H as quantum-accessible random oracles, as stated in Theorem 5. The proof of Theorem 5 is postponed to our full version [28, Appendix E].

Theorem 5. *Let* KEM_0 *be* $(1-\delta_0)$*-correct and have public keys with* γ_0 *bits of entropy and messages with* α_0 *bits of entropy. Let* KEM_1 *be* $(1-\delta_1)$*-correct and have public keys with* γ_1 *bits of entropy and messages with* α_1 *bits of entropy.* $\mathcal{K}_0$ *and* $\mathcal{K}_1$ *are the KEM key spaces of* KEM_0 *and* KEM_1*, respectively.*

Let $\mathsf{AKE}_{\mathsf{kem}}$ *be as defined in Fig. 13, where Let* $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$ *and* $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$*. For every adversary* $\mathcal{A}$ *that breaks the*

$(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}})$-IND-FS-*security of* $\mathsf{AKE}_{\mathsf{KC}}$*, there exists an adversary* $\mathcal{B}_0$ *that breaks the* $(t'_0, \varepsilon'_0, S, S)$-MUC-IND-CCA *security of* KEM_0 *and an adversary* $\mathcal{B}_1$ *that breaks the* $(t'_1, \varepsilon'_1, S)$-MC-IND-CCA *security of* KEM_1 *with* $t'_0 \approx t'_1 \approx t$ *and*

$$\varepsilon \leq 2\varepsilon'_0 + 2\mu\varepsilon'_1 + 2S(\delta_0 + \mu\delta_1) + \mu^2 2^{-\gamma_1} + \mu S 2^{-\lambda+1} + S^2(2^{-\alpha_1} + 2^{-\gamma_0} + 2^{-\alpha_0}) + \frac{2\mu(Q_{\mathsf{G_R}} + Q_{\mathsf{G_I}})\sqrt{S}}{\sqrt{|\mathcal{K}_1|}} + \frac{2Q_{\mathsf{H}}\sqrt{S}}{\sqrt{|\mathcal{K}_0|}},$$

where $Q_{\mathsf{G_I}}, Q_{\mathsf{G_R}}$ *and* Q_{H} *are the number of quantum-superposition queries to* $\mathsf{G_I}$, $\mathsf{G_R}$ *and* H.

Remark 1 (Implicit Rejection). Following [31], when proving Theorem 5, we assume KEM_1 and KEM_0 have implicit rejection [5], namely, if the input ciphertext is invalid, then the decapsulation algorithm returns a pseudorandom KEM key. We use implicit-rejection KEM because it simplifies our AKE proof.

To adapt the proof of Theorem 5 to the one that uses explicit-rejection KEMs, we can add extra codes in the games sequence to deal with explicit rejections from KEM. Concretely, upon receiving an invalid KEM ciphertext, the session oracle (e.g., $\textsc{Session}_{\mathsf{R}}$, $\textsc{Der}_{\mathsf{R}}$, or $\textsc{Der}_{\mathsf{I}}$) simply sets the session key as "reject" and returns $\perp$. This can be tightly simulated by MUC-IND-CCA and MC-IND-CCA secure KEMs with explicit rejection, and thus the security bound in Theorem 5 also applies to explicit-rejection KEMs.

Remark 2 (Instantiations with LWE). In [31], Pan et al. proposed lattice-based instantiations of MC-IND-CCA-secure KEM and MUC-IND-CCA-secure KEM that have (almost-)tight reduction from the well-known Learning With Errors (LWE) problem in the QROM. Here we only discuss the security loss of these KEM schemes and give the final security loss of our AKE protocol instantiated with these KEM schemes.

Let $\varepsilon_{\mathsf{lwe}}$ be the best computational advantage against LWE assumptions and λ be the security parameter (which decides the security level, the length of message, etc.). The two KEM schemes proposed in [31] have asymptotic bounds $\varepsilon'_1 \leq \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}}$ and $\varepsilon'_0 \leq \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}}$, where ε'_1 and ε'_0 are the computational advantages against MC-IND-CCA security and MUC-IND-CCA security of the KEM schemes in [31], respectively. By combining these bounds with the bounds given in Theorem 5, we have

$$\varepsilon \leq \Theta(\lambda) + \Theta(\mu) \cdot \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}} = \Theta(\mu) \cdot \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}},$$

where ε is the computational advantage against the resulting AKE protocol. This gives us a session-tight and square-root-tight (namely, does not suffer from the square-root security loss) LWE-based instantiation of AKE with full forward secrecy in the QROM.

Acknowledgements. We thank the anonymous reviewers for their valuable comments on better motivating our works and comparisons with the related work. Doreen Riepel was supported in part by Bellare's KACST grant. Jiaxin Pan was supported in part by the Research Council of Norway (RCN) under Project No. 324235, and Runzhi Zeng were supported by the same project from RCN.

References

1. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_6
2. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: security proofs and improvements. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_18
3. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21
4. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_25
5. Bernstein, D.J., Persichetti, E.: Towards KEM unification. Cryptology ePrint Archive, Report 2018/526 (2018). https://eprint.iacr.org/2018/526
6. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
7. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_28
8. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-03332-3_15
9. Chen, J., Wee, H.: Fully, (almost) tightly secure IBE and dual system groups. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 435–460. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_25
10. Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 767–797. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-26954-8_25
11. Davis, H., Günther, F.: Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 2021, Part II. LNCS, vol. 12727, pp. 448–479. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-78375-4_18
12. Delpech de Saint Guilhem, C., Fischlin, M., Warinschi, B.: Authentication in key-exchange: definitions, relations and composition. In: Jia, L., Küsters, R. (eds.) CSF 2020 Computer Security Foundations Symposium, pp. 288–303. IEEE Computer Society Press (2020). https://doi.org/10.1109/CSF49147.2020.00028
13. Diemert, D., Gellert, K., Jager, T., Lyu, L.: More efficient digital signatures with tight multi-user security. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 1–31. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-75248-4_1

14. Diemert, D., Jager, T.: On the tight security of TLS 1.3: theoretically sound cryptographic parameters for real-world deployments. J. Cryptol. **34**(3), 30 (2021). https://doi.org/10.1007/s00145-021-09388-x
15. Fischlin, M., Günther, F., Schmidt, B., Warinschi, B.: Key confirmation in key exchange: a formal treatment and implications for TLS 1.3. In: 2016 IEEE Symposium on Security and Privacy, pp. 452–469. IEEE Computer Society Press, May 2016. https://doi.org/10.1109/SP.2016.34
16. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 254–271. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_17
17. Gay, R., Hofheinz, D., Kiltz, E., Wee, H.: Tightly CCA-secure encryption without pairings. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 1–27. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_1
18. Gellert, K., Gjøsteen, K., Jacobsen, H., Jager, T.: On optimal tightness for key exchange with full forward secrecy via key confirmation. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. LNCS, Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-38551-3_10
19. Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 95–125. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-96881-0_4
20. Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 389–422. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-45388-6_14
21. Jager, T., Kiltz, E., Riepel, D., Schäge, S.: Tightly-secure authenticated key exchange, revisited. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 117–146. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-77870-5_5
22. Kawashima, T., Takashima, K., Aikawa, Y., Takagi, T.: An efficient authenticated key exchange from random self-reducibility on CSIDH. In: Hong, D. (ed.) ICISC 2020. LNCS, vol. 12593, pp. 58–84. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-68890-5_4
23. de Kock, B., Gjøsteen, K., Veroni, M.: Practical isogeny-based key-exchange with optimal tightness. In: Dunkelman, O., Jacobson Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 451–479. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-81652-0_18
24. Krawczyk, H.: HMQV: a high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_33
25. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (Nov (2007)
26. Liu, X., Liu, S., Gu, D., Weng, J.: Two-pass authenticated key exchange with explicit authentication and tight security. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 785–814. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-64834-3_27

27. Pan, J., Qian, C., Ringerud, M.: Signed (group) Diffie-Hellman key exchange with tight security. J. Cryptol. **35**(4), 26 (2022). https://doi.org/10.1007/s00145-022-09438-y
28. Pan, J., Riepel, D., Zeng, R.: Key exchange with tight (full) forward secrecy via key confirmation. In: Cryptology ePrint Archive (2024)
29. Pan, J., Wagner, B.: Lattice-based signatures with tight adaptive corruptions and more. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part II. LNCS, vol. 13178, pp. 347–378. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-030-97131-1_12
30. Pan, J., Wagner, B., Zeng, R.: Lattice-based authenticated key exchange with tight security. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. LNCS, Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-38554-4_20
31. Pan, J., Wagner, B., Zeng, R.: Tighter security for generic authenticated key exchange in the QROM. In: ASIACRYPT 2023. LNCS, Springer, Heidelberg (2023). https://doi.org/10.1007/978-981-99-8730-6_13, https://eprint.iacr.org/2023/1380
32. Unruh, D.: Revocable quantum timed-release encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 129–146. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_8