# Tightly-Secure Authenticated Key Exchange, Revisited

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

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

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

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

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

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

## 1 Introduction

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

Tightness of AKE. When proving a cryptographic scheme secure, one commonly describes a security reduction which transforms an adversary $\mathcal{A}$ that breaks the cryptographic scheme into an adversary $\mathcal{B}$ that solves some underlying complexity assumption. For instance, if $\mathcal{A}$ has advantage $\epsilon$ in breaking the scheme and $\mathcal{B}$ solves the problem with advantage $\epsilon' = \epsilon/L$, then $L$ is called the reduction's security loss. If $L$ is constant (and in particular independent of the number of $\mathcal{A}$'s oracle queries) and additionally the running times of $\mathcal{A}$ and $\mathcal{B}$ are roughly identical, then we say the reduction is *tight*. Especially when choosing protocol-specific system parameters, the tightness of a security proof plays an important role. In the security model for AKE the attacker can actively control all messages sent between the involved parties and is additionally allowed to reveal secret information such as a long-term secret key (by corrupting a party), or a session key. The adversary breaks security if it is able to distinguish non-revealed session keys from random.

Multi-Challenge Security definitions. The standard and well established security notion in the context of multiple challenges [3,18,20,10] is "Single-Bit Guess" (SBG) security. The blueprint of a SBG security experiment is as follows. First, the experiment picks a secret random bit $b \in \{0, 1\}$. Next, the adversary is allowed to make multiple (up to, say, $T$) challenge queries. On each challenge query, the experiment returns a "real key" if $b = 0$, and an independent "random key" if $b = 1$. The adversary wins if it can guess the challenge bit $b$ with a probability better than 1/2.

In AKE protocols, challenge queries are usually called test queries and non-revealed session keys can be accessed by making multiple calls to a Test oracle. If $b = 0$, a query to Test returns the real challenge key; if $b = 1$, a query to Test returns an independent random challenge key. This notation of multi-challenge SBG security for AKE was first formalized in 2019 by Cohn-Gordon et al. [10]. By conditioning on bit $b$, SBG security is known to be tightly equivalent to (single-bit) "Real-Or-Random" (ROR) security, where the adversary has to distinguish a real game (where all challenge keys output by Test are real) from a random game (where all challenge keys are random). Using the above equivalence, SBG security precisely captures the intuition that *all challenge keys* are simultaneously pseudo-random.

Surprisingly, in the first publication on tightly secure AKE protocols in 2015, Bader et al. [1] defined a different and non-standard "Multi-Bit-Guess" (MBG) AKE security notion. In MBG security, the experiment picks multiple independent challenge bits $b_1, \dots, b_T$ and, on the $i$-th Test query, it returns a real challenge key if $b_i = 0$ and a random challenge key if $b_i = 1$. That is, each of the $T$ challenge keys depends on an independent challenge bit $b_i$. The adversary wins if it can guess correctly one of the $T$ challenge bits $b_{i^*}$ with a probability better than 1/2. We are not aware of any meaningful multi-bit ROR security game that is tightly equivalent to MBG security.[3] This makes it difficult to provide a good intuition of what MBG security tries to model.

[3] If one tries to apply a similar conditioning argument as in the single-bit case, MBG can be shown equivalent to a ROR-type security experiment where in the real game ($b_{i^*} = 0$) the $i^*$-th challenge key output by Test is real and in the random game ($b_{i^*} = 1$) it is random. However, the remaining $T - 1$ keys still depends on the random bits $b_i$ ($i \neq i^*$): the $i$-th challenge key is real if $b_i = 0$ and it is random if $b_i = 1$. Hence, about one half of the challenge keys is expected to be real (the ones

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

First off, as already pointed out, SBG security is the standard and well established security notion in the context of multiple challenges [3,18,20,10]. Cohn-Gordon et al. [10, Section 3] already pointed out that, in the AKE setting, SBG security tightly composes with symmetric primitives, whereas MBG security doesn't. Let us elaborate. AKE is not intended to be used as a stand-alone primitive. Rather, it is naturally composed with symmetric primitives to establish a secure channel [7,24], for example to encrypt (e.g., using AES) a message with the session key. Since SBG security is tightly equivalent to ROR security, it offers precisely the right security interface to switch *all challenge keys at once* from real to random. This step allows to infer the privacy of the encrypted messages from the security properties of the symmetric primitive. MBG security, on the other hand, does not have a meaningful ROR-style security, which makes it difficult to argue about the privacy of the encrypted messages without relying on a hybrid argument. In summary, in the context of tightness of AKE protocols, SBG security is a meaningful notion whereas MBG isn't.

Previous Results. Previous work on tight AKE protocols by Gjøsteen and Jager [21] and Liu at al. [32] exclusively concentrated on the MBG model by Bader et al. [1]. We now give a brief overview of existing AKE protocols in the context of tight SBG security.

- At CRYPTO 2019, Cohn-Gordon et al. [10] presented highly efficient two message AKE protocols with implicit authentication, in the style of HMQV [26] and similar protocols. Their schemes achieve a loss of $O(N)$ in the SBG security model with weak forward secrecy, where $N$ is the number of users. They also extend the impossibility results from [2] to show that a loss of $O(N)$ is unavoidable for many natural protocols (including HMQV [26], NAXOS [28], Kudla-Paterson [27], KEA+ [29], and more) with respect to typical cryptographic security proofs (so-called simple reductions). Furthermore, since their protocol does not feature explicit authentication, a well-known impossibility result applies [26,6,34] and their protocol cannot achieve full forward security.
- Diemert and Jager [16] and independently Davis and Günther [15] considered the three message TLS 1.3 handshake AKE protocol with explicit authentication. Its design follows the standard "$1\times\mathsf{KEM}+2\times\mathsf{SIG}$" (aka. signed Diffie-Hellman) AKE approach [9,14,21,16,15,32]. TLS 1.3, when instantiated with standardized signatures (e.g., RSA-PSS, RSA-PKCS #1 v1.5, ECDSA, or EdDSA), has rather non-tight SBG security with full forward security. But when instantiated with tightly secure signatures in the multi-user setting with adaptive corruptions [1], then SBG security of TLS 1.3 actually becomes tight. Since the TLS 1.3 protocol contains two signatures, the ineffi-

with $b_i = 0$) whereas the other half is random, and the adversary does not have any information on them.

ciency of currently known tightly secure signature schemes [1,21] makes the resulting TLS instantiation very impractical.

### 1.1 The Difficulty of Constructing Tightly Secure AKE

Security models for authenticated key exchange are extremely complex, as they consider very strong adversaries that may modify, drop, or inject messages. Furthermore, usually an adversary may adaptively corrupt users' long-term secrets via Corrupt-queries, session keys via Reveal-queries, and sometimes even ephemeral states of sessions via Rev-State-queries. Security is formalized with multiple Test queries, where the adversary specifies a session, receives back a real key or a random key, and has to distinguish these. This complexity makes achieving tight security challenging, particularly because all the following difficulties must be tackled simultaneously.

The "commitment problem". As explained in more detail in [21], this problem is the reason why nearly all security proofs of classical key exchange protocols have a quadratic security loss. Essentially, the problem is that most AKE protocols have security proofs where a reduction can only extract a solution to a computationally hard problem if an instance of the problem is embedded into the protocol messages of the Tested sessions, but at the same time the reduction is not able to answer Reveal queries for such sessions. The standard way to resolve this is to let the reduction guess the Tested session, and to embed an instance of a computationally hard problem only there. However, this incurs a significant security loss. A tight reduction has to be able to respond to *both* Test and Reveal queries for *every* session.

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

In order to resolve this issue, previous works [1,21] constructed signature schemes based on non-interactive OR-proof systems, which enable a reduction to "know" one out of two signing keys. It is argued that the adversary will forge a signature with respect to the other, unknown key with sufficiently high probability. However, these signature schemes are much less efficient than classical ones, and thus impose a performance penalty on the protocols.

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

### 1.2 Main Contributions

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

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

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

In this work, we answer the two questions to the positive. Following [4,10], we consider SBG security, allowing adaptive corruptions of long-term secrets, adaptive reveals of session keys, and multiple adaptive Test queries. Our model also captures (weak and full) forward security (FS), and prevents key-compromise impersonation and reflection attacks. In comparison to prior work on tightly-secure key exchange [1,21,10,16,15], we consider a model which additionally allows to reveal some internal state information.

Our DDH-Based AKE Protocols. Our two protocols instantiated from $\mathsf{DDH}$ are given in Figure 1. $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ is an implicitly-authenticated two-message protocol $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ in the sense of [26]. It requires the exchange of only five group elements in total, and thus is the first efficient implicitly-authenticated protocol with weak FS that achieves full tightness.

Our second protocol $\mathsf{AKE}_{\mathsf{FS,DDH}}$ achieves full FS. Instead of using the standard "1×$\mathsf{KEM}$+2×$\mathsf{SIG}$" approach, it replaces one of the signatures with a more efficient MAC and an additional KEM ciphertext, which yields a "2×$\mathsf{KEM}$+1×$\mathsf{SIG}$ +1×$\mathsf{MAC}$" construction. When instantiated at "128-bit security" with the most efficient tightly-secure signatures of [21],[4] the communication complexity is 448 bytes, again with ephemeral state reveals. In comparison, the previously most efficient tightly and fully forward-secure protocol with SBG security $\mathsf{TLS}^*$ (which is TLS 1.3 instantiated with the tightly-secure signature of [21]) requires three messages, the transmission of 704 bytes and does not allow state reveals. See Figure 2 for a comparison of our protocols with previous works. Note that the communication bottleneck in all full FS protocols is the number of signatures. For completeness the figure also list previous protocols with tight MBG security [21,32].

[4] The signatures of [21] consist of 2 group elements, 4 elements in $\mathbb{Z}_p$ and 2 hashes in $\{0,1\}^\kappa$. At "128-bit security" this corresponds to 256 bytes per signature.

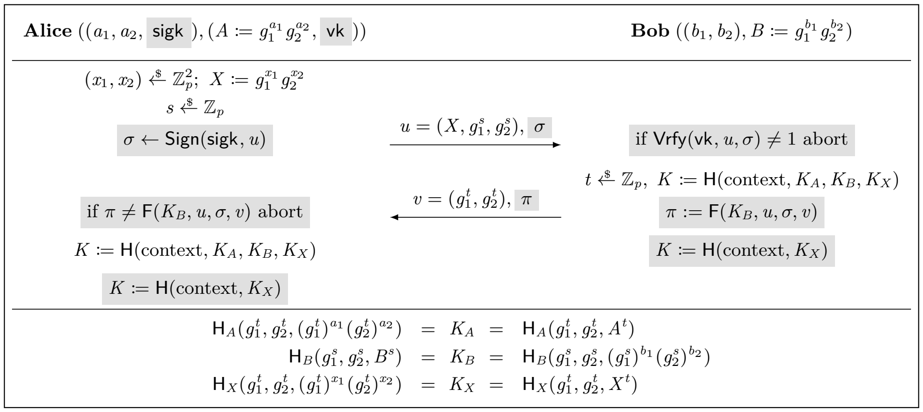
**Alice** $((a_1, a_2, \mathsf{sigk}), (A := g_1^{a_1} g_2^{a_2}, \mathsf{vk}))$
**Bob** $((b_1, b_2), B := g_1^{b_1} g_2^{b_2})$
$(x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2;\ X := g_1^{x_1} g_2^{x_2}$
$s \xleftarrow{\$} \mathbb{Z}_p$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}, u)$
$u = (X, g_1^s, g_2^s), \sigma$
if $\mathsf{Vrfy}(\mathsf{vk}, u, \sigma) \neq 1$ abort
$t \xleftarrow{\$} \mathbb{Z}_p,\ K := \mathsf{H}(\text{context}, K_A, K_B, K_X)$
$v = (g_1^t, g_2^t), \pi$
if $\pi \neq \mathsf{F}(K_B, u, \sigma, v)$ abort
$\pi := \mathsf{F}(K_B, u, \sigma, v)$
$K := \mathsf{H}(\text{context}, K_A, K_B, K_X)$
$K := \mathsf{H}(\text{context}, K_X)$
$K := \mathsf{H}(\text{context}, K_X)$
$\mathsf{H}_A(g_1^t, g_2^t, (g_1^t)^{a_1}(g_2^t)^{a_2}) = K_A = \mathsf{H}_A(g_1^t, g_2^t, A^t)$
$\mathsf{H}_B(g_1^s, g_2^s, B^s) = K_B = \mathsf{H}_B(g_1^s, g_2^s, (g_1^s)^{b_1}(g_2^s)^{b_2})$
$\mathsf{H}_X(g_1^t, g_2^t, (g_1^t)^{x_1}(g_2^t)^{x_2}) = K_X = \mathsf{H}_X(g_1^t, g_2^t, X^t)$

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

Generic constructions of AKE from NCKE. Our main technical tool is a new approach to achieve a tight reduction for authenticated key exchange protocols. Our starting point is an extension of (receiver) non-committing encryption (NCE) [8,33] to *non-committing key encapsulation (NCKE) in the multi-user setting with corruptions.* We construct an NCKE scheme in the random oracle model from any smooth projective hash proof system (HPS) [11]. If the HPS' subset membership problem (SMP) is hard in the multi-instance setting, then the NCKE scheme is also tightly secure in our multi-user setting. We provide two such HPS, one from the DDH assumption, and another one from a subgroup assumption over groups of unknown order. The construction allows us to address the commitment problem described above.

We give a generic construction of an implicitly authenticated two-message AKE protocol $\mathsf{AKE}_{\mathsf{wFS}}$ with weak forward security from any NCKE scheme, whose security is tightly based on the multi-user security of the underlying NCKE scheme. Furthermore, we give a generic construction of an explicitly authenticated two-message AKE protocol $\mathsf{AKE}_{\mathsf{FS}}$ with perfect forward security by adding a tightly-secure signature scheme and a message authentication code (MAC) to our first construction, see Figure 3. Thus, we require only a single signature which is particularly useful for tightly-secure key exchange, because known constructions of suitable tightly-secure signature schemes [1,21] have relatively large signatures and replacing one signature with a MAC significantly improves the computational efficiency and communication complexity of the protocol.[5]

All these generic constructions leverage NCKE in order to resolve the technical difficulties in constructing tightly-secure AKE protocols described before.

[5] [31] showed how to generically avoid signatures in forward-secure AKE protocols, but at the cost of additional messages.

| **Protocol** | **Comm.** $(\mathbb{G}, \{0,1\}^\kappa, \mathsf{Sig})$ | **Bytes** | **#Msg.** | **Assump.** | **Auth.** | **Model** | **State Reveal** | **Sec. Loss** |
|---|---|---|---|---|---|---|---|---|
| Protocols with full forward security | | | | | | | | |
| $\mathsf{TLS}^*$ [16,15] | $(2,4,2)$ | 704 | 3 | Strong-DH + DDH | expl. | SBG | no | $O(1)$ |
| $\mathsf{GJ}$ [21] | $(2,1,2)$ | 608 | 3 | DDH | expl. | MBG | no | $O(1)$ |
| $\mathsf{LLGW}$ [32] | $(3,0,2)$ | 608 | 2 | DDH | expl. | MBG | no | $O(1)$ |
| $\mathsf{AKE}_{\mathsf{FS,DDH}}$ (Fig. 1) | $(5,1,1)$ | 448 | 2 | DDH | expl. | SBG | yes | $O(1)$ |
| Protocols with weak forward security | | | | | | | | |
| $\mathsf{HMQV}$ [26] | $(2,0,0)$ | 64 | 2 | CDH | impl. | SBG | yes | $O(TN^2\ell^2)$ |
| $\mathsf{CCGJJ}$ [10] | $(2,0,0)$ | 64 | 2 | Strong-DH | impl. | SBG | no | $O(N)$ |
| $\mathsf{CCGJJ}_{\mathrm{Twin}}$ [10] | $(3,0,0)$ | 96 | 2 | CDH | impl. | SBG | no | $O(N)$ |
| $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ (Fig. 1) | $(5,0,0)$ | 160 | 2 | DDH | impl. | SBG | yes | $O(1)$ |

**Fig. 2.** Comparison of AKE protocols over a group $\mathbb{G}$, where $N$ refers to the number of parties, $\ell$ to the number of sessions per party and $T$ is the number of test queries. $\mathsf{TLS}^*$ refers to the TLS 1.3 handshake, instantiated with the tightly-secure signatures of [21]. The column **Comm.** counts the communication complexity of the protocols in terms of the number of group elements, hashes, and signatures. The column **Model** lists the AKE security model and distinguishes between multi-bit guessing (MBG) and the single-bit-guessing (SBG) security.

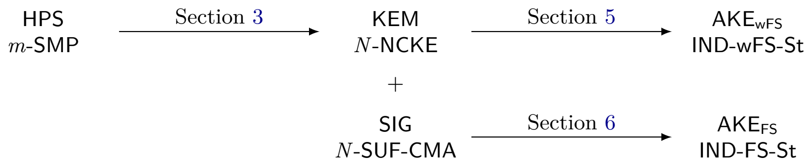

**Fig. 3.** Overview of our transformations, where $N$ is the maximum number of users in the $\mathsf{NCKE}$ security game and in the $\mathsf{SUF\text{-}CMA}$ security game. The subset membership problem of HPS is $m$-fold for $m = N \cdot q$, where $q$ is the maximum number of challenge queries in the $\mathsf{NCKE}$ security game.

Handling Ephemeral State Reveals. Our protocols are secure against ephemeral state reveals. We construct the first tightly-secure protocols to achieve this. Note that this requires us to deal with the situation that the reduction must “know” valid ephemeral states for *all* sessions, even tested sessions. To this end, we encrypt the state information with a symmetric long-term key. An adversary now needs to query both long-term secret key and ephemeral state to reveal the secret state information, similarly to the approach used in the NAXOS protocol [28]. While the idea of achieving security against ephemeral state reveals by relying on the security of long-term keys was used before [28,5,36,19], the approach to simply encrypt the state is new. It avoids the expensive re-computation of protocol messages required in prior generic approaches, which makes it particularly efficient. Also, previous work did not focus on tightness and it is unclear if a tight proof can be achieved in an even stronger security model which requires to reveal the randomness.

Our approach does not work generically, e.g., it cannot be applied to the protocols in [21,10], so we have to design our protocols such that they are compatible. This is due to the fact that in both works, the state is a secret DH exponent which is implicitly determined by rerandomizing the CDH (or DDH) challenge and then is embedded in multiple sessions. Thus, the reduction is able to extract the solution independently of which session is the test session, but it also does not know any of the secret exponents, which the adversary could reveal for non-test sessions.

### 1.3 Related Work and Open Problems

Concurrent and independent work of Liu et al. [32] also proposed a tightly secure 2-message AKE with full forward security. Compared to our protocols, they do not consider state reveal attacks and their proofs only hold in the MBG security model. Their AKE construction $\mathsf{LLGW}$ follows the well known $1\times\mathsf{KEM}+2\times\mathsf{SIG}$ approach, meaning that even neglecting the issues with the MBG security model, it is still considerably less efficient than ours (c.f. Fig. 2). The main novelty of [32] is the new KEM security notion of (multi-bit) "IND-mCPA with adaptive reveals" that gives them the handle to prove tight MBG security. It is a natural question whether this KEM security notion can be adapted to a single-bit notion such that the resulting AKE protocol achieves tight SBG (rather than MBG) security. This is in particular interesting since IND-mCPA KEMs with adaptive reveals can be instantiated in the standard model, whereas our $\mathsf{NCKE}$ notion seem to inherently rely on random oracles. More concretely this raises the question whether (variants of) [32] can also be proved in the SBG model, without relying on random oracles.

## 2 Preliminaries

For an integer $n$, $[n]$ denotes the set $\{1, ..., n\}$. For a set $S$, $s \xleftarrow{\$} S$ denotes that $s$ is sampled uniformly and independently at random from $S$. $y \leftarrow \mathcal{A}(x_1, x_2, ...)$ denotes that on input $x_1, x_2, ...$ the probabilistic algorithm $\mathcal{A}$ returns $y$. $\mathcal{A}^{\mathrm{O}}$ denotes that algorithm $\mathcal{A}$ has access to oracle O. We will use code-based games as introduced in [35]. An adversary is a probabilistic algorithm. $\Pr[G^{\mathcal{A}} \Rightarrow 1]$ denotes the probability that the final output $G^{\mathcal{A}}$ of game $G$ running adversary $\mathcal{A}$ is 1.

## 3 Multi-Receiver Non-Committing Key Encapsulation

In this section, we introduce Multi-Receiver Non-Committing Key Encapsulation (NCKE). We will use this concept to resolve the "commitment problem" described in the introduction, which often makes proofs for multi-party protocols with adaptive corruptions non-tight, as for example AKE protocols.

Syntax. A key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ consists of three algorithms. The key generation algorithm $\mathsf{Gen}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where $\mathsf{pk}$ is the public key and $\mathsf{sk}$ the secret key. The encapsulation algorithm inputs a public key $\mathsf{pk}$ and outputs a ciphertext $c$ and a key $K$ from the key space $\mathcal{K}$, where $c$ is called an encapsulation of $K$. The deterministic decapsulation algorithm inputs the secret key $\mathsf{sk}$ and a ciphertext $c$ and outputs $K$.

By $\mu$ we denote the *collision probability* of the key generation algorithm. In particular,

$$\Pr[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}, (\mathsf{pk}',\mathsf{sk}') \leftarrow \mathsf{Gen} : \mathsf{pk} = \mathsf{pk}'] \leq 2^{-\mu} \ .$$

We denote the *min-entropy* of the encapsulation algorithm $\mathsf{Encaps}$ by $\gamma(\mathsf{pk}) \coloneqq -\log \max_{c \in \mathcal{C}} \Pr[c = \mathsf{Encaps}(\mathsf{pk})]$. We say KEM is $\gamma$-spread if for all $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen} : \gamma(\mathsf{pk}) \geq \gamma$. This implies that for all $c \in \mathcal{C}$:

$$\Pr[c = \mathsf{Encaps}(\mathsf{pk})] \leq 2^{-\gamma} \ .$$

Security. Following [33], we introduce a security definition of Multi-Receiver Non-Committing Key Encapsulation (NCKE) for a key encapsulation mechanism $\mathsf{KEM}$ in the random oracle model, i. e., the $\mathsf{KEM}$ algorithms have access to a random oracle $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, indicated by $\mathsf{Encaps}^{\mathsf{H}}$. Our definition is relative to a simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$. The simulated key generation algorithm $\mathsf{SimGen}$ generates a key pair $(\mathsf{pk},\mathsf{sk})$. The simulated encapsulation algorithm $\mathsf{SimEncaps}$ takes both the public and private key and outputs a ciphertext $c$. The simulated hash algorithm $\mathsf{SimHash}$ inputs the key pair as well as three sets (used for bookkeeping) and deterministically computes a simulated hash value.

We define the two games $\mathsf{NCKE}_{\mathsf{real}}$ and $\mathsf{NCKE}_{\mathsf{sim}}$ in Figure 4 where we consider $N$ receivers each holding a key pair $(\mathsf{pk}_n,\mathsf{sk}_n)$. In the $\mathsf{NCKE}_{\mathsf{real}}$ game, the original $\mathsf{Encaps}$ algorithm is used. We give each user an individual hash function $\mathsf{H}_n$ such that keys are computed independently. (In general, this can be implemented by using the user's public key and identity as input to the hash function as well, where collisions have to be considered.) In the $\mathsf{NCKE}_{\mathsf{sim}}$ game, the $\mathsf{SimEncaps}$ algorithm is used to compute the ciphertexts. Keys are chosen uniformly at random. The adversary may also adaptively corrupt some receivers. We require that ciphertexts of corrupted receivers always decapsulate to the key output by Encaps, which is modeled by the $\mathsf{SimHash}$ algorithm. Therefore, if the receiver is corrupted, the algorithm takes sets $\mathcal{CK}$, $\mathcal{D}$ and $\mathcal{H}$, where the first one stores all challenge ciphertexts and keys output to the adversary, the second one stores all decapsulation queries and the third one stores all hash queries which have been issued so far. Thus, the $\mathsf{SimHash}$ algorithm can answer future queries based on everything that is known to the adversary. If the receiver is not corrupted, set $\mathcal{C}$ is used instead of $\mathcal{CK}$. This set stores only challenge ciphertexts and thus a hash value is computed independently of previous challenge keys.

The goal of an adversary $\mathcal{A}$ is to distinguish between the real KEM algorithms used in game $\mathsf{NCKE}_{\mathsf{real}}$ and the simulated algorithms used in game $\mathsf{NCKE}_{\mathsf{sim}}$. This is captured in Definition 1. Note that the non-committing property is due to the $\mathsf{SimHash}$ algorithm. In particular, the $\mathsf{SimHash}$ algorithm ensures that a (uniformly random) challenge key can be explained by the corresponding ciphertext generated by $\mathsf{SimEncaps}$ as soon as the receiver is corrupted.

```
$\mathsf{NCKE_{real}}$ and $\boxed{\mathsf{NCKE_{sim}}}$
for $n \in [N]$
    $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}$
    $\boxed{(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{SimGen}}$
    opened$[n] :=$ false
    $\mathcal{CK}_n := \varnothing,\ \mathcal{C}_n := \varnothing,\ \mathcal{D}_n := \varnothing,\ \mathcal{H}_n := \varnothing$
$b' \leftarrow \mathcal{A}^{\textsc{Encaps},\textsc{Decaps},\textsc{Open},\mathsf{H}_1,\ldots,\mathsf{H}_N}(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
return $b'$

$\mathsf{H}_n(M)$                                        // $n \in [N]$
if $\exists h$ s.t. $(M, h) \in \mathcal{H}_n$ return $h$
$h \xleftarrow{\$} \{0,1\}^\kappa$
if opened$[n]$
    $h \leftarrow \mathsf{SimHash}(\mathsf{pk}_n, \mathsf{sk}_n, \mathcal{CK}_n, \mathcal{D}_n, \mathcal{H}_n, M)$
else
    $h \leftarrow \mathsf{SimHash}(\mathsf{pk}_n, \mathsf{sk}_n, \mathcal{C}_n, \mathcal{D}_n, \mathcal{H}_n, M)$
$\mathcal{H}_n := \mathcal{H}_n \cup \{(M, h)\}$
return $h$

$\textsc{Encaps}(n \in [N])$
$(c, K) \leftarrow \mathsf{Encaps}^{\mathsf{H}_n}(\mathsf{pk}_n)$
$\boxed{c \leftarrow \mathsf{SimEncaps}(\mathsf{pk}_n, \mathsf{sk}_n)}$
$\boxed{K \xleftarrow{\$} \mathcal{K}}$
$\mathcal{CK}_n := \mathcal{CK}_n \cup \{(c, K)\}$
$\mathcal{C}_n := \mathcal{C}_n \cup \{(c, \perp)\}$
return $(c, K)$

$\textsc{Decaps}(n \in [N], c)$
if $\exists K$ s.t. $(c, K) \in \mathcal{CK}_n$
    return $\perp$
$K := \mathsf{Decaps}^{\mathsf{H}_n}(\mathsf{sk}_n, c)$
$\mathcal{D}_n := \mathcal{D}_n \cup \{c\}$
return $K$

$\textsc{Open}(n \in [N])$
opened$[n] :=$ true
return $\mathsf{sk}_n$
```

**Fig. 4.** Real and simulated game for $N$-receiver non-committing key encapsulation in the random oracle model.

**Definition 1 ($N$-Receiver Non-Committing Key Encapsulation).** *We define games* $\mathsf{NCKE_{real}}$ *and* $\mathsf{NCKE_{sim}}$ *as in Figure 4, where $N$ is the number of users. The simulator* $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$ *is defined relative to* $\mathsf{KEM}$ *and is used in* $\mathsf{NCKE_{sim}}$*. The advantage of an adversary $\mathcal{A}$ against* $\mathsf{KEM}$ *and* $\mathsf{Sim}$ *is defined as*

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{A}) := \left|\Pr[\mathsf{NCKE}^{\mathcal{A}}_{\mathsf{real}} \Rightarrow 1] - \Pr[\mathsf{NCKE}^{\mathcal{A}}_{\mathsf{sim}} \Rightarrow 1]\right| \ .$$

When we write $\mathsf{NCKE}$, we mean $\mathsf{NCKE\text{-}CCA}$, where the adversary is allowed to access a decapsulation oracle. Sometimes we will explicitly write $\mathsf{NCKE\text{-}CCA}$ to differentiate from $\mathsf{NCKE\text{-}CPA}$, where the adversary cannot issue decapsulation queries.

We stress that compared to the standard definition of non-committing encryption in the random oracle model (e.g., [33]), Definition 1 is for KEMs (rather than encryption), only considers receiver corruptions (rather than sender and receiver corruptions), and considers multiple receivers (rather than one single receiver).

Instantiations from Hash Proof Systems. We recall the definition of hash proof systems by Cramer and Shoup [11] and properties defined in [25].

Smooth Projective Hashing. Let $\mathcal{Y}$ and $\mathcal{Z}$ be sets and $\mathcal{X} \subset \mathcal{Y}$ a language. Let $\Lambda_{\mathsf{sk}} : \mathcal{Y} \to \mathcal{Z}$ be a hash function indexed with $\mathsf{sk} \in \mathcal{SK}$, where $\mathcal{SK}$ is a set. A hash function $\Lambda_{\mathsf{sk}}$ is projective if there exists a projection $\mu : \mathcal{SK} \to \mathcal{PK}$ such that $\mu(\mathsf{sk}) \in \mathcal{PK}$ defines the action of $\Lambda_{\mathsf{sk}}$ over $\mathcal{X}$. In particular, for every $c \in \mathcal{X}$, $Z = \Lambda_{\mathsf{sk}}(c)$ is uniquely determined by $\mu(\mathsf{sk})$ and $c$. However, there is no guarantee for $c \in \mathcal{Y} \setminus \mathcal{X}$ and it may not be possible to compute $\Lambda_{\mathsf{sk}}(c)$ from $\mu(\mathsf{sk})$ and $C$. A projective hash function is $k$-entropic if for all $c \in \mathcal{Y} \setminus \mathcal{X}$ it holds that $H_\infty(\Lambda_{\mathsf{sk}}(c) \mid \mathsf{pk}) \geq k$, where $\mathsf{pk} = \mu(\mathsf{sk})$ for $\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$.

```
$\mathsf{Gen}(\mathsf{par})$
$\mathsf{sk} \xleftarrow{\$} \mathcal{SK}$
$\mathsf{pk} := \mu(\mathsf{sk})$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Encaps}^{\mathsf{H}}(\mathsf{pk})$
$c \xleftarrow{\$} \mathcal{X}$ with witness $r$
$K := \mathsf{H}(c, \mathsf{Pub}(\mathsf{pk}, c, r))$
return $(c, K)$

$\mathsf{Decaps}^{\mathsf{H}}(\mathsf{sk}, c)$
$K := \mathsf{H}(c, \mathsf{Priv}(\mathsf{sk}, c))$
return $K$
```

**Fig. 5.** Key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$.

```
$\mathsf{SimEncaps}(\mathsf{pk}, \mathsf{sk})$
$c \xleftarrow{\$} \mathcal{Y} \setminus \mathcal{X}$
return $c$

$\mathsf{SimHash}(\mathsf{pk}, \mathsf{sk}, \mathcal{E}, \mathcal{D}, \mathcal{H}, M)$
$(c, Z) := M$
if $\exists K$ s.t. $(c, K) \in \mathcal{E}$ and $\mathsf{Priv}(\mathsf{sk}, c) = Z$
    $h := K$
else
    $h \xleftarrow{\$} \{0,1\}^{\kappa}$
return $h$
```

**Fig. 6.** Simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$ for $\mathsf{KEM}$, where $\mathsf{SimGen} = \mathsf{Gen}$. List $\mathcal{E}$ is either $\mathcal{CK}$ or $\mathcal{C}$.

Hash Proof System. A hash proof system $\mathsf{HPS} = (\mathsf{Par}, \mathsf{Priv}, \mathsf{Pub})$ consists of three algorithms. The randomized algorithm $\mathsf{Par}$ generates parametrized instances of $\mathsf{par} = (group, \mathcal{Z}, \mathcal{Y}, \mathcal{X}, \mathcal{PK}, \mathcal{SK}, \Lambda_{(\cdot)} : \mathcal{Y} \to \mathcal{Z}, \mu : \mathcal{SK} \to \mathcal{PK})$, where *group* may contain additional structural parameters. The deterministic public evaluation algorithm $\mathsf{Pub}$ inputs the projection key $\mathsf{pk} = \mu(\mathsf{sk})$, $c \in \mathcal{X}$ and a witness $r$ of the fact that $c \in \mathcal{X}$ and returns $Z = \Lambda_{\mathsf{sk}}(c)$. The deterministic private evaluation algorithm $\mathsf{Priv}$ takes $\mathsf{sk} \in \mathcal{SK}$ and returns $\Lambda_{\mathsf{sk}}(c)$ without knowing a witness. Furthermore, we assume that $\mu$ is efficiently computable and that there are efficient algorithms for sampling $c \in \mathcal{X}$ uniformly together with a witness $r$, sampling $c \in \mathcal{Y}$ uniformly and checking membership in $\mathcal{Y}$.

($m$-fold) Subset Membership Problem. We define the $m$-fold subset membership problem for HPS which requires to distinguish $m$ ciphertexts uniformly drawn from $\mathcal{X}$ from $m$ ciphertexts uniformly drawn from $\mathcal{Y} \setminus \mathcal{X}$. The advantage of an adversary $\mathcal{A}$ is defined as

$$\mathrm{Adv}_{\mathsf{HPS}}^{m-\mathsf{SM}}(\mathcal{A}) := |\Pr[\mathcal{A}(\mathcal{Y}, \mathcal{X}, c_1, ..., c_m) \Rightarrow 1] - \Pr[\mathcal{A}(\mathcal{Y}, \mathcal{X}, c'_1, ..., c'_m) \Rightarrow 1]| \quad ,$$

where $c_1, ..., c_m \xleftarrow{\$} \mathcal{X}$ and $c'_1, ..., c'_m \xleftarrow{\$} \mathcal{Y} \setminus \mathcal{X}$.

$N$-Receiver NCKE from HPS. We use a $k$-entropic hash proof system $\mathsf{HPS} = (\mathsf{Par}, \mathsf{Pub}, \mathsf{Priv})$ with $m$-fold subset membership problem and a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\kappa}$ in order to construct a key encapsulation algorithm $\mathsf{KEM}$ and a simulator $\mathsf{Sim}$ as shown in Figures 5 and 6. The encapsulation algorithm $\mathsf{Encaps}$ samples an element $c$ from $\mathcal{X}$ and a witness $r$. It runs the public evaluation algorithm and computes the key $K$ as $\mathsf{H}(c, \mathsf{Pub}(\mathsf{pk}, c, r))$. The decapsulation algorithm $\mathsf{Decaps}$ uses the result of the private evaluation algorithm $\mathsf{Priv}$ as input to $\mathsf{H}$ to compute $K$. Instead of sampling an element from $\mathcal{X}$, the $\mathsf{SimEncaps}$ algorithm samples an element $c$ uniformly at random from $\mathcal{Y} \setminus \mathcal{X}$ and only returns $c$. The $\mathsf{SimHash}$ algorithm takes as input three sets $\mathcal{E}, \mathcal{D}, \mathcal{H}$, where $\mathcal{E} \in \{\mathcal{C}, \mathcal{CK}\}$, and the value $M = (c, Z)$ chosen by the adversary. If there exists a key $K$ such that $(c, K) \in \mathcal{E}$ (note that for $\mathcal{E} = \mathcal{C}$ this will never be true) and the adversary's input to $\mathsf{H}$ satisfies $\mathsf{Priv}(\mathsf{sk}, c) = Z$, then the output value $h$ is set to $K$.

**Theorem 1 ($k$-entropic HPS with $(N \cdot q_E)$-fold SMP $\Rightarrow$ $N$-$\mathsf{NCKE}$).** *For any $N$-$\mathsf{NCKE}$ adversary $\mathcal{A}$ against $\mathsf{KEM}$ and $\mathsf{Sim}$ that issues at most $q_E$ queries to* Encaps*, $q_D$ queries to* Decaps *and at most $q_\mathsf{H}$ queries to each random oracle $\mathsf{H}_n$ for $n \in [N]$, there exists an adversary $\mathcal{B}$ against the $(N \cdot q_E)$-fold subset membership problem of $\mathsf{HPS}$ such that*

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{A}) \leq \mathrm{Adv}^{(N \cdot q_E)\text{-}\mathsf{SM}}_{\mathsf{HPS}}(\mathcal{B}) + \frac{N \cdot q_E \cdot q_\mathsf{H}}{2^k} + \frac{N \cdot q_E \cdot q_D}{|\mathcal{Y} \setminus \mathcal{X}|} \ ,$$

*where $\mathsf{HPS}$ is $k$-entropic, $\mathcal{Y}$ is the set of all ciphertexts and $\mathcal{X}$ is the set of valid ciphertexts.*

We will give an instantiation based on the $\mathsf{DDH}$ assumption in Section 7.1. For the proof of Theorem 1 and an instantiation based on the higher residuosity assumption, we refer to the full version [23].

## 4 Security Model for Two-Message Authenticated Key Exchange

A two-message key exchange protocol $\mathsf{AKE} = (\mathsf{Gen}_\mathsf{AKE}, \mathsf{Init}_\mathrm{I}, \mathsf{Der}_\mathsf{R}, \mathsf{Der}_\mathrm{I})$ consists of four algorithms which are executed interactively by two parties as shown in Figure 7. We denote the party which initiates the session by $\mathsf{P}_i$ and the party which responds to the session by $\mathsf{P}_r$. The key generation algorithm $\mathsf{Gen}_\mathsf{AKE}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ for one party. The initialization algorithm $\mathsf{Init}_\mathrm{I}$ inputs the initiator's long-term secret key $\mathsf{sk}_i$ and the responder's long-term public key $\mathsf{pk}_r$ and outputs a message $I$ and a state st. The responder's derivation algorithm $\mathsf{Der}_\mathsf{R}$ takes as input the responder's long-term secret key $\mathsf{sk}_r$, the initiator's long-term public key $\mathsf{pk}_i$ and a message $I$. It computes a message $R$ and a session key $K$. The initiator's derivation algorithm $\mathsf{Der}_\mathrm{I}$ inputs the initiator's long-term secret key $\mathsf{sk}_i$, the responder's long-term public key $\mathsf{pk}_r$, a message $R$ and a state st. It outputs a session key $K$. Note that in contrast to the initiating party $\mathsf{P}_i$, the responding party $\mathsf{P}_r$ will not be required to save any (secret) state information besides the session key $K$. The session key can be derived immediately after receiving the initiator's message.

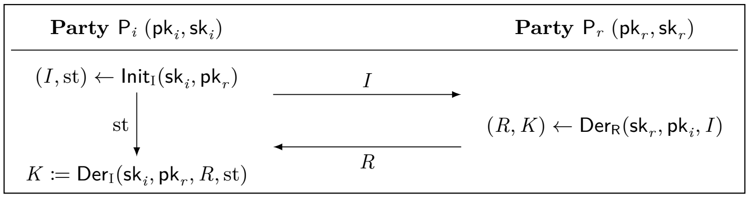
**Party** $\mathsf{P}_i$ $(\mathsf{pk}_i, \mathsf{sk}_i)$
**Party** $\mathsf{P}_r$ $(\mathsf{pk}_r, \mathsf{sk}_r)$
$(I, \mathrm{st}) \leftarrow \mathsf{Init}_\mathrm{I}(\mathsf{sk}_i, \mathsf{pk}_r)$
$I$
st
$(R, K) \leftarrow \mathsf{Der}_\mathsf{R}(\mathsf{sk}_r, \mathsf{pk}_i, I)$
$R$
$K \coloneqq \mathsf{Der}_\mathrm{I}(\mathsf{sk}_i, \mathsf{pk}_r, R, \mathrm{st})$

**Fig. 7.** Running a key exchange protocol between two parties.

Following [22], we define a game-based security model for authenticated key exchange using pseudocode. Our models for two different levels of security are

```
GAMES $\mathsf{IND\text{-}wFS\text{-}St}_b$ and $\boxed{\mathsf{IND\text{-}FS\text{-}St}_b}$
$\mathrm{cnt} := 0$                                  //session counter
$\mathcal{S} := \varnothing$                          //set of test sessions
for $n \in [N]$
    $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{AKE}}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}_1, \cdots, \mathsf{pk}_N)$
for $\mathrm{sID}^* \in \mathcal{S}$
    if $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$
        return 0                                 //session not fresh
    if $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
        return 0                                 //no valid attack
return $b'$

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

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

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

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, R)$
if $\mathrm{state}[\mathrm{sID}] = \bot$
    return $\bot$                                //not initialized
if $\mathrm{sKey}[\mathrm{sID}] \neq \bot$
    return $\bot$                                //no re-use
$(i, r) := (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
$\mathrm{st} := \mathrm{state}[\mathrm{sID}]$
$\mathrm{peerCorrupted}[\mathrm{sID}] := \mathrm{corrupted}[r]$
$K := \mathsf{Der}_{\mathrm{I}}(\mathsf{sk}_i, \mathsf{pk}_r, R, \mathrm{st})$
$(R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) := (R, K)$
return $\varepsilon$

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

$\textsc{Rev-State}(\mathrm{sID})$
if $\mathrm{type}[\mathrm{sID}] \neq$ "In" return $\bot$
$\mathrm{revState}[\mathrm{sID}] := \mathbf{true}$
return $\mathrm{state}[\mathrm{sID}]$

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

**Fig. 8.** Games $\mathsf{IND\text{-}wFS\text{-}St}_b$ and $\mathsf{IND\text{-}FS\text{-}St}_b$ for $\mathsf{AKE}$, where $b \in \{0, 1\}$. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}\}$. Helper procedures Fresh and Valid are defined in Figure 9. If there exists any test session which is neither fresh nor valid, the game will return 0.

given in Figure 8. We consider $N$ parties $\mathsf{P}_1, ..., \mathsf{P}_N$ with long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$, $n \in [N]$. Each session between two parties has a unique identification number sID and variables which are defined relative to sID:

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

To establish a session between two parties, the adversary is given access to oracles $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathsf{R}}$, where the first one starts a session of type "In"

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

$\underline{\textsc{Valid}(\mathrm{sID}^*)}$
$(i^*, r^*) := (\mathrm{init}[\mathrm{sID}^*], \mathrm{resp}[\mathrm{sID}^*])$
$\mathfrak{M}(\mathrm{sID}^*) := \{\mathrm{sID} \mid (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) = (i^*, r^*) \ \wedge \ (I[\mathrm{sID}], R[\mathrm{sID}]) = (I[\mathrm{sID}^*], R[\mathrm{sID}^*])$
                     $\wedge \ \mathrm{type}[\mathrm{sID}] \neq \mathrm{type}[\mathrm{sID}^*]\}$        //matching sessions
$\mathfrak{P}(\mathrm{sID}^*) := \{\mathrm{sID} \mid I[\mathrm{sID}] = I[\mathrm{sID}^*] \ \wedge \ \mathrm{type}[\mathrm{sID}] = \text{“In”} \ \wedge \ \mathrm{type}[\mathrm{sID}] \neq \mathrm{type}[\mathrm{sID}^*]\}$
                                          //partially matching sessions
for attack $\in$ Table 1 $\boxed{\text{Table 2}}$
    if attack = true return true
return false
```

**Fig. 9.** Helper procedures Fresh and Valid for games IND-wFS-St and IND-FS-St defined in Figure 8. Procedure Fresh checks if the adversary performed some trivial attack. In procedure Valid, each attack is evaluated by the set of variables shown in Table 1 (IND-wFS-St) or Table 2 (IND-FS-St) and checks if an allowed attack was performed. If the values of the variables are set as in the corresponding row, the attack was performed, i. e. attack = **true**, and thus the session is valid.

and the second one of type “Re”. Following [26,28], these oracles also take the intended peer's identity as input. In order to complete the initiator's session, the oracle $\textsc{Der}_{\mathrm{I}}$ has to be queried. Furthermore, the adversary has access to oracles Corrupt, Reveal and Rev-State to obtain secret information. As the responder can directly compute the key in a two-message protocol, we only require the initiator to store a state. The state contains information that is needed to compute the session key when the response is received, so it will consist of public and private information. We do not require to reveal the full randomness as in the eCK model [28]. A Rev-State query may be issued at any time. We use the following boolean values to keep track of which queries the adversary made:

- corrupted$[n]$ denotes whether the long-term secret key of party $\mathsf{P}_n$ was given to the adversary.
- revealed[sID] denotes whether the session key was given to the adversary.
- revState[sID] denotes whether the state information of that session was given to the adversary.
- peerCorrupted[sID] denotes whether the peer of the session was corrupted at the time the session key is computed, which is important for forward security.

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

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

- **Partially Matching Session**: A session $\mathrm{sID}'$ of type "In" is *partially matching* to session sID of type "Re" if the initial messages are the same ($I[\mathrm{sID}] = I[\mathrm{sID}']$).

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

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

*Attack Tables.* All attacks are defined using variables to indicate which queries the adversary may (not) make. We consider three dimensions covering all possible combinations of reveal queries the adversary can make:

- whether the test session is on the initiator's ($\mathrm{type}[\mathrm{sID}^*] =$"In") or the responder's side ($\mathrm{type}[\mathrm{sID}^*] =$"Re"),
- all combinations of long-term secret key and state reveals (corrupted and revState variables), also taking into account when a corruption happened (peerCorrupted),
- whether the adversary acted passively (matching session), partially active (partially matching session) or actively (no matching session).

This yields a full table of 24 attacks, in particular capturing *key compromise impersonation* (KCI) and *maximal exposure* (MEX) attacks. An attack was performed if the variables are set to the corresponding values in the table. However, when considering two-message protocols, where the responder's side does not have a state, and we only consider *weak forward security*, some of the attacks are redundant. Thus, we obtain *distilled* tables. We exclude trivial attacks, e.g., the generic attack on two-message AKE protocols with state-reveals described in [30]. Therefore, the adversary is not allowed to obtain the state of a partially matching session. Also note that by definition, a partially matching session for a two-message protocol can only be of type "Re". Table 1 is the distilled table used for the IND-wFS-St security game and Table 2 is used for the IND-FS-St security game. Note that the numbering of attacks in the distilled tables is inherited from the full table given in the full version [23].

| | $\mathcal{A}$ gets (Initiator, Responder) | corrupted[$i^*$] | corrupted[$r^*$] | type[sID$^*$] | revState[sID$^*$] | $\exists$sID $\in \mathfrak{M}$(sID$^*$) :<br>revState[sID] | $\lvert\mathfrak{M}$(sID$^*$)$\rvert$ | $\exists$sID $\in \mathfrak{P}$(sID$^*$) :<br>revState[sID] | $\lvert\mathfrak{P}$(sID$^*$)$\rvert$ |
|---|---|---|---|---|---|---|---|---|---|
| (0) | **multiple partially matching sessions** | – | – | – | – | – | – | – | $> 1$ |
| $(1 \vee 2)$ | **(long-term, long-term)** | – | – | – | **F** | **F** | 1 | – | – |
| $(7 \vee 8)$ | **(state, long-term)** | **F** | – | – | – | – | 1 | – | – |
| (10) | **(long-term, long-term)** | – | – | "Re" | **F** | n/a | 0 | **F** | 1 |
| (16) | **(state, long-term)** | **F** | – | "Re" | **F** | n/a | 0 | – | 1 |
| (19) | **(state, state)** | **F** | **F** | "In" | – | n/a | 0 | n/a | 0 |
| (21) | **(long-term, state)** | – | **F** | "In" | **F** | n/a | 0 | n/a | 0 |
| (24) | **(state, long-term)** | **F** | – | "Re" | **F** | n/a | 0 | n/a | 0 |

**Table 1.** Distilled table of attacks for wFS adversaries against two-message protocols. This table is obtained from the full table of attacks by using that responders do not have a state and that we are considering weak forward security. The numbering of attacks is inherited from the full table. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. **F** means "false" and "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

| | $\mathcal{A}$ gets (Initiator, Responder) | corrupted[$i^*$] | corrupted[$r^*$] | peerCorrupted[sID$^*$] | type[sID$^*$] | revState[sID$^*$] | $\exists$sID $\in \mathfrak{M}$(sID$^*$) :<br>revState[sID] | $\lvert\mathfrak{M}$(sID$^*$)$\rvert$ | $\exists$sID $\in \mathfrak{P}$(sID$^*$) :<br>revState[sID] | $\lvert\mathfrak{P}$(sID$^*$)$\rvert$ |
|---|---|---|---|---|---|---|---|---|---|---|
| (0) | **multiple partially matching sessions** | – | – | – | – | – | – | – | – | $> 1$ |
| $(1 \vee 2)$ | **(long-term, long-term)** | – | – | – | – | **F** | **F** | 1 | – | – |
| $(7 \vee 8)$ | **(state, long-term)** | **F** | – | – | – | – | – | 1 | – | – |
| (10) | **(long-term, long-term)** | – | – | **F** | "Re" | **F** | n/a | 0 | **F** | 1 |
| (16) | **(state, long-term)** | **F** | – | – | "Re" | **F** | n/a | 0 | – | 1 |
| (17) | **(long-term, long-term)** | – | – | **F** | "In" | **F** | n/a | 0 | n/a | 0 |
| (18) | **(long-term, long-term)** | – | – | **F** | "Re" | **F** | n/a | 0 | n/a | 0 |
| (23) | **(state, long-term)** | **F** | – | **F** | "In" | – | n/a | 0 | n/a | 0 |

**Table 2.** Distilled table of attacks for full FS adversaries against two-message protocols. This table is obtained from the full table of attacks by removing redundant rows and using that responders do not have a state. The numbering of attacks is inherited from the full table. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. **F** means "false" and "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

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

*Example.* If the test session is an initiating session (type[sID$^*$] ="In"), the state was not revealed (revState[sID$^*$] = **false**) and there is a matching session ($|\mathfrak{M}(\text{sID}^*)| = 1$), then row (1∨2) will evaluate to true. In this scenario, the adversary is allowed to query both long-term secret keys.

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

**Definition 2 (Key Indistinguishability of AKE).** *We define games* IND-wFS-St$_b$ *and* IND-FS-St$_b$ *for $b \in \{0,1\}$ as in Figures 8 and 9. The advantage of an adversary $\mathcal{A}$ against* AKE *in these games is defined as*

$$\text{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr[\mathsf{IND\text{-}wFS\text{-}St}^{\mathcal{A}}_1 \Rightarrow 1] - \Pr[\mathsf{IND\text{-}wFS\text{-}St}^{\mathcal{A}}_0 \Rightarrow 1]\right| \quad \textit{and}$$

$$\text{Adv}^{\mathsf{IND\text{-}FS\text{-}St}}_{\mathsf{AKE}}(\mathcal{A}) := \left|\Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_1 \Rightarrow 1] - \Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_0 \Rightarrow 1]\right| \quad .$$

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

### 4.1 Relation to other Definitions

In this section, we will refer to the most widely used security definitions for authenticated key exchange protocols. In the first place, these include the CK model [9] and the stronger definition used for the HMQV protocol (CK+) in [26], the eCK model [28] and the strengthened version of [14], the definitions given in [24] and [1] which are both extensions of the BR model [4], and the definition of IND-Å security in [22]. In [13,12], Cremers showed that the CK, CK+ und eCK model are incomparable. Thus, we will not do a formal comparison of security models, but only point out similarities and differences between our definition and the definitions listed above.

Party Corruption. We allow the adversary to corrupt a party which means that it will obtain that party's long-term secret key as in the eCK model and the models given in [24,1,22]. In contrast, a corrupt query in the CK and CK+ model will reveal all information in the memory of that party, i. e. long-term secrets and session-specific information.

State-Reveals. Our model only allows state-reveal queries on initiating sessions because the initiator has to wait for the response to compute the session key. Thus, the state contains all that information that is needed to derive the session key as soon as the responder's message is received. The responder can directly compute the session key and does not have to store other information. The eCK model explicitly defines the state as the randomness that is used in the protocol. In the CK model, it is not clear which information is included in the state, but it is left to be specified by the AKE protocol itself. Other models such as [24], its extension given in [1] and the one used in [10] do not allow state-reveals at all. Here, we want to emphasize that in particular all previous work on tight AKE does not consider state reveals and we are the first ones to address this problem.

(Weak) Forward Security. Following Krawczyk [26], we specify two levels of forward security. IND-wFS-St models weak forward security, whereas IND-FS-St models full forward security. The first one is intended for 2-message protocols with implicit authentication, as those cannot achieve full forward security [26]. The second one is intended for protocols with explicit authentication. With those definitions, we capture the same properties as the most common security models given in [9,26,28,24,1], where some of them only define either weak or full forward security depending on whether they consider implicitly or explicitly authenticated protocols.

Matching Sessions and Partnering. As most security models, ours use the concept of matching sessions to define a relation between two sessions. Following Cremer and Feltz [14], we additionally use the term of origin (or partially matching) sessions, which refers to a relaxation of the definition of matching sessions. The concept of origin sessions is used for full forward security, in particular we need this to handle the no-match attack described by Li and Schäge [30], where two sessions compute the same session key but do not have matching conversations. Recent works such as [21,10] take up the approach of origin sessions and oracle partnering based on session keys as additional requirement.

On registering corrupt keys. Some security models for AKE allow the adversary also to *register* adversarially-generated keys, this holds in particular for previous works considering tightly-secure key exchange [1,21,10]. Technically this makes the security model strictly stronger, as one can easily construct contrived protocols that are insecure with adversarially-registered keys, but secure without.

However, in the actual security proofs in [1,21,10], adversarially-registered keys are treated no differently than corrupted keys. We chose to keep model, security proofs and notation as simple as possible (it is already complex enough, anyway), and thus omitted this query. However, it is straightforward to extend our model with it, and the proofs need not to be changed. Whenever the adversary registers a new key, it would immediately be marked as "corrupted" (just like in [1,21,10]). Apart from that, no additional changes to the proofs are required, since the proofs deal with all corrupted keys in the same way, regardless of their distribution or whether they are generated by the experiment or an external entity. We also do not require a proof of knowledge of the corrensponding secret key for the registration, or a proof that the registered public key is valid in any sense.

## 5 AKE with Weak Forward Security

In this section, we show how to build an implicitly authenticated AKE protocol using the concept of non-committing key encapsulation.

In particular, from two key encapsulation mechanisms $\mathsf{KEM}_{\mathsf{CPA}} = (\mathsf{Gen}_{\mathsf{CPA}}, \mathsf{Encaps}_{\mathsf{CPA}}, \mathsf{Decaps}_{\mathsf{CPA}})$ and $\mathsf{KEM}_{\mathsf{CCA}} = (\mathsf{Gen}_{\mathsf{CCA}}, \mathsf{Encaps}_{\mathsf{CCA}}, \mathsf{Decaps}_{\mathsf{CCA}})$, we construct a two-message authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{wFS}} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathsf{R}}, \mathsf{Der}_{\mathrm{I}})$ as shown in Figures 10 and 11. W.l.o.g. $\mathsf{KEM}_{\mathsf{CPA}}$, $\mathsf{KEM}_{\mathsf{CCA}}$, $\mathsf{AKE}_{\mathsf{wFS}}$ have identical key space $\mathcal{K}$. Each party holds a long-term key pair $(\mathsf{pk}, \mathsf{sk})$ for $\mathsf{KEM}_{\mathsf{CCA}}$ and a symmetric key $k$ to encrypt the secret state information which has to be stored by the initiating party. State encryption protects against state attacks and is implemented using a symmetric encryption scheme defined as $\mathsf{E}_k(\mathrm{st}') := (IV, \mathsf{G}(k, IV) \oplus \mathrm{st}')$ for a random nonce $IV$. Here $\mathsf{G} : \{0,1\}^* \to \{0,1\}^d$ is a random oracle and $d$ is an integer denoting the maximum bit length of the unencrypted state $\mathrm{st}'$. The protocol uses an additional cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ to output the session key.

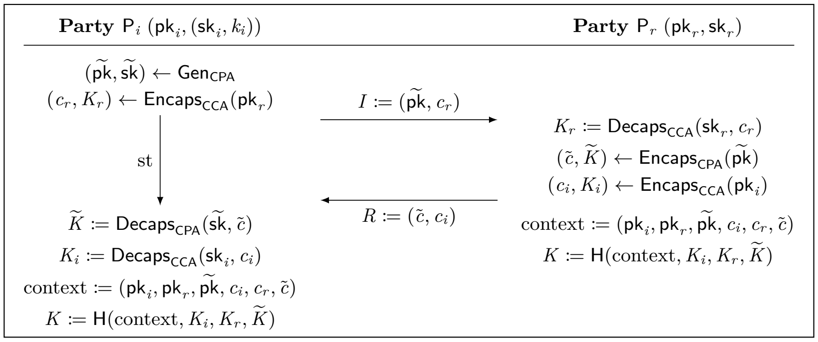
Party $\mathsf{P}_i$ $(\mathsf{pk}_i, (\mathsf{sk}_i, k_i))$
Party $\mathsf{P}_r$ $(\mathsf{pk}_r, \mathsf{sk}_r)$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$
$(c_r, K_r) \leftarrow \mathsf{Encaps}_{\mathsf{CCA}}(\mathsf{pk}_r)$
$I := (\widetilde{\mathsf{pk}}, c_r)$
st
$K_r := \mathsf{Decaps}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$
$(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$
$(c_i, K_i) \leftarrow \mathsf{Encaps}_{\mathsf{CCA}}(\mathsf{pk}_i)$
$R := (\tilde{c}, c_i)$
$\widetilde{K} := \mathsf{Decaps}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$
$K_i := \mathsf{Decaps}_{\mathsf{CCA}}(\mathsf{sk}_i, c_i)$
$\mathrm{context} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
$K := \mathsf{H}(\mathrm{context}, K_i, K_r, \widetilde{K})$
$\mathrm{context} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
$K := \mathsf{H}(\mathrm{context}, K_i, K_r, \widetilde{K})$

**Fig. 10.** Visualization: Running protocol $\mathsf{AKE}_{\mathsf{wFS}}$ between two parties.

The initiating party generates an ephemeral key pair for $\mathsf{KEM}_{\mathsf{CPA}}$, then runs the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the peer's public key to output a ciphertext $c_r$ and a key $K_r$ and sends the ephemeral public key and $c_r$ to the intended receiver. All values are stored temporarily and encrypted as described above, as they will later be needed to compute the session key. The responding party uses its secret key $\mathsf{sk}_r$ to compute key $K_r$ from $c_r$. Next, it runs the $\mathsf{Encaps}_{\mathsf{CPA}}$ algorithm on the received ephemeral public key to compute a ciphertext $\tilde{c}$ and a key $\widetilde{K}$ and then the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the initiator's public key to output $c_i$ and $K_i$. It sends both ciphertexts to the initiating party and computes the session key evaluating the hash function $\mathsf{H}$ on all public context and the three shared keys $K_r$, $K_i$ and $\widetilde{K}$. The initiator retrieves the secret state information and computes $K_i$ and $\widetilde{K}$ from $c_i$ and $\tilde{c}$. Now, it can also establish the session key.

**Theorem 2** ($\mathsf{KEM}_{\mathsf{CPA}}$ $\mathsf{NCKE\text{-}CPA}$ $+$ $\mathsf{KEM}_{\mathsf{CCA}}$ $\mathsf{NCKE\text{-}CCA}$ $\overset{\mathrm{ROM}}{\Rightarrow}$ $\mathsf{AKE}_{\mathsf{wFS}}$ $\mathsf{IND\text{-}wFS\text{-}St}$)**.** *For any* $\mathsf{IND\text{-}wFS\text{-}St}$ *adversary* $\mathcal{A}$ *against* $\mathsf{AKE}_{\mathsf{wFS}}$ *with* $N$ *parties*

```
$\underline{\mathsf{Gen}_{\mathsf{AKE}}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$
$k \xleftarrow{\$} \{0,1\}^{\kappa}$
return $(\mathsf{pk}', \mathsf{sk}') := (\mathsf{pk}, (\mathsf{sk}, k))$

$\underline{\mathsf{Init}_{\mathrm{I}}((\mathsf{sk}_i, k_i), \mathsf{pk}_r)}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$
$(c_r, K_r) \leftarrow \mathsf{Encaps}_{\mathsf{CCA}}(\mathsf{pk}_r)$
$IV \xleftarrow{\$} \{0,1\}^{\kappa}$
$\mathrm{st}' := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r)$
$\mathrm{st} := (IV, \mathsf{G}(k_i, IV) \oplus \mathrm{st}')$
return $((\widetilde{\mathsf{pk}}, c_r), \mathrm{st})$

$\underline{\mathsf{Der}_{\mathrm{R}}((\mathsf{sk}_r, k_r), \mathsf{pk}_i, (\widetilde{\mathsf{pk}}, c_r))}$
$K_r := \mathsf{Decaps}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$
$(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$
$(c_i, K_i) \leftarrow \mathsf{Encaps}_{\mathsf{CCA}}(\mathsf{pk}_i)$
$\mathrm{context} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
$K := \mathsf{H}(\mathrm{context}, K_i, K_r, \widetilde{K})$
return $((\tilde{c}, c_i), K)$

$\underline{\mathsf{Der}_{\mathrm{I}}((\mathsf{sk}_i, k_i), \mathsf{pk}_r, (\tilde{c}, c_i), \mathrm{st})}$
$(IV, \psi) := \mathrm{st}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r) := \mathsf{G}(k_i, IV) \oplus \psi$
$\widetilde{K} := \mathsf{Decaps}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$
$K_i := \mathsf{Decaps}_{\mathsf{CCA}}(\mathsf{sk}_i, c_i)$
$\mathrm{context} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
$K := \mathsf{H}(\mathrm{context}, K_i, K_r, \widetilde{K})$
return $K$
```

**Fig. 11.** Authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{wFS}}$ from $\mathsf{KEM}_{\mathsf{CPA}}$ and $\mathsf{KEM}_{\mathsf{CCA}}$. Lines written in purple color are only used to encrypt the state.

*that establishes at most $S$ sessions and issues at most $T$ queries to the test oracle* TEST*, $q_{\mathsf{G}}$ queries to random oracle $\mathsf{G}$ and at most $q_{\mathsf{H}}$ queries to random oracle $\mathsf{H}$, there exists an $N$-*NCKE-CCA *adversary $\mathcal{B}$ against $\mathsf{KEM}_{\mathsf{CCA}}$ and $\mathsf{Sim}_{\mathsf{CCA}}$ and an $S$-*NCKE-CPA *adversary $\mathcal{C}$ against $\mathsf{KEM}_{\mathsf{CPA}}$ and $\mathsf{Sim}_{\mathsf{CPA}}$ such that*

$$\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}_{\mathsf{wFS}}}(\mathcal{A}) \leq\ 2 \cdot \left(\mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}) + \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C})\right) + T \cdot \left(\frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right)$$
$$+ N^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{CCA}}}} + \frac{1}{2^{\kappa}}\right) + S^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{CPA}}}} + \frac{1}{2^{\gamma_{\mathsf{CCA}}}} + \frac{1}{2^{\gamma_{\mathsf{CPA}}}} + \frac{1}{2^{\kappa}}\right) + 2S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}}\ ,$$

*where $\mathsf{Sim}_{\mathsf{CCA}}$ and $\mathsf{Sim}_{\mathsf{CPA}}$ are the simulators from the* NCKE *experiments, $\mu_{\mathsf{CCA}}$ and $\mu_{\mathsf{CPA}}$ are the collision probability of the key generation algorithms $\mathsf{Gen}_{\mathsf{CCA}}$ and $\mathsf{Gen}_{\mathsf{CPA}}$, $\gamma_{\mathsf{CCA}}$ and $\gamma_{\mathsf{CPA}}$ are the spreadness parameters of the encapsulation algorithms $\mathsf{Encaps}_{\mathsf{CCA}}$ and $\mathsf{Encaps}_{\mathsf{CPA}}$ and $\kappa$ is a security parameter. The running times of $\mathcal{B}$ and $\mathcal{C}$ consist essentially of the time required to execute the security experiment with the adversary once, plus a minor number of additional operations (including bookkeeping, lookups etc.).*

*Proof (Sketch).* Let $\mathcal{A}$ be an adversary against IND-wFS-St security of $\mathsf{AKE}_{\mathsf{wFS}}$. For $b \in \{0,1\}$, game $G_{0,b}$ is the $\mathsf{IND\text{-}wFS\text{-}St}_b$ game, where we additionally exclude that collisions between long-term key pairs, ephemeral key pairs, ciphertexts and nonces occur.

In game $G_{1,b}$, we replace the computations for $\mathsf{KEM}_{\mathsf{CCA}}$ by the simulator $\mathsf{Sim}_{\mathsf{CCA}}$, which allows to draw keys $K_i$ and $K_r$ uniformly at random. This change affects all sessions which makes the proof tight. If the adversary reveals a long-term key pair of any user, the property of receiver non-committing key encapsulation ensures that the correct keys $K_i$ and $K_r$ can be computed by the adversary.

Next, we want to replace the computations for $\mathsf{KEM}_{\mathsf{CPA}}$ by the simulator $\mathsf{Sim}_{\mathsf{CPA}}$, which allows to draw keys $\widetilde{K}$ uniformly at random. However, the ephemeral secret key $\widetilde{\mathsf{sk}}$ is part of the state and will not be available to the

$\mathsf{NCKE\text{-}CPA}$ reduction in the first place. Thus, we introduce an intermediate game $G_{2,b}$ and do not compute the state when the session is initiated but only when the adversary queries the Rev-State oracle. In game $G_{3,b}$, we can then use the simulator for $\mathsf{KEM}_{\mathsf{CPA}}$ and draw keys $\widetilde{K}$ uniformly at random, whenever the ephemeral public key $\widetilde{\mathsf{pk}}$ comes from the experiment (i.e. the adversary creates a partially matching session). Again, the non-committing property of $\mathsf{KEM}_{\mathsf{CPA}}$ ensures consistency in case the adversary reveals both the state of a session and the long-term key of the initiator, which reveals the ephemeral secret key $\widetilde{\mathsf{sk}}$.

Depending on whether there exists a (partially) matching session and which queries to Rev-State and Corrupt the adversary makes, we can argue that at least one key $K_i$, $K_r$ or $\widetilde{K}$ in each test session is chosen uniformly at random and unknown to $\mathcal{A}$ and thus it cannot distinguish the session key from a uniformly random key in the last game $G_{4,b}$. □

The full proof of Theorem 2 can be found in the full version [23]. Note that the non-committing property is essential to embed random KEM keys in each session and thus to achieve tightness. This way, we only need to make a case distinction at the end and can argue that for all test sessions at least one KEM key is independent of the adversary's view no matter which queries it has made (provided it did not make a trivial attack). Relying on a weaker assumption requires to make a case distinction earlier in the proof and may involve guessing as in some cases it is not clear which KEM key will be revealed (through corruption and/or reveal or state reveal) at a later point in time.

# 6 AKE with Full Forward Security

We show how to build an explicitly authenticated AKE protocol using the concept of non-committing key encapsulation. As we also need a signature scheme, we will first give the relevant definitions.

## 6.1 Digital Signatures

A digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}_{\mathsf{SIG}}, \mathsf{Sign}, \mathsf{Vrfy})$ consists of three algorithms. The key generation algorithm $\mathsf{Gen}_{\mathsf{SIG}}$ outputs a key pair $(\mathsf{vk}, \mathsf{sigk})$, where $\mathsf{vk}$ is the verification key and $\mathsf{sigk}$ the signing key. The signing algorithm $\mathsf{Sign}$ inputs a signing key $\mathsf{sigk}$ and a message $m$ and outputs a signature $\sigma$. The deterministic verification algorithm $\mathsf{Vrfy}$ inputs the verification key $\mathsf{vk}$, a message $m$ and a signature $\sigma$ and outputs 1 if $\sigma$ is a valid signature for $m$, otherwise it outputs 0.

In Figure 12, we define the security game $N$ user <u>S</u>trong <u>U</u>n<u>F</u>orgeability under <u>C</u>hosen <u>M</u>essage <u>A</u>ttacks with corruptions ($N$-$\mathsf{SUF\text{-}CMA}$). The definition is similar to the one given in [1], except that we require *strong* unforgeability, i. e. the adversary may also find a new signature for a message it queried to the Sign oracle before. The advantage of an adversary $\mathcal{A}$ is defined as

$$\mathrm{Adv}_{\mathsf{SIG}}^{N\text{-}\mathsf{SUF\text{-}CMA}}(\mathcal{A}) := \Pr[N\text{-}\mathsf{SUF\text{-}CMA}^{\mathcal{A}} \Rightarrow 1] \ .$$

```
GAME N-SUF-CMA
$\mathcal{S}^{\mathsf{corr}} := \varnothing$
for $n \in [N]$
    $(\mathsf{vk}_n, \mathsf{sigk}_n) \leftarrow \mathsf{Gen}_{\mathsf{SIG}}$
    $\mathcal{S}_n := \varnothing$
$(n^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\text{Sign},\text{Corrupt}}(\mathsf{vk}_1, \cdots, \mathsf{vk}_N)$
if $\mathsf{Vrfy}(\mathsf{vk}_{n^*}, m^*, \sigma^*) = 1$ and $n^* \notin \mathcal{S}^{\mathsf{corr}}$
and $(m^*, \sigma^*) \notin \mathcal{S}_{n^*}$
    return 1
else
    return 0

Sign($n \in [N], m$)
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, m)$
$\mathcal{S}_n := \mathcal{S}_n \cup \{(m, \sigma)\}$
return $\sigma$

Corrupt($n \in [N]$)
$\mathcal{S}^{\mathsf{corr}} := \mathcal{S}^{\mathsf{corr}} \cup \{n\}$
return $\mathsf{sigk}_n$
```

**Fig. 12.** Game $N$-$\mathsf{SUF}$-$\mathsf{CMA}$ for $\mathsf{SIG}$.

### 6.2 Transformation using NCKE and a Signature Scheme

From two key encapsulation mechanisms $\mathsf{KEM}_{\mathsf{CPA}} = (\mathsf{Gen}_{\mathsf{CPA}}, \mathsf{Encaps}_{\mathsf{CPA}}, \mathsf{Decaps}_{\mathsf{CPA}})$ and $\mathsf{KEM}_{\mathsf{CCA}} = (\mathsf{Gen}_{\mathsf{CCA}}, \mathsf{Encaps}_{\mathsf{CCA}}, \mathsf{Decaps}_{\mathsf{CCA}})$ with key space $\mathcal{K}$ and a digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}_{\mathsf{SIG}}, \mathsf{Sign}, \mathsf{Vrfy})$, we construct a two-message authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{FS}} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{R}}, \mathsf{Der}_{\mathrm{I}})$ with key space $\mathcal{K}$ as shown in Figures 13 and 14. Each party has a key pair $(\mathsf{vk}, \mathsf{sigk})$ for $\mathsf{SIG}$, a key pair $(\mathsf{pk}, \mathsf{sk})$ for $\mathsf{KEM}_{\mathsf{CCA}}$ and a symmetric key $k$ to encrypt the secret state information which has to be stored by the initiating party (cf. Section 5). The protocol uses additional cryptographic hash functions $\mathsf{F} : \{0,1\}^* \to \{0,1\}^\kappa$ to compute value $\pi$ and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ to output the session key.

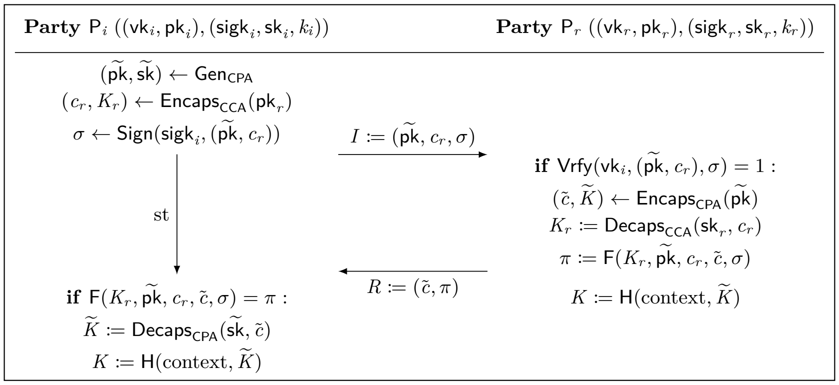
Party $\mathsf{P}_i$ $((\mathsf{vk}_i, \mathsf{pk}_i), (\mathsf{sigk}_i, \mathsf{sk}_i, k_i))$
Party $\mathsf{P}_r$ $((\mathsf{vk}_r, \mathsf{pk}_r), (\mathsf{sigk}_r, \mathsf{sk}_r, k_r))$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$
$(c_r, K_r) \leftarrow \mathsf{Encaps}_{\mathsf{CCA}}(\mathsf{pk}_r)$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_i, (\widetilde{\mathsf{pk}}, c_r))$
$I := (\widetilde{\mathsf{pk}}, c_r, \sigma)$
if $\mathsf{Vrfy}(\mathsf{vk}_i, (\widetilde{\mathsf{pk}}, c_r), \sigma) = 1$ :
$(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$
$K_r := \mathsf{Decaps}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$
$\pi := \mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma)$
$K := \mathsf{H}(\text{context}, \widetilde{K})$
st
$R := (\tilde{c}, \pi)$
if $\mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma) = \pi$ :
$\widetilde{K} := \mathsf{Decaps}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$
$K := \mathsf{H}(\text{context}, \widetilde{K})$

**Fig. 13.** Visualization: Running $\mathsf{AKE}_{\mathsf{FS}}$ between two parties, where $K$ is the resulting session key and context $:= (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$

The initiating party computes an ephemeral key pair for $\mathsf{KEM}_{\mathsf{CPA}}$, runs the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the intended receiver's public key $\mathsf{pk}_r$ to obtain a ciphertext $c_r$ and a key $K_r$ and signs both the ephemeral public key and $c_r$, which are sent to the receiver along with the signature. The receiver verifies the signature and then runs the $\mathsf{Encaps}_{\mathsf{CPA}}$ algorithm on the ephemeral public key to output a ciphertext $\tilde{c}$ and a key $\widetilde{K}$. It computes $K_r$ using its secret key $\mathsf{sk}_r$. It then

```
$\underline{\mathsf{Gen}_{\mathsf{AKE}}}$
$(\mathsf{vk}, \mathsf{sigk}) \leftarrow \mathsf{Gen}_{\mathsf{SIG}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$
$k \xleftarrow{\$} \{0,1\}^{\kappa}$
$\textbf{return } (\mathsf{pk}', \mathsf{sk}') := ((\mathsf{vk}, \mathsf{pk}), (\mathsf{sigk}, \mathsf{sk}, k))$

$\underline{\mathsf{Init}_{\mathrm{I}}((\mathsf{sigk}_i, \mathsf{sk}_i, k_i), (\mathsf{vk}_r, \mathsf{pk}_r))}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$
$(c_r, K_r) \leftarrow \mathsf{Encaps}_{\mathsf{CCA}}(\mathsf{pk}_r)$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_i, (\widetilde{\mathsf{pk}}, c_r))$
$IV \xleftarrow{\$} \{0,1\}^{\kappa}$
$\mathrm{st}' := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r, \sigma)$
$\mathrm{st} := (IV, \mathsf{G}(k_i, IV) \oplus \mathrm{st}')$
$I := (\widetilde{\mathsf{pk}}, c_r, \sigma)$
$\textbf{return } (I, \mathrm{st})$

$\underline{\mathsf{Der}_{\mathrm{R}}((\mathsf{sigk}_r, \mathsf{sk}_r, k_r), (\mathsf{vk}_i, \mathsf{pk}_i), (\widetilde{\mathsf{pk}}, c_r, \sigma))}$
$\textbf{if } \mathsf{Vrfy}(\mathsf{vk}_i, (\widetilde{\mathsf{pk}}, c_r), \sigma) \neq 1$
    $\textbf{return } \bot$
$(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$
$K_r := \mathsf{Decaps}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$
$\pi := \mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma)$
$\mathrm{context} := (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$
$K := \mathsf{H}(\mathrm{context}, \widetilde{K})$
$R := (\tilde{c}, \pi)$
$\textbf{return } (R, K)$

$\underline{\mathsf{Der}_{\mathrm{I}}((\mathsf{sigk}_i, \mathsf{sk}_i, k_i), (\mathsf{vk}_r, \mathsf{pk}_r), (\tilde{c}, \pi), \mathrm{st})}$
$(IV, \psi) := \mathrm{st}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r, \sigma) := \mathsf{G}(k_i, IV) \oplus \psi$
$\textbf{if } \mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma) \neq \pi$
    $\textbf{return } \bot$
$\widetilde{K} := \mathsf{Decaps}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$
$\mathrm{context} := (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$
$K := \mathsf{H}(\mathrm{context}, \widetilde{K})$
$\textbf{return } K$
```

**Fig. 14.** Authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{FS}}$ from $\mathsf{KEM}_{\mathsf{CPA}}$, $\mathsf{KEM}_{\mathsf{CCA}}$ and $\mathsf{SIG}$. Lines written in purple color are only used to encrypt the state.

tags the received message together with $\tilde{c}$ and $K_r$ by evaluating hash function $\mathsf{F}$ and sends the output together with $\tilde{c}$ to the initiator. The initiator retrieves $K_r$ from the secret state and also evaluates $\mathsf{F}$. If the output is the same, it computes $\widetilde{K}$ using the ephemeral secret key. The session key is computed evaluating hash function $\mathsf{H}$ on all public context and key $\widetilde{K}$. We establish the following theorem and give a proof sketch. The full proof can be found in the full version [23].

**Theorem 3** **($\mathsf{KEM}_{\mathsf{CPA}}$ NCKE-CPA+$\mathsf{KEM}_{\mathsf{CCA}}$ NCKE-CCA+SIG $N$-SUF-CMA $\overset{\mathrm{ROM}}{\Rightarrow}$ $\mathsf{AKE}_{\mathsf{FS}}$ IND-FS-St).** *For any* IND-FS-St *adversary $\mathcal{A}$ against* $\mathsf{AKE}_{\mathsf{FS}}$ *with $N$ parties that establishes at most $S$ sessions and issues at most $T$ queries to test oracle* Test*, at most $q_{\mathsf{H}}$, $q_{\mathsf{G}}$ and $q_{\mathsf{F}}$ queries to random oracles* $\mathsf{H}$, $\mathsf{G}$ *and* $\mathsf{F}$*, there exists an $N$-*SUF-CMA *adversary $\mathcal{B}$ against* SIG*, an $S$-*NCKE-CPA *adversary $\mathcal{C}$ against* $\mathsf{KEM}_{\mathsf{CPA}}$ *and* $\mathsf{Sim}_{\mathsf{CPA}}$ *and an $N$-*NCKE-CCA *adversary $\mathcal{D}$ against* $\mathsf{KEM}_{\mathsf{CCA}}$ *and* $\mathsf{Sim}_{\mathsf{CCA}}$ *such that*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}FS\text{-}St}}_{\mathsf{AKE}_{\mathsf{FS}}}(\mathcal{A}) \leq 2 \cdot \Big(&\mathrm{Adv}^{N\text{-}\mathsf{SUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{B}) + \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}) + \mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{D})\Big)\\ &+ T \cdot \left(\frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right) + N^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{SIG}}}} + \frac{1}{2^{\mu_{\mathsf{CCA}}}} + \frac{1}{2^{\kappa}}\right)\\ &+ S^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{CPA}}}} + \frac{1}{2^{\gamma_{\mathsf{CCA}}}} + \frac{1}{2^{\gamma_{\mathsf{CPA}}}} + \frac{1}{2^{\kappa}}\right) + 2S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} \quad ,\end{aligned}$$

*where* $\mathsf{Sim}_{\mathsf{CPA}}$ *and* $\mathsf{Sim}_{\mathsf{CCA}}$ *are the simulators from the* NCKE-CPA *and* NCKE-CCA *experiment, $\mu_{\mathsf{SIG}}$, $\mu_{\mathsf{CPA}}$, $\mu_{\mathsf{CCA}}$ are collision probabilities of the key generation algorithms* $\mathsf{Gen}_{\mathsf{SIG}}$, $\mathsf{Gen}_{\mathsf{CPA}}$ *and* $\mathsf{Gen}_{\mathsf{CCA}}$ *and $\gamma_{\mathsf{CPA}}$, $\gamma_{\mathsf{CCA}}$ are the spreadness parameters of the encapsulation algorithms. The running times of $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ consist essentially of the time required to execute the security experiment with the adver-*

*sary once, plus a minor number of additional operations (including bookkeeping, lookups etc.).*

*Proof (Sketch).* Let $\mathcal{A}$ be an adversary against $\mathsf{IND\text{-}FS\text{-}St}$ security of $\mathsf{AKE}_{\mathsf{FS}}$. For $b \in \{0,1\}$, $G_{0,b}$ is the $\mathsf{IND\text{-}FS\text{-}St}_b$ game, where we exclude that collisions between long-term key pairs, ephemeral key pairs, ciphertexts and nonces occur.

In game $G_{1,b}$, we abort when $\mathcal{A}$ computes a valid signature for an uncorrupted user that was not output by the experiment, reducing to $N$-$\mathsf{SUF\text{-}CMA}$ security of the signature scheme.

In game $G_{2,b}$, we replace the computations for $\mathsf{KEM}_{\mathsf{CCA}}$ by the simulator $\mathsf{Sim}_{\mathsf{CCA}}$ in all sessions using the non-committing property of $\mathsf{KEM}_{\mathsf{CCA}}$, which allows to draw key $K_r$ which serves as key for the MAC uniformly at random. Thus, the adversary cannot compute a valid MAC for an uncorrupted user.

In game $G_{3,b}$ (as in the proof of Theorem 2), we do not compute the state when the session is initiated but only when the adversary queries the Rev-State oracle. After that, we can switch $\widetilde{\mathsf{KEM}_{\mathsf{CPA}}}$ to the corresponding simulator $\mathsf{Sim}_{\mathsf{CPA}}$ in game $G_{4,b}$ and draw keys $\widetilde{K}$ uniformly at random, whenever the ephemeral public key $\widetilde{\mathsf{pk}}$ comes from the experiment (i.e. the adversary creates a partially matching session). As the adversary can only complete a (partially) matching sessions (otherwise it would have forged a signature or MAC), we can argue that $\widetilde{K}$ in each test session is chosen uniformly at random and unknown to $\mathcal{A}$ and thus he cannot distinguish the session key from a uniformly random key in the last game $G_{5,b}$. □

## 7 Concrete Instantiation of AKE Protocols

### 7.1 NCKE from the DDH Assumption

Let us first describe the hash proof system we will use. Therefore, let $\mathsf{GGen}$ be a group generation algorithm which takes the security parameter $1^\kappa$ as input and returns $(\mathbb{G}, p, g_1)$, where $g_1$ is a generator of the cyclic group $\mathbb{G}$ with prime order $p$. Define *group* $= (\mathbb{G}, p, g_1, g_2)$, where $g_2 = g_1^w$ for $w \xleftarrow{\$} \mathbb{Z}_p$. Define $\mathcal{Y} = \mathbb{Z}_p^2$ and $\mathcal{X} = \{(g_1^r, g_2^r) : r \in \mathbb{Z}_p\}$. A value $r$ is a witness that $(c_1, c_2) \in \mathcal{X}$. Define $\mathcal{SK} = \mathbb{Z}_p^2$, $\mathcal{PK} = \mathbb{Z}_p$ and $\mathcal{Z} = \mathbb{Z}_p$. For $\mathsf{sk} = (x_1, x_2) \in \mathbb{Z}_p^2$, define $\mu(\mathsf{sk}) = X = g_1^{x_1} g_2^{x_2}$. This defines the output of the parameter generation algorithm $\mathsf{Par}$.

For $(c_1, c_2) \in \mathcal{Y}$ define $\Lambda_{\mathsf{sk}}(c_1, c_2) := Z = (c_1^{x_1} c_2^{x_2})$. This defines the private evaluation algorithm $\mathsf{Priv}(\mathsf{sk}, (c_1, c_2))$. Given $\mathsf{pk} = \mu(\mathsf{sk}) = X$, $(c_1, c_2) \in \mathcal{X}$ and a witness $r \in \mathbb{Z}_p$ such that $(c_1, c_2) = (g_1^r, g_2^r)$, the public evaluation algorithm $\mathsf{Pub}(\mathsf{pk}, (c_1, c_2), r)$ computes $Z = \Lambda_{\mathsf{sk}}(c_1, c_2)$ as $Z = X^r$.

We define $\mathsf{KEM}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Encaps}_{\mathsf{DDH}}, \mathsf{Decaps}_{\mathsf{DDH}})$ with global parameters $\mathsf{par} := (\mathbb{G}, p, g_1, g_2)$ as shown in Figure 15.

**Definition 3 ($m$-fold DDH Problem).** *Let* $\mathsf{GGen}$ *be a PPT algorithm that on input* $1^\kappa$ *outputs a cyclic group* $\mathbb{G}$ *of prime order* $2^{k-1} \leq p \leq 2^k$ *with generator*

```
$\mathsf{Gen}_{\mathsf{DDH}}(\mathsf{par})$
$(x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2$
$X := g_1^{x_1} g_2^{x_2}$
return $(\mathsf{pk} := X,$
        $\mathsf{sk} := (x_1, x_2))$
```

```
$\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{DDH}}(\mathsf{pk}, m)$
$r \xleftarrow{\$} \mathbb{Z}_p$
$(c_1, c_2) := (g_1^r, g_2^r)$
$K := \mathsf{H}(c_1, c_2, X^r)$
return $((c_1, c_2), K)$
```

```
$\mathsf{Decaps}^{\mathsf{H}}_{\mathsf{DDH}}(\mathsf{sk}, (c_1, c_2))$
$K := \mathsf{H}(c_1, c_2, c_1^{x_1} c_2^{x_2})$
return $K$
```

**Fig. 15.** Key encapsulation mechanism $\mathsf{KEM}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Encaps}_{\mathsf{DDH}}, \mathsf{Decaps}_{\mathsf{DDH}})$.

*$g_1$. Furthermore let $g_2 = g_1^{\omega}$ for $\omega \xleftarrow{\$} \mathbb{Z}_p$. The $m$-DDH problem requires to distinguish $m$ DDH tuples from $m$ uniformly random tuples:*

$$\mathrm{Adv}^{m\text{-}\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{A}) := \Big|\Pr[\mathcal{A}(\mathbb{G}, p, g_1, g_2, (g_1^{r_i}, g_2^{r_i})_{i\in[m]}) \Rightarrow 1] \\ - \Pr[\mathcal{A}(\mathbb{G}, p, g_1, g_2, (g_1^{r_i}, g_2^{r_i'})_{i\in[m]}) \Rightarrow 1]\Big| \quad ,$$

*where probability is taken over $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}$, $r_i, r_i' \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [m]$, as well as the coin tosses of $\mathcal{A}$.*

**Lemma 1 (Random self-reducibility of DDH [17]).** *For any adversary $\mathcal{C}$ against the $m$-fold DDH problem, there exists an adversary $\mathcal{B}$ against the DDH problem with roughly the same running time such that*

$$\mathrm{Adv}^{m\text{-}\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{C}) \le \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + \frac{1}{p-1} \quad .$$

The following theorem establishes that the construction given in Figure 15 is an $N$-receiver non-committing encapsulation mechanism under the DDH assumption.

**Theorem 4.** *Under the DDH assumption and in the random oracle model, $\mathsf{KEM}_{\mathsf{DDH}}$ is an $N$-receiver non-committing key encapsulation mechanism. In particular, for any $N$-NCKE-CCA adversary $\mathcal{A}$ against $\mathsf{KEM}_{\mathsf{DDH}}$ and $\mathsf{Sim}_{\mathsf{DDH}}$ that issues at most $q_E$ queries per user to* Encaps, *$q_D$ queries to* Decaps *and at most $q_{\mathsf{H}}$ queries to each random oracle $\mathsf{H}_n$, $n \in [N]$, there exists an adversary $\mathcal{B}$ against DDH with roughly the same running time such that*

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{DDH}},\mathsf{Sim}_{\mathsf{DDH}}}(\mathcal{A}) \le \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + \frac{N \cdot q_E \cdot (q_{\mathsf{H}} + q_D + 1)}{p} + \frac{1}{p-1} \quad ,$$

*where $\mathsf{Sim}_{\mathsf{DDH}}$ is the simulator defined relative to $\mathsf{KEM}_{\mathsf{DDH}}$.*

*Proof.* We apply Theorem 1 and analyze the entropy of the underlying HPS. The key space $\mathcal{Z}$ is $\mathbb{Z}_p$. For $\mathsf{sk} = (x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathsf{pk} = \mu(\mathsf{sk}) = g_1^{x_1} g_2^{x_2}$ and $Z = \mathsf{Priv}(\mathsf{sk}, (c_1, c_2)) = c_1^{x_1} c_2^{x_2}$, where $(c_1, c_2) = (g_1^r, g_2^{r'})$ and $(r, r') \xleftarrow{\$} \mathbb{Z}_p^2$, we have

$$\begin{pmatrix} \log_{g_1} \mathsf{pk} \\ \log_{g_1} Z \end{pmatrix} = M \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \text{ where } M = \begin{pmatrix} 1 & w \\ r & wr' \end{pmatrix} \quad .$$

If $r \neq r'$, then $\det M = w(r' - r) \neq 0$, which implies that $\mathsf{pk}$ and $Z$ are random and independent group elements as long as $x_1, x_2$ are unknown. Thus, for all

$Z' \in \mathcal{Z}$, holds that $\Pr[Z = Z'] = 1/p$ . In Definition 3, all values $r_i$ and $r'_i$ are drawn uniformly at random from $\mathbb{Z}_p$. The probability that $r_i = r'_i$ for any $i \in [N \cdot q_E]$ is upper bounded by $N \cdot q_E/p$. Furthermore, the probability that a specific challenge ciphertext is issued to Decaps before it is output by Encaps is at most $q_D/p$. It follows that

$$\mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM},\mathsf{Sim}}(\mathcal{A}) \le \mathrm{Adv}^{m\text{-}\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + \frac{N \cdot q_E}{p} + \frac{N \cdot q_E \cdot q_{\mathsf{H}}}{p} + \frac{N \cdot q_E \cdot q_D}{p} \ .$$

Now Theorem 4 follows directly from Lemma 1. □

### 7.2 Concrete Instantiation of AKE Protocols

We instantiate protocols $\mathsf{AKE}_{\mathsf{wFS}}$ (Section 5) and $\mathsf{AKE}_{\mathsf{FS}}$ (Section 6.2) with $\mathsf{KEM}_{\mathsf{DDH}}$ (Section 7.1) for both $\mathsf{KEM}_{\mathsf{CPA}}$ and $\mathsf{KEM}_{\mathsf{CCA}}$. We will not give a concrete instantiation of the signature scheme used in $\mathsf{AKE}_{\mathsf{FS}}$ at this point. The resulting protocols $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ and $\mathsf{AKE}_{\mathsf{FS,DDH}}$ are shown in Figure 1 in the introduction.

Note that for $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ we can improve efficiency by sending only one ciphertext for both $\widetilde{\mathsf{pk}}$ and $\mathsf{pk}_i$ in the second message, as $\mathsf{KEM}_{\mathsf{DDH}}$ is a multi-recipient KEM. We establish Theorem 5 and give a proof sketch.

**Theorem 5 ($\mathsf{IND\text{-}wFS\text{-}St}$ security of $\mathsf{AKE}_{\mathsf{wFS,DDH}}$).** *Under the* $\mathsf{DDH}$ *assumption,* $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ *is* $\mathsf{IND\text{-}wFS\text{-}St}$ *secure in the random oracle model. In particular, for any* $\mathsf{IND\text{-}wFS\text{-}St}$ *adversary* $\mathcal{A}$ *against* $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ *with* $N$ *parties that establishes at most* $S$ *sessions and issues at most* $T$ *queries to the test oracle* Test, $q_{\mathsf{G}}$ *queries to random oracle* $\mathsf{G}$, $q_{\widetilde{\mathsf{H}}}$, $q_{\mathsf{H}_n}$ *queries to each random oracle* $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ *and* $\mathsf{H}_n$ *and at most* $q_{\mathsf{H}}$ *queries to random oracle* $\mathsf{H}$, *there exists an adversary* $\mathcal{B}$ *against* $\mathsf{DDH}$ *with roughly the same running time such that*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}wFS\text{-}St}}_{\mathsf{AKE}_{\mathsf{wFS,DDH}}}(\mathcal{A}) \le\ & 2 \cdot \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + T \cdot \frac{q_{\mathsf{G}} + q_{\mathsf{H}}}{2^{\kappa}} + (N+S)^2 \cdot \frac{1}{p} + N^2 \cdot \frac{1}{2^{\kappa}} \\ & + S^2 \cdot \left(\frac{2}{p} + \frac{1}{2^{\kappa}}\right) + 2S \cdot \left(\frac{q_{\mathsf{G}}}{2^{2\kappa}} + \frac{q_{\widetilde{\mathsf{H}}} + q_{\mathsf{H}_n} + 1}{p}\right) + \frac{2}{p-1} \ ,\end{aligned}$$

*where* $\kappa$ *is a security parameter.*

Due to the improved construction, we cannot apply Theorem 2 directly, but we give a proof sketch from the $\mathsf{DDH}$ assumption and show that the same technique as in the proofs of Theorems 2 and 4 can be used.

*Proof.* We proceed similar and consider collisions first. We assume that all key pairs generated by $\mathsf{Gen}_{\mathsf{DDH}}$ are different. Note that we also have to consider collisions between long-term and ephemeral public keys. It holds that

$$\Pr[x_1, x_2, x'_1, x'_2 \xleftarrow{\$} \mathbb{Z}_p : g_1^{x_1} g_2^{x_2} = g_1^{x'_1} g_2^{x'_2}] = 1/p \ .$$

Union bound yields $(N+S)^2/p$, as we have $N$ long-term public keys and at most $S$ ephemeral public keys. For ciphertexts $(c_1, c_2) \in \mathcal{C}$ output by the encapsulation algorithm $\mathsf{Encaps}_{\mathsf{DDH}}$, it holds that $\Pr[r \xleftarrow{\$} \mathbb{Z}_p : (c_1, c_2) = (g_1^r, g_2^r)] = 1/p$, which yields an upper bound for collisions of $S^2/p$, as there are at most $S$ sessions with one ciphertext. We also assume that values $IV$ are different in all sessions and keys $k_n$ are different for all parties.

We use the secret keys to compute keys $K_i$, $K_r$ and $\widetilde{K}$. Next, we replace all ciphertexts by uniformly random group elements at the same time, reducing to the $S$-fold $\mathsf{DDH}$ assumption and use the random self-reducibility property. In addition to that, we ensure that all ciphertexts are indeed invalid by adding $S/p$ which is the probability that exponents are the same for any ciphertext.

Instead of the corresponding random oracles, we use internal hash functions $\widetilde{\mathsf{H}}'_{\mathrm{sID}}$ and $\mathsf{H}'_n$ for sID $\in [S]$ and $n \in [N]$ to compute keys $K_i$, $K_r$ and $\widetilde{K}$, but patch the random oracles if the secret key is known to the adversary. As there are at most $S$ challenge keys computed with a long-term key pair and at most $S$ challenge keys computed with an ephemeral key pair, the difference can be upper bounded by $S \cdot q_{\mathsf{H}_n}/p + S \cdot q_{\widetilde{\mathsf{H}}}/p$ using a hybrid argument. Now we can replace $K_i$, $K_r$ and $\widetilde{K}$ by uniformly random keys.

The rest of the proof is equal to the proof of Theorem 2. The size of the key space of $\mathsf{KEM}_{\mathsf{DDH}}$ is $2^\kappa$ and the bound follows by collecting all probabilities. □

For protocol $\mathsf{AKE}_{\mathsf{FS,DDH}}$, we apply Theorem 3 to show $\mathsf{IND\text{-}FS\text{-}St}$ security. The collision probabilities for $\mathsf{KEM}_{\mathsf{DDH}}$ are already shown in the previous proof. Additionally, we need a strongly unforgeable signature scheme.

**Theorem 6 ($\mathsf{IND\text{-}FS\text{-}St}$ security of $\mathsf{AKE}_{\mathsf{FS,DDH}}$).** *For an $N$-$\mathsf{SUF\text{-}CMA}$ secure signature scheme $\mathsf{SIG}$ and under the $\mathsf{DDH}$ assumption, $\mathsf{AKE}_{\mathsf{FS,DDH}}$ is $\mathsf{IND\text{-}FS\text{-}St}$ secure in the random oracle model. In particular, for any $\mathsf{IND\text{-}FS\text{-}St}$ adversary $\mathcal{A}$ against $\mathsf{AKE}_{\mathsf{FS,DDH}}$ with $N$ parties that establishes at most $S$ sessions and issues at most $T$ queries to the test oracle* Test*, $q_{\mathsf{G}}$ queries to random oracle $\mathsf{G}$, $q_{\mathsf{F}}$ queries to random oracle $\mathsf{F}$, $q_{\widetilde{\mathsf{H}}}$, $q_{\mathsf{H}_n}$ queries to each random oracle $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ and $\mathsf{H}_n$ and at most $q_{\mathsf{H}}$ queries to random oracle $\mathsf{H}$, there exists an adversary $\mathcal{B}$ against $\mathsf{DDH}$ and an adversary $\mathcal{C}$ against $N$-$\mathsf{SUF\text{-}CMA}$ such that*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}FS\text{-}St}}_{\mathsf{AKE}_{\mathsf{FS,DDH}}}(\mathcal{A}) \le 4 \cdot \mathrm{Adv}^{\mathsf{DDH}}_{\mathsf{GGen}}(\mathcal{B}) + 2 \cdot \mathrm{Adv}^{N\text{-}\mathsf{SUF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{C}) + T \cdot \frac{q_{\mathsf{F}} + q_{\mathsf{G}} + q_{\mathsf{H}}}{2^\kappa} \\ + N^2 \cdot \left(\frac{1}{2^{\mu_{\mathsf{SIG}}}} + \frac{1}{p} + \frac{1}{2^\kappa}\right) + S^2 \cdot \left(\frac{2q_{\widetilde{\mathsf{H}}} + 6}{p} + \frac{1}{2^\kappa}\right) \\ + 2NS \cdot \frac{q_{\mathsf{H}_n} + 2}{p} + 2S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} + \frac{4}{p-1} \quad ,\end{aligned}$$

*where $\mu_{\mathsf{SIG}}$ is the collision probability of the key generation algorithm $\mathsf{Gen}_{\mathsf{SIG}}$ and $\kappa$ is a security parameter.*

The signature scheme can be instantiated with the tight scheme based on the $\mathsf{DDH}$ and $\mathsf{CDH}$ assumption proposed by Gjøsteen and Jager in [21], which is also used in their authenticated key exchange protocol.

## Acknowledgments

Tibor Jager was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823. Eike Kiltz was supported by the BMBF iBlockchain project, the EU H2020 PROMETHEUS project 780701, DFG SPP 1736 Big Data, and the DFG Cluster of Excellence 2092 CASA. Doreen Riepel was supported by the Deutsche Forschungsgemeinschaft (DFG) Cluster of Excellence 2092 CASA. Sven Schäge was supported by the German Federal Ministry of Education and Research (BMBF), Project DigiSeal (16KIS0695) and Huawei Technologies Düsseldorf, Project vHSM.

## References

1. Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 629–658. Springer, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-46494-6_26
2. Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 273–304. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_10
3. 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 (May 2000). https://doi.org/10.1007/3-540-45539-6_18
4. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_21
5. Bergsma, F., Jager, T., Schwenk, J.: One-round key exchange with strong security: An efficient and generic construction in the standard model. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 477–494. Springer, Heidelberg (Mar / Apr 2015). https://doi.org/10.1007/978-3-662-46447-2_21
6. Boyd, C., González Nieto, J.M.: On forward secrecy in one-round key exchange. In: Chen, L. (ed.) 13th IMA International Conference on Cryptography and Coding. LNCS, vol. 7089, pp. 451–468. Springer, Heidelberg (Dec 2011)
7. Brzuska, C., Fischlin, M., Warinschi, B., Williams, S.C.: Composability of Bellare-Rogaway key exchange protocols. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 51–62. ACM Press (Oct 2011). https://doi.org/10.1145/2046707.2046716
8. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: 28th ACM STOC. pp. 639–648. ACM Press (May 1996). https://doi.org/10.1145/237814.238015
9. 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 (May 2001). https://doi.org/10.1007/3-540-44987-6_28

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 (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_25
11. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_4
12. Cremers, C.: Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In: Cheung, B.S.N., Hui, L.C.K., Sandhu, R.S., Wong, D.S. (eds.) ASIACCS 11. pp. 80–91. ACM Press (Mar 2011)
13. Cremers, C.J.F.: Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D. (eds.) ACNS 09. LNCS, vol. 5536, pp. 20–33. Springer, Heidelberg (Jun 2009). https://doi.org/10.1007/978-3-642-01957-9_2
14. Cremers, C.J.F., Feltz, M.: Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 734–751. Springer, Heidelberg (Sep 2012). https://doi.org/10.1007/978-3-642-33167-1_42
15. Davis, H., Günther, F.: Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029 (2020), https://eprint.iacr.org/2020/1029
16. Diemert, D., Jager, T.: On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Cryptology ePrint Archive, Report 2020/726 (2020), https://eprint.iacr.org/2020/726
17. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_8
18. 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 (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_17
19. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 467–484. Springer, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8_28
20. 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 (May 2016). https://doi.org/10.1007/978-3-662-49890-3_1
21. 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 (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_4
22. 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 (May 2020). https://doi.org/10.1007/978-3-030-45388-6_14

23. Jager, T., Kiltz, E., Riepel, D., Schäge, S.: Tightly-secure authenticated key exchange, revisited. Cryptology ePrint Archive, Report 2020/1279 (2020), https://eprint.iacr.org/2020/1279
24. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_17
25. Kiltz, E., Pietrzak, K., Stam, M., Yung, M.: A new randomness extraction paradigm for hybrid encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 590–609. Springer, Heidelberg (Apr 2009). https://doi.org/10.1007/978-3-642-01001-9_34
26. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_33
27. Kudla, C., Paterson, K.G.: Modular security proofs for key agreement protocols. In: Roy, B.K. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 549–565. Springer, Heidelberg (Dec 2005). https://doi.org/10.1007/11593447_30
28. 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)
29. Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 378–394. Springer, Heidelberg (Apr 2006). https://doi.org/10.1007/11745853_25
30. Li, Y., Schäge, S.: No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1343–1360. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134006
31. Li, Y., Schäge, S., Yang, Z., Bader, C., Schwenk, J.: New modular compilers for authenticated key exchange. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 14. LNCS, vol. 8479, pp. 1–18. Springer, Heidelberg (Jun 2014). https://doi.org/10.1007/978-3-319-07536-5_1
32. 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 (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_27
33. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_8
34. Schäge, S.: TOPAS: 2-pass key exchange with full perfect forward secrecy and optimal communication complexity. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1224–1235. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813683
35. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), http://eprint.iacr.org/2004/332
36. Yoneyama, K.: One-round authenticated key exchange with strong forward secrecy in the standard model against constrained adversary. In: Hanaoka, G., Yamauchi, T. (eds.) IWSEC 12. LNCS, vol. 7631, pp. 69–86. Springer, Heidelberg (Nov 2012). https://doi.org/10.1007/978-3-642-34117-5_5