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

Table of Contents

1 Introduction

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

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

Multi-Challenge Security definitions. The standard and well established security notion in the context of multiple challenges [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 $b_1, \ldots, b_T$ and, on the i-th Test query, it returns a real challenge key if $b_i = 0$ and a random challenge key if $b_i = 1$. That is, each of the T challenge keys depends on an independent challenge bit b_i. The adversary wins if it can guess correctly one of the T challenge bits b_{i^*} with a probability better than $1/2$. We are not aware of any meaningful multi-bit ROR security game that is tightly equivalent to MBG security.[3] This makes it difficult to provide a good intuition of what MBG security tries to model.

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

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

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

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×KEM+2×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 inefficiency 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 DDH are given in Figure 1. $\mathsf{AKE}_{\mathsf{wFS},\mathsf{DDH}}$ is an implicitly-authenticated two-message protocol $\mathsf{AKE}_{\mathsf{wFS},\mathsf{DDH}}$ in the sense of [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 $\mathsf{AKE}_{\mathsf{FS},\mathsf{DDH}}$ achieves full FS. Instead of using the standard "1×KEM+2×SIG" approach, it replaces one of the signatures with a more efficient MAC and an additional KEM ciphertext, which yields a "2×KEM+1×SIG+1×MAC" construction. When instantiated at "128-bit security" with the most efficient tightly-secure signatures of [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 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].

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.

[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.

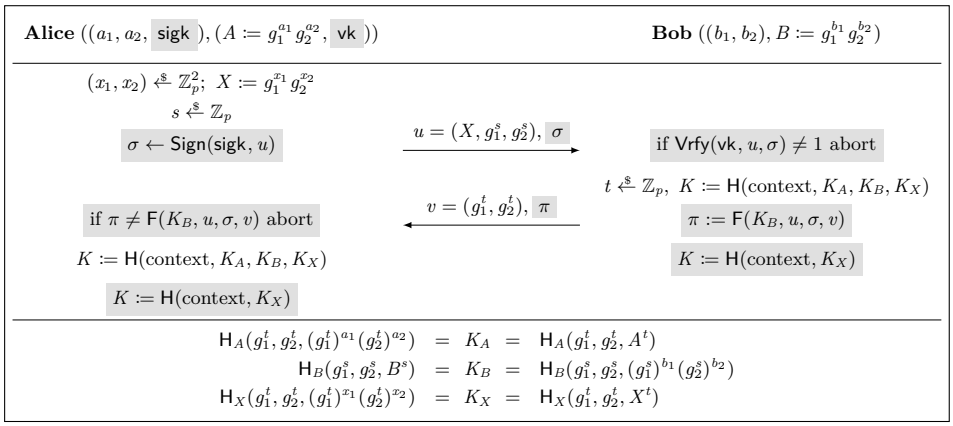

Fig. 1. The two message protocols $\mathsf{AKE}_{\mathsf{wFS,DDH}}$ (without the gray boxes) and $\mathsf{AKE}_{\mathsf{FS,DDH}}$ (including the gray boxes), where K is the resulting session key. We define context $:= (A, B, X,$ vk $, g_1^s, g_2^s, g_1^t, g_2^t,$ σ, π $)$. $\mathsf{H}, \mathsf{H}_A,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

Protocol	**Comm.** $(\mathbb{G}, \{0,1\}^\kappa, \text{Sig})$	**Bytes**	**#Msg.**	**Assumption**	**Auth.**	**Model**	**State Reveal**	**Security Loss**
Protocols with full forward security								
TLS* [16,15]	(2, 4, 2)	704	3	Strong-DH + DDH	expl.	SBG	no	$O(1)$
GJ [21]	(2, 1, 2)	608	3	DDH	expl.	MBG	no	$O(1)$
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								
HMQV [26]	(2, 0, 0)	64	2	CDH	impl.	SBG	yes	$O(TN^2\ell^2)$
CCGJJ [10]	(2, 0, 0)	64	2	Strong-DH	impl.	SBG	no	$O(N)$
$\mathsf{CCGJJ}_{\text{Twin}}$ [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, ℓ to the number of sessions per party and T is the number of test queries. TLS* refers to the TLS 1.3 handshake, instantiated with the tightly-secure signatures of [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.

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.

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,

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

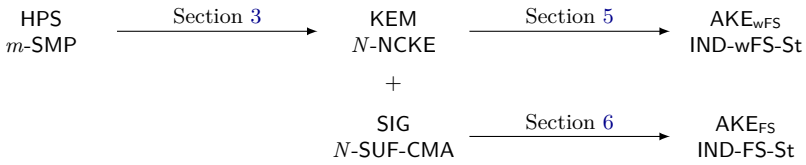

Fig. 3. Overview of our transformations, where N is the maximum number of users in the NCKE security game and in the SUF-CMA security game. The subset membership problem of HPS is m-fold for $m = N \cdot q$, where q is the maximum number of challenge queries in the NCKE security game.

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 LLGW follows the well known 1×KEM+2×SIG approach, meaning that even neglecting the issues with the MBG security model, it is still considerably less efficient than ours (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 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 Gen outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk is the public key and sk the secret key. The encapsulation algorithm inputs a public key pk and outputs a ciphertext c and a key K from the key space $\mathcal{K}$, where c is called an encapsulation of K. The deterministic decapsulation algorithm inputs the secret key sk and a ciphertext c and outputs K.

By μ we denote the *collision probability* of the key generation algorithm. In particular,

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

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

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

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

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

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

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

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

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

HASH PROOF SYSTEM. A hash proof system $\mathsf{HPS} = (\mathsf{Par}, \mathsf{Priv}, \mathsf{Pub})$ consists of three algorithms. The randomized algorithm Par generates parametrized instances of $\mathsf{par} = (\mathit{group}, \mathcal{Z}, \mathcal{Y}, \mathcal{X}, \mathcal{PK}, \mathcal{SK}, \Lambda_{(\cdot)} : \mathcal{Y} \to \mathcal{Z}, \mu : \mathcal{SK} \to \mathcal{PK})$, where *group* may contain additional structural parameters. The deterministic public evaluation algorithm Pub inputs the projection key $\mathsf{pk} = \mu(\mathsf{sk})$, $c \in \mathcal{X}$ and a witness r of the fact that $c \in \mathcal{X}$ and returns $Z = \Lambda_{\mathsf{sk}}(c)$. The deterministic private evaluation algorithm Priv takes $\mathsf{sk} \in \mathcal{SK}$ and

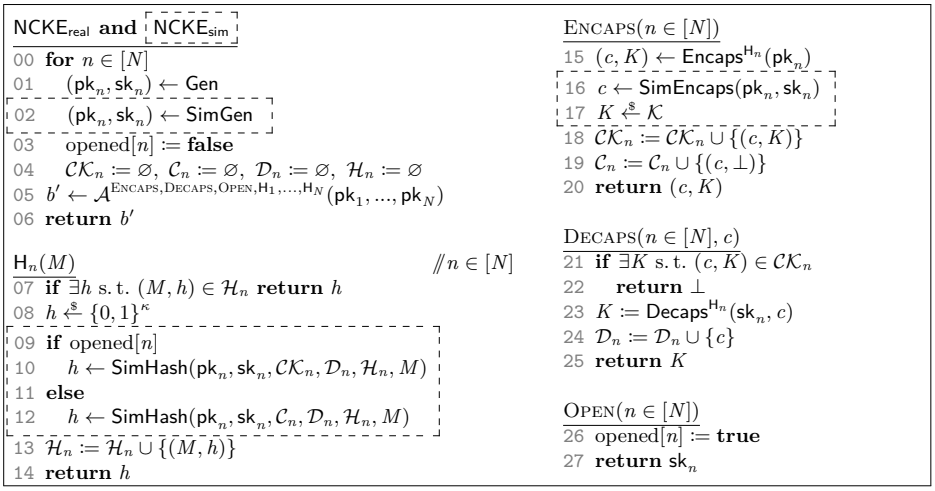

Fig. 4. Real and simulated game for N-receiver non-committing key encapsulation in the random oracle model.

returns $\Lambda_{\mathsf{sk}}(c)$ without knowing a witness. Furthermore, we assume that μ is efficiently computable and that there are efficient algorithms for sampling $c \in \mathcal{X}$ uniformly together with a witness r, sampling $c \in \mathcal{Y}$ uniformly and checking membership in $\mathcal{Y}$.

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

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

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

N-RECEIVER NCKE FROM HPS. We use a k-entropic hash proof system $\mathsf{HPS} = (\mathsf{Par}, \mathsf{Pub}, \mathsf{Priv})$ with m-fold subset membership problem and a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ in order to construct a key encapsulation algorithm KEM and a simulator Sim as shown in Figures 5 and 6. The encapsulation algorithm Encaps samples an element c from $\mathcal{X}$ and a witness r. It runs the public evaluation algorithm and computes the key K as $\mathsf{H}(c, \mathsf{Pub}(\mathsf{pk}, c, r))$. The decapsulation algorithm Decaps uses the result of the private evaluation algorithm Priv as input to H to compute K. Instead of sampling an element from $\mathcal{X}$, the $\mathsf{SimEncaps}$ 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 H satisfies $\mathsf{Priv}(\mathsf{sk}, c) = Z$, then the output value h is set to K.

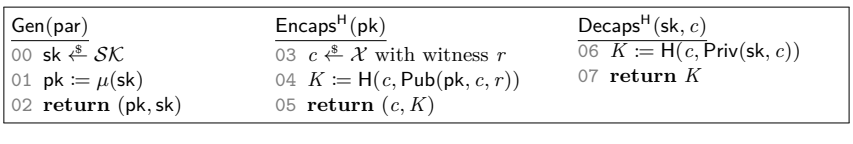

Fig. 5. Key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$.

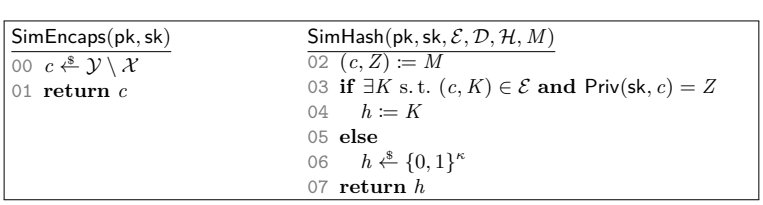

Fig. 6. Simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$ for KEM, where $\mathsf{SimGen} = \mathsf{Gen}$. List $\mathcal{E}$ is either $\mathcal{CK}$ or $\mathcal{C}$.

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

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

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

Proof. Let $\mathcal{A}$ be an adversary against KEM and Sim in the NCKE games. Consider the sequence of games in Figure 7.

Game G_0. This is the original $\mathsf{NCKE}_{\mathsf{real}}$ game, hence

$$\Pr[G_0^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{NCKE}^{\mathcal{A}}_{\mathsf{real}} \Rightarrow 1] \ .$$

Game G_1. In game G_1, the Encaps oracle is modified in a way that it uses the private evaluation algorithm to compute K in line 22. It holds that $\mathsf{Pub}(\mathsf{pk}, c, r) = \mathsf{Priv}(\mathsf{sk}, c)$. Thus, this does not change the adversary's view and

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

Game G_2. The Encaps oracle now chooses the ciphertext from $\mathcal{Y} \setminus \mathcal{X}$ in line 17. We claim that

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathrm{Adv}^{(N\cdot q_E)\text{-}\mathsf{SM}}_{\mathsf{HPS}}(\mathcal{B}) \ . \tag{1}$$

In Figure 8, we construct adversary $\mathcal{B}$ against the $(N \cdot q_E)$-fold subset membership problem. $\mathcal{B}$ inputs sets $\mathcal{Y}, \mathcal{X}$ and ciphertexts $c_{n,k}$, where $n \in [N], k \in [q_E]$. If $\mathcal{B}$'s input values are elements from $\mathcal{X}$, then $\mathcal{B}$ perfectly simulates G_1. Otherwise, if the input elements are from $\mathcal{Y} \setminus \mathcal{X}$, $\mathcal{B}$ simulates G_2. This yields Equation 1.

Game G_3. In game G_3, we raise flag BAD in line 19 and abort if the Encaps oracle chooses a ciphertext that was issued to the Decaps oracle before. As a challenge ciphertext is chosen uniformly at random from $\mathcal{Y} \setminus \mathcal{X}$, the probability that BAD is raised for one specific challenge ciphertext is at most $q_D/|\mathcal{Y} \setminus \mathcal{X}|$. Union bound over all challenge ciphertexts yields

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathrm{BAD}] \leq \frac{N \cdot q_E \cdot q_D}{|\mathcal{Y} \setminus \mathcal{X}|} \ .$$

Game G_4. In game G_4, we use internal hash functions H'_n to compute K in line 23. These are not directly accessible to the adversary and independent of the random oracle as long as the secret key has not been opened. However, if the adversary opens the secret key of user n, then it can simply compute the value $Z = \mathsf{Priv}(\mathsf{sk}_n, c)$ for any challenge ciphertext c of that user. This is why we have to patch the random oracle and output $\mathsf{H}'_n(c, Z)$ whenever $\mathcal{A}$ issues such a query (lines 09 and 10).

The only possibility for $\mathcal{A}$ to notice the difference is when it queries H_n on $(c, Z = \mathsf{Priv}(\mathsf{sk}_n, c))$ before sk_n is opened, where c is a ciphertext output by Encaps. Here, we use the fact that HPS is k-entropic and show that

$$\left|\Pr[G_4^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{N \cdot q_E \cdot q_{\mathsf{H}}}{2^k} \ .$$

Using a hybrid argument, we modify the computation of K using function H'_n independent of the random oracle as long as the adversary does not see the corresponding sk_n. Therefore, we parameterize the hybrids with j, where $j \in [N \cdot q_E]$ denotes that K is replaced in the first j challenge ciphertexts.

In the following, we consider two consecutive hybrids, where the only difference is that the computation of K in the j-th query is modified. Let pk_{n^*} be the public key of the corresponding user and (c^*, K^*) the challenge ciphertext. In hybrid H_{j-1}, the adversary observes $(c^*, K^* = \mathsf{H}_{n^*}(c^*, Z^*))$, where $c^* \in \mathcal{Y} \setminus \mathcal{X}$ and $Z^* = \mathsf{Priv}(\mathsf{sk}_{n^*}, c^*)$. In hybrid H_j, the key K is computed with H'_{n^*} independent of H_{n^*} assuming that the adversary has not opened sk_{n^*}. Thus, in order to notice the difference, the adversary

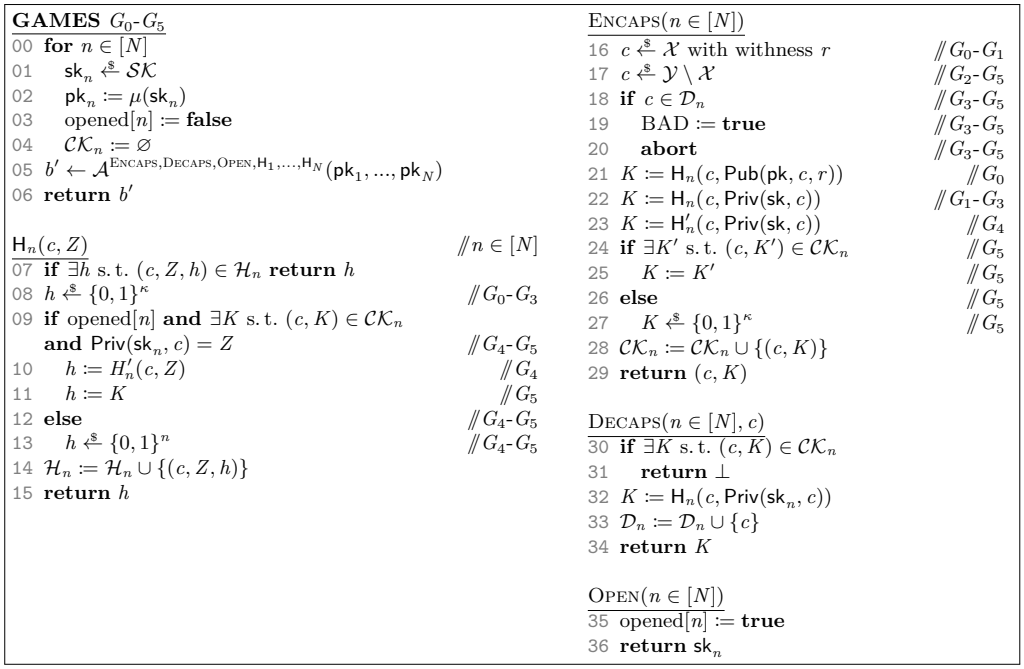

Fig. 7. Games G_0-G_5 for the proof of Theorem 1. H'_n in line 23 is used as an internal hash function which is not directly accessible to the adversary.

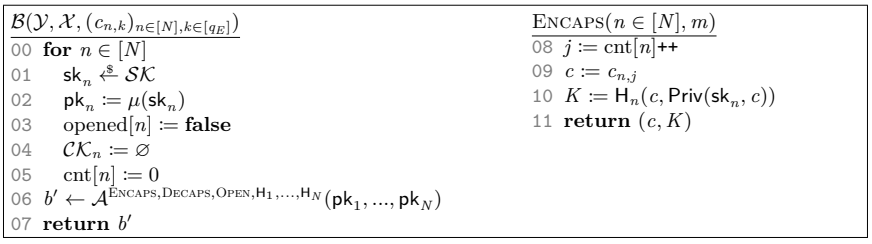

Fig. 8. Adversary $\mathcal{B}$ against the $(N \cdot q_E)$-fold subset membership problem for the proof of Theorem 1, where H_n for $n \in [N]$, Decaps and Open are defined as in G_1 of Fig. 7.

must query H_{n^*} on (c^*, Z^*). As for every $c \in \mathcal{Y} \setminus \mathcal{X}$, we have that $H_\infty(\mathsf{Priv}(\mathsf{sk}_{n^*}, c^*) \mid \mathsf{pk}_{n^*}) \geq k$, we can bound the probability of this event by the number of random oracle queries:

$$\left|\Pr[H_j^{\mathcal{A}} \Rightarrow 1] - \Pr[H_{j-1}^{\mathcal{A}} \Rightarrow 1]\right| \leq \frac{q_{\mathsf{H}}}{2^k} \ .$$

Note that a query (n^*, c) to Decaps, where $c \neq c^*$, will not reveal any additional information because the output of H_{n^*} will be different anyway.

Game G_5. In game G_5, the Encaps oracle chooses key K uniformly at random in line 27. If the same ciphertext as in a previous challenge is generated, the key from that challenge will be used again to maintain consistency (see lines 24 and 25). In addition to that, the random oracle has to be modified again so that it now outputs the same K as chosen before in case a secret key has already been opened and Z is computed correctly (see line 11). The adversary's view does not change as $\mathsf{H}_n(c, \mathsf{Priv}(\mathsf{sk}_n, c)) = K$ for every $(c, K) \in \mathcal{CK}_n$ if sk_n is opened. If sk_n is not opened, K is independent of H_n in both games G_4 and G_5. Hence,

$$\Pr[G_5^{\mathcal{A}} \Rightarrow 1] = \Pr[G_4^{\mathcal{A}} \Rightarrow 1] \ .$$

Finally, observe that the last game G_5 is the original $\mathsf{NCKE}_{\mathsf{sim}}$ game. Hence,

$$\Pr[G_5^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{NCKE}_{\mathsf{sim}}^{\mathcal{A}} \Rightarrow 1] \ .$$

Collecting all probabilities yields the bound stated in Theorem 1. □

We will give two concrete instantiations, one based on the DDH assumption (Section 7.1) and one based on the higher residuosity assumption (Appendix A).

4 Security Model for Two-Message Authenticated Key Exchange

A two-message key exchange protocol $\mathsf{AKE} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{R}}, \mathsf{Der}_{\mathrm{I}})$ consists of four algorithms which are executed interactively by two parties as shown in Figure 9. We denote the party which initiates the session by P_i and the party which responds to the session by P_r. The key generation algorithm $\mathsf{Gen}_{\mathsf{AKE}}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ for one party. The initialization algorithm $\mathsf{Init}_{\mathrm{I}}$ inputs the initiator's long-term secret key sk_i and the responder's long-term public key pk_r and outputs a message I and a state st. The responder's derivation algorithm $\mathsf{Der}_{\mathrm{R}}$ takes as input the responder's long-term secret key sk_r, the initiator's long-term public key pk_i and a message I. It computes a message R and a session key K. The initiator's derivation algorithm $\mathsf{Der}_{\mathrm{I}}$ inputs the initiator's long-term secret key sk_i, the responder's long-term public key pk_r, a message R and a state st. It outputs a session key K.

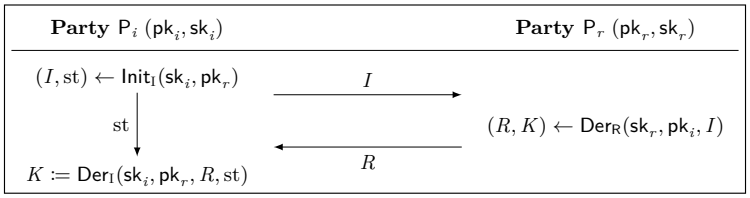

Fig. 9. Running a key exchange protocol between two parties.

Note that in contrast to the initiating party P_i, the responding party P_r will not be required to save any (secret) state information besides the session key K. The session key can be derived immediately after receiving the initiator's message.

Following [23], we define a game-based security model for authenticated key exchange using pseudocode. Our models for two different levels of security are given in Figure 10. We consider N parties $\mathsf{P}_1, ..., \mathsf{P}_N$ with long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$, $n \in [N]$. Each session between two parties has a unique identification number sID and variables which are defined relative to sID:

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

To establish a session between two parties, the adversary is given access to oracles $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathrm{R}}$, where the first one starts a session of type "In" 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 $\textsc{Corrupt}, \textsc{Reveal}$ and $\textsc{Rev-State}$ to obtain secret information. As the responder can directly compute the key in a two-message protocol, we only require the initiator to store a state. The state contains information that is needed to compute the session key when the response is received, so it will consist of public and private information. We do not require to reveal the full randomness as in the eCK model [28]. A $\textsc{Rev-State}$

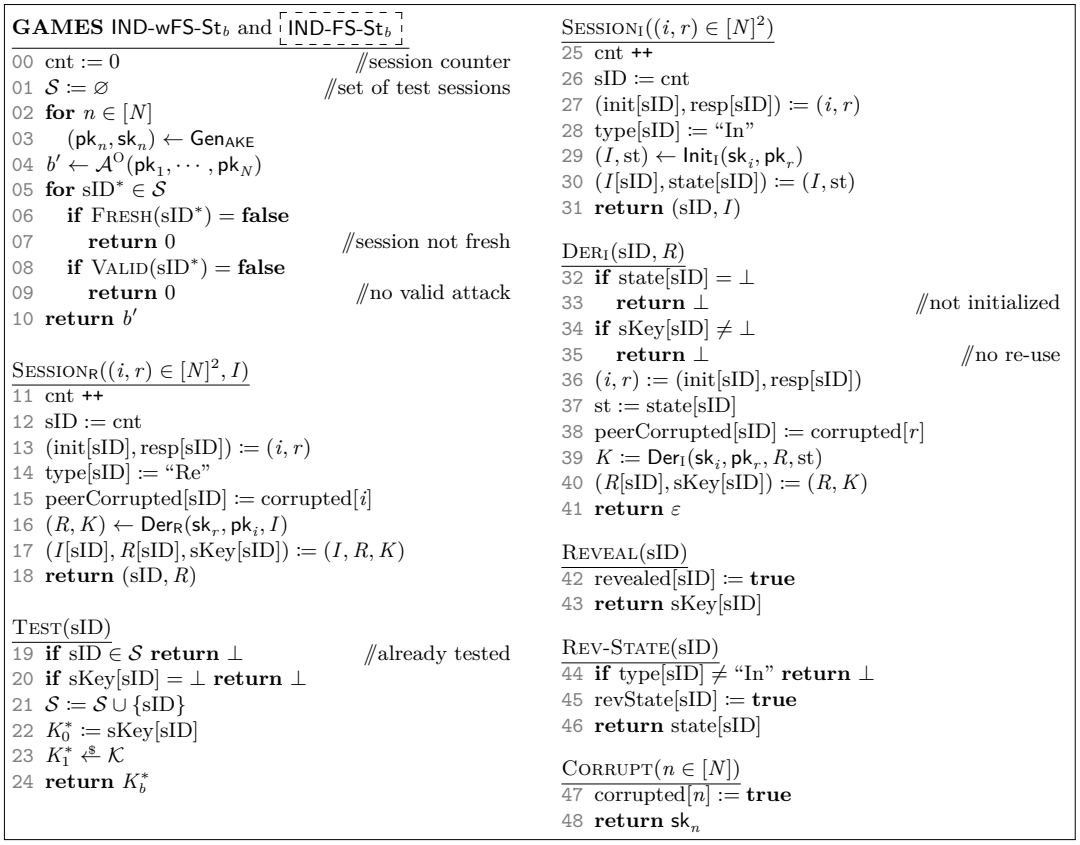

Fig. 10. Games IND-wFS-St$_b$ and IND-FS-St$_b$ for AKE, where $b \in \{0, 1\}$. $\mathcal{A}$ has access to oracles O := {Session$_{\mathrm{I}}$, Session$_{\mathrm{R}}$, Der$_{\mathrm{I}}$, Reveal, Rev-State, Corrupt, Test}. Helper procedures Fresh and Valid are defined in Figure 11. If there exists any test session which is neither fresh nor valid, the game will return 0.

query may be issued at any time. We use the following boolean values to keep track of which queries the adversary made:

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

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

- **Matching Session**: Two sessions sID, sID$'$ *match* if the same parties are involved (init[sID] = init[sID$'$] and resp[sID] = resp[sID$'$]), the messages sent and received are the same (I[sID] = I[sID$'$] and R[sID] = R[sID$'$]) and they are of different types (type[sID] $\neq$ type[sID$'$]).
- **Partially Matching Session**: A session sID$'$ of type "In" is *partially matching* to session sID of type "Re" if the initial messages are the same (I[sID] = I[sID$'$]).

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

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

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

Fig. 11. Helper procedures FRESH and VALID for games IND-wFS-St and IND-FS-St defined in Figure 10. Procedure FRESH checks if the adversary performed some trivial attack. In procedure VALID, each attack is evaluated by the set of variables shown in Table 1 (IND-wFS-St) or Table 2 (IND-FS-St) and checks if an allowed attack was performed. If the values of the variables are set as in the corresponding row, the attack was performed, i. e. attack = **true**, and thus the session is valid.

– **Validity**: A (test) session is called *valid* if it is fresh and the adversary performed any attack which is defined in the security model. We capture this with attack tables (cf. Tables 1 and 2). A description of how to read the tables is given below.

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

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

This yields a full table of 24 attacks (cf. Table 3 in Appendix B), in particular capturing *key compromise impersonation* (KCI) and *maximal exposure* (MEX) attacks. An attack was performed if the variables are set to the corresponding values in the table. However, 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. A more detailed justification on how the distilled tables are obtained by pointing out trivial attacks is given in Appendix B. Note that the numbering of attacks in the distilled tables is inherited from the full table.

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

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

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

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

Table 1. Distilled table of attacks for wFS adversaries against two-message protocols. This table is obtained from Table 3 in Section B by using that responders do not have a state and that we are considering weak forward security. The numbering of attacks is inherited from Table 3. 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*) : revState[sID]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$	$\exists$sID $\in \mathfrak{P}$(sID*) : revState[sID]	$\lvert\mathfrak{P}(\text{sID}^*)\rvert$
(0)	**multiple partially matching sessions**	–	–	–	–	–	–	–	–	> 1
(1∨2)	**(long-term, long-term)**	–	–	–	–	**F**	**F**	1	–	–
(7∨8)	**(state, long-term)**	**F**	–	–	–	–	–	1	–	–
(10)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	**F**	1
(16)	**(state, long-term)**	**F**	–	–	"Re"	**F**	n/a	0	–	1
(17)	**(long-term, long-term)**	–	–	**F**	"In"	**F**	n/a	0	n/a	0
(18)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	n/a	0
(23)	**(state, long-term)**	**F**	–	**F**	"In"	–	n/a	0	n/a	0

Table 2. Distilled table of attacks for full FS adversaries against two-message protocols. This table is obtained from Table 3 in Section B by removing redundant rows and using that responders do not have a state. The numbering of attacks is inherited from Table 3. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. **F** means "false" and "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

Definition 2 (Key Indistinguishability of AKE). *We define games* $\mathsf{IND\text{-}wFS\text{-}St}_b$ *and* $\mathsf{IND\text{-}FS\text{-}St}_b$ *for* $b \in \{0,1\}$ *as in Figures 10 and 11. The advantage of an adversary* $\mathcal{A}$ *against* AKE *in these games is defined as*

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

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

4.1 Relation to other Definitions

In this section, we will refer to the most widely used security definitions for authenticated key exchange protocols. In the first place, these include the CK model [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-AA security in [23]. 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 [24,1,23]. 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 $\mathsf{AKE}_{\mathsf{wFS}} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{R}}, \mathsf{Der}_{\mathrm{I}})$ as shown in Figures 12 and 13. W.l.o.g. $\mathsf{KEM}_{\mathsf{CPA}}$, $\mathsf{KEM}_{\mathsf{CCA}}$, $\mathsf{AKE}_{\mathsf{wFS}}$ have identical key space $\mathcal{K}$. Each party holds a long-term key pair $(\mathsf{pk}, \mathsf{sk})$ for $\mathsf{KEM}_{\mathsf{CCA}}$ and a symmetric key k to encrypt the secret state information which has to be stored by the initiating party. State encryption protects against state attacks and is implemented using a symmetric encryption scheme defined as $\mathsf{E}_k(\mathrm{st}') := (IV, \mathsf{G}(k, IV) \oplus \mathrm{st}')$ for a random nonce IV. Here $\mathsf{G} : \{0,1\}^* \to \{0,1\}^d$ is a random oracle and d is an integer denoting the maximum bit length of the unencrypted state st'. The protocol uses an additional cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ to output the session key.

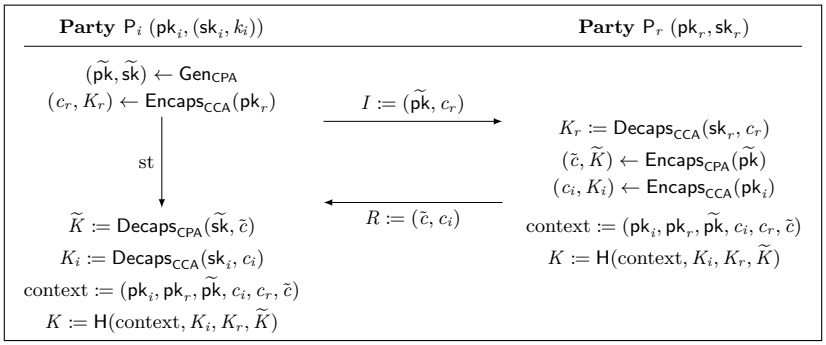

Fig. 12. Visualization: Running protocol $\mathsf{AKE}_{\mathsf{wFS}}$ between two parties.

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

Theorem 2 ($\mathsf{KEM}_{\mathsf{CPA}}$ NCKE-CPA + $\mathsf{KEM}_{\mathsf{CCA}}$ NCKE-CCA $\overset{\text{ROM}}{\Rightarrow}$ $\mathsf{AKE}_{\mathsf{wFS}}$ IND-wFS-St)**.** *For any* IND-wFS-St *adversary* $\mathcal{A}$ *against* $\mathsf{AKE}_{\mathsf{wFS}}$ *with* N *parties that establishes at most* S *sessions and issues at most* T *queries to the test oracle* TEST, q_{G} *queries to random oracle* G *and at most* q_{H} *queries to random oracle* H*, there exists an* N-NCKE-CCA *adversary* $\mathcal{B}$ *against* $\mathsf{KEM}_{\mathsf{CCA}}$ *and* $\mathsf{Sim}_{\mathsf{CCA}}$ *and an* S-NCKE-CPA *adversary* $\mathcal{C}$ *against* $\mathsf{KEM}_{\mathsf{CPA}}$ *and* $\mathsf{Sim}_{\mathsf{CPA}}$ *such that*

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

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

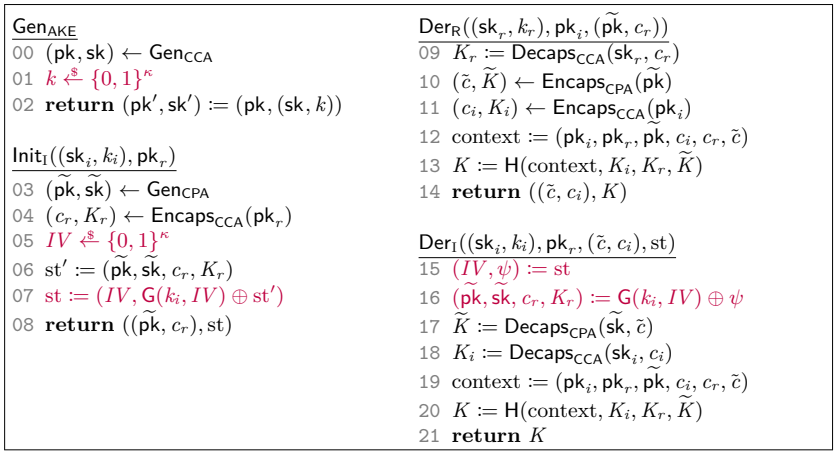

Fig. 13. Authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{wFS}}$ from $\mathsf{KEM}_{\mathsf{CPA}}$ and $\mathsf{KEM}_{\mathsf{CCA}}$. Lines written in purple color are only used to encrypt the state.

Proof. Let $\mathcal{A}$ be an adversary against IND-wFS-St security of $\mathsf{AKE}_{\mathsf{wFS}}$, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is the maximum number of test queries. Consider the sequence of games in Figures 14 and 15.

GAMES $G_{0,b}$. These are the original IND-wFS-St$_b$ games. In order to exclude collisions, we implicitly assume that all key pairs, long-term keys as well as ephemeral keys generated by $\mathsf{Gen}_{\mathsf{CCA}}$ and $\mathsf{Gen}_{\mathsf{CPA}}$, and all ciphertexts output by the $\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{CCA}}$ and $\mathsf{Encaps}^{\widetilde{\mathsf{H}}}_{\mathsf{CPA}}$ algorithms are distinct. (If such a collision happens at any time in the game, we would abort. However, for sake of readability we do not explicitly write that in the code of games $G_{0,b}$.)

We consider this in our bound. Therefore, let μ_{CCA} and μ_{CPA} be the collision probabilities of the key generation algorithms $\mathsf{Gen}_{\mathsf{CCA}}$ and $\mathsf{Gen}_{\mathsf{CPA}}$ and let γ_{CCA} and γ_{CPA} be the spreadness parameters of $\mathsf{KEM}_{\mathsf{CCA}}$ and $\mathsf{KEM}_{\mathsf{CPA}}$. Then by union bound and the birthday bound, the upper bound for key collisions is $N^2/2^{\mu_{\mathsf{CCA}}} + S^2/2^{\mu_{\mathsf{CPA}}}$ and for ciphertext collisions $S^2/2^{\gamma_{\mathsf{CCA}}} + S^2/2^{\gamma_{\mathsf{CPA}}}$, as we have N parties, at most S sessions with at most one ephemeral key pair and at most two ciphertexts. We also assume that values k_n and IV are distinct, which adds the additional term $N^2/2^{\kappa} + S^2/2^{\kappa}$, where κ is the bit length of k_n and IV.

We additionally store the state of a session sID in plaintext in variable state$'$[sID] (line 59, Fig. 14) which is directly accessed in $\mathrm{DER}_{\mathrm{I}}$, instead of decrypting the state. This is only conceptual. For bookkeeping, we introduce the two sets $\mathcal{C}$ and $\mathcal{CK}$ from the NCKE-CCA game in lines 35, 36 (Fig. 15) and 51, 52 (Fig. 14). In total, we have

$$\begin{aligned} |\Pr[\text{IND-wFS-St}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\text{IND-wFS-St}_0^{\mathcal{A}} \Rightarrow 1]| &\leq \left|\Pr[G_{0,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{0,0}^{\mathcal{A}} \Rightarrow 1]\right| \\ &+ N^2 \cdot \left(2^{-\mu_{\mathsf{CCA}}} + 2^{-\kappa}\right) + S^2 \cdot \left(2^{-\mu_{\mathsf{CPA}}} + 2^{-\gamma_{\mathsf{CCA}}} + 2^{-\gamma_{\mathsf{CPA}}} + 2^{-\kappa}\right) . \end{aligned} \tag{2}$$

In the following, we want to use the property of receiver non-committing key encapsulation of $\mathsf{KEM}_{\mathsf{CCA}}$. Therefore, we use the simulator $\mathsf{Sim}_{\mathsf{CCA}} = (\mathsf{SimGen}_{\mathsf{CCA}}, \mathsf{SimEncaps}_{\mathsf{CCA}}, \mathsf{SimHash}_{\mathsf{CCA}})$ which is defined relative to $\mathsf{KEM}_{\mathsf{CCA}}$.

GAMES $G_{1,b}$. In games $G_{1,b}$, we use the $\mathsf{SimGen}_{\mathsf{CCA}}$ algorithm to generate long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$ in line 04 (Fig. 14). $\mathrm{SESSION}_{\mathrm{I}}$ uses the $\mathsf{SimEncaps}_{\mathsf{CCA}}$ algorithm to compute c_r in line 49 and draws a random key K_r in line 50. To maintain consistency, challenges are saved in line 52 and K_r is retrieved in $\mathrm{SESSION}_{\mathrm{R}}$ (line 28, Fig. 15) when the same c_r is issued. The same is done for ciphertexts c_i and keys K_i in $\mathrm{SESSION}_{\mathrm{R}}$: $\mathsf{SimEncaps}_{\mathsf{CCA}}$ is used to generate c_i (line 33, Fig. 15), K_i is drawn uniform at random (line 34) and both are saved and retrieved (line 36, Fig. 15 and line 73, Fig. 14). Furthermore, the $\mathsf{SimHash}_{\mathsf{CCA}}$ algorithm is used in random oracles H_n, where $n \in [N]$. In case party n is corrupted, i. e. sk_n is known to $\mathcal{A}$, we call $\mathsf{SimHash}_{\mathsf{CCA}}$ with set $\mathcal{CK}_n$, otherwise with set $\mathcal{C}_n$.

For $b \in \{0,1\}$, we construct adversaries $\mathcal{B}_b$ against N-NCKE-CCA security of $\mathsf{KEM}_{\mathsf{CCA}}$ in Figure 16. $\mathcal{B}_b$ inputs long-term public keys $\mathsf{pk}_1, ..., \mathsf{pk}_N$ and has access to oracles $\mathrm{ENCAPS}, \mathrm{DECAPS}$ and OPEN as well as random oracles H'_n, where $n \in [N]$. $\mathcal{B}_b$ generates N symmetric keys k_n which are part of the long-term secret key and forwards its input public keys to $\mathcal{A}$.

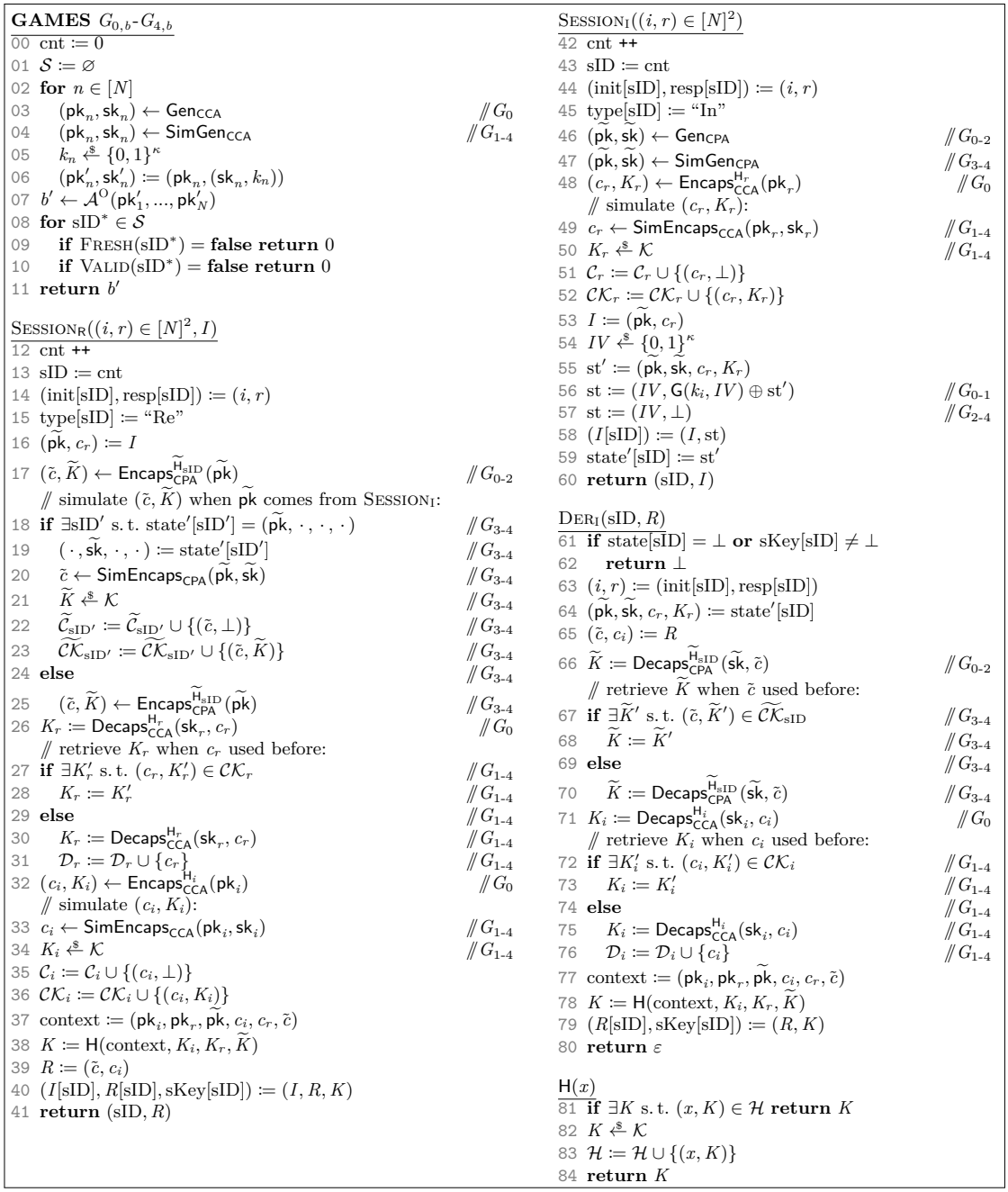

Fig. 14. Games $G_{0,b}$-$G_{4,b}$ for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{SessionI}, \textsc{SessionR}, \textsc{DerI}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, ..., \mathsf{H}_N, \widetilde{\mathsf{H}}_1, ..., \widetilde{\mathsf{H}}_S\}$. Reveal and Corrupt are defined as in the original IND-wFS-St game (Fig. 10). Rev-State, Test, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for $\mathrm{sID} \in [S]$ and H_n for $n \in [N]$ are defined in Fig. 15.

If $\mathcal{A}$ queries SessionI, $\mathcal{B}_b$ generates an ephemeral key pair $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$. Next, $\mathcal{B}_b$ calls Encaps on party r in line 35. As described before, challenges are saved and retrieved when ciphertexts match. If they do not match which means that c_r issued to SessionR is new, Decaps is queried to receive the corresponding key K_r in line 19. Next, $\mathcal{B}_b$ calls the Encaps oracle on party i (line 20) and in DerI key K_i is retrieved (line 50) or Decaps is queried (line 52).

If $\mathcal{A}$ queries Corrupt on party n, $\mathcal{B}_b$ queries Open to obtain sk_n and outputs both sk_n and k_n. Queries to a random oracle H_n are forwarded to H'_n.

If $\mathcal{B}_b$ is in the NCKE-CCA$_{\mathsf{real}}$ game, it perfectly simulates $G_{0,b}$. Otherwise, if $\mathcal{B}_b$ is in the NCKE-CCA$_{\mathsf{sim}}$ game, it perfectly simulates $G_{1,b}$. We have

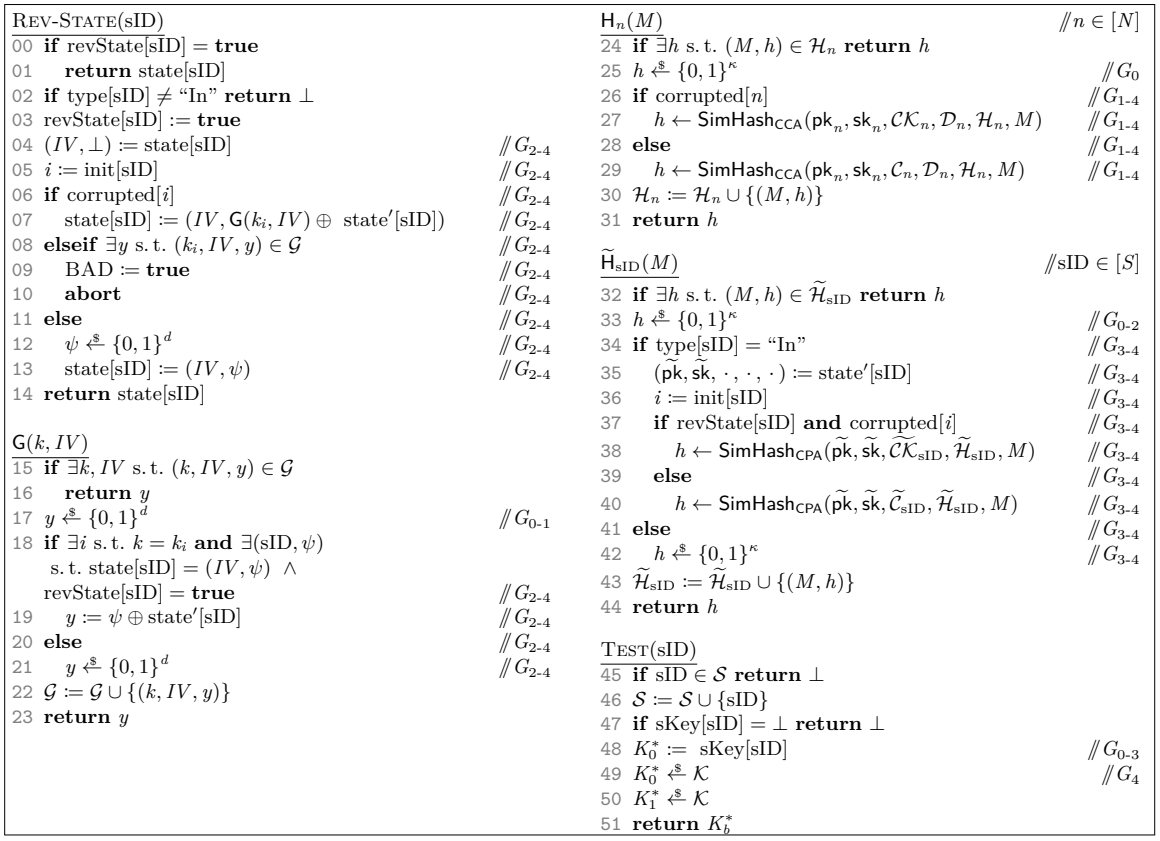

Fig. 15. Oracles REV-STATE, G, H_n for $n \in [N]$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for $\mathrm{sID} \in [S]$ and TEST for games $G_{0,b}$-$G_{4,b}$ in Figure 14.

$$\begin{aligned}\left|\Pr[G^{\mathcal{A}}_{1,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{0,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CCA}^{\mathcal{B}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CCA}^{\mathcal{B}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \mathrm{Adv}^{N\text{-}\mathsf{NCKE\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{CCA}},\mathsf{Sim}_{\mathsf{CCA}}}(\mathcal{B}_b) \end{aligned} \tag{3}$$

Instead of using the algorithms $\mathsf{Gen}_{\mathsf{CPA}}$ and $\mathsf{Encaps}_{\mathsf{CPA}}$, we now also want to use the simulator $\mathsf{Sim}_{\mathsf{CPA}} = (\mathsf{SimGen}_{\mathsf{CPA}}, \mathsf{SimEncaps}_{\mathsf{CPA}}, \mathsf{SimHash}_{\mathsf{CPA}})$ for the ephemeral keys and ciphertexts. However, we first introduce an intermediate game which moves the encryption of the state to the REV-STATE oracle. We do this to prepare the reduction to NCKE-CPA security, where $\widetilde{\mathsf{sk}}$ will not be available and thus the state cannot be computed in the first place. This means we first delay the computation of the state as long as possible in games $G_{2,b}$ and then in games $G_{3,b}$ the simulator will finally be used.

GAMES $G_{2,b}$. Here, we move the encryption of the state from $\mathrm{SESSION}_{\mathrm{I}}$ (line 56, Fig. 14) to the REV-STATE oracle. When REV-STATE is queried, we check if the initiator i is corrupted in line 06 and honestly compute the state because then the adversary can simply make the same computation. Next, we check if the adversary already made a query to G, where (k_i, IV) are as in the corresponding session. If this is the case, we raise flag BAD in line 09 and abort. Otherwise, we choose a random string ψ in line 12 and return this value. Line 00 ensures that answers will be consistent when REV-STATE is queried twice on the same sID. However, we have to patch G for the case that $\mathcal{A}$ issues a query with the correct symmetric key k_i such that the random value ψ decrypts correctly. Note that if BAD is not raised, the adversary's view is the same in games $G_{2,b}$ and $G_{1,b}$. As BAD implies that G is queried on any correct pair (k_i, IV) although $IV \in \{0,1\}^\kappa$ is unknown and $k_i \in \{0,1\}^\kappa$ is also unknown because otherwise the state would have been computed honestly in line 07, we have

$$\left|\Pr[G_{2,b} \Rightarrow 1] - \Pr[G_{1,b} \Rightarrow 1]\right| \leq \Pr[\mathrm{BAD}] \leq S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} \ .$$

GAMES $G_{3,b}$. In games $G_{3,b}$, $\mathrm{SESSION}_{\mathrm{I}}$ uses the $\mathsf{SimGen}_{\mathsf{CPA}}$ algorithm to generate ephemeral key pairs $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ in line 47 (Fig. 14). In lines 19-23 (Fig. 15), we first recover the ephemeral secret key $\widetilde{\mathsf{sk}}$ from

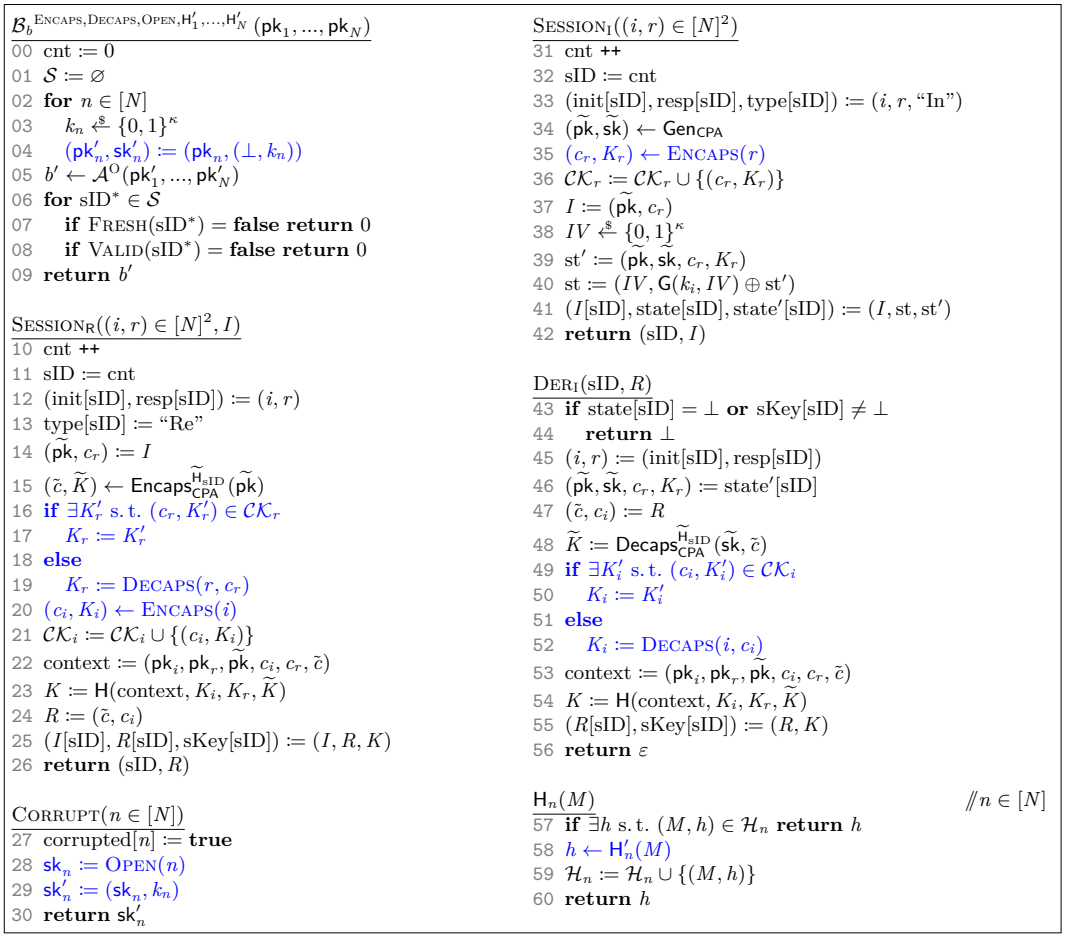

Fig. 16. Adversaries $\mathcal{B}_b$ against N-NCKE-CCA for the proof of Eqn. (3). $\mathcal{A}$ has access to oracles $\text{O} := \{\text{SESSION}_\text{I}, \text{SESSION}_\text{R}, \text{DER}_\text{I}, \text{REVEAL}, \text{REV-STATE}, \text{CORRUPT}, \text{TEST}, \mathsf{G}, \mathsf{H}, \widetilde{\mathsf{H}}_1, \ldots \widetilde{\mathsf{H}}_S, \mathsf{H}_1, \ldots, \mathsf{H}_N\}$, where REVEAL, REV-STATE, TEST, H and $\widetilde{\mathsf{H}}_{\text{sID}}$ for sID $\in [S]$ are defined as in Figure 14 resp. 15. Lines written in blue color highlight how the adversary simulates $G_{0,b}$ and interpolates to $G_{1,b}$.

the state of the corresponding initiating session if the ephemeral public key specified in I was output by SESSION_I, i. e., it was not chosen by the adversary. Next, the $\mathsf{SimEncaps}_{\mathsf{CPA}}$ algorithm is used to compute $\tilde{c}$ and we draw a random key $\widetilde{K}$. To maintain consistency, we save challenges to restore $\widetilde{K}$ later in line 68 (Fig. 14) if the same $\tilde{c}$ is queried to DER_I. Furthermore, we use the $\mathsf{SimHash}_{\mathsf{CPA}}$ algorithm in random oracles $\widetilde{\mathsf{H}}_{\text{sID}}$, but only for those sessions that choose their own ephemeral key pair $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ in SESSION_I as explained above. In particular, we first check if sID is a session of type "In" in line 34 (Fig. 15). In case the state of that session is revealed and the initiator is corrupted, we call $\mathsf{SimHash}_{\mathsf{CPA}}$ with set $\widetilde{\mathcal{CK}}_{\text{sID}}$, otherwise with set $\widetilde{\mathcal{C}}_{\text{sID}}$.

For $b \in \{0,1\}$, we construct adversaries $\mathcal{C}_b$ against S-NCKE-CPA security of $\mathsf{KEM}_{\mathsf{CPA}}$ in Figures 17 and 18. $\mathcal{C}_b$ inputs S ephemeral public keys $\widetilde{\mathsf{pk}}_1, \ldots, \widetilde{\mathsf{pk}}_S$ and has access to oracles ENCAPS, OPEN and random oracles $\widetilde{\mathsf{H}}'_s$, where $s \in [S]$.

If $\mathcal{A}$ queries SESSION_I, $\mathcal{C}_b$ computes (c_r, K_r) and sets $\widetilde{\mathsf{pk}}$ to $\widetilde{\mathsf{pk}}_{\text{sID}}$ in line 38 (Fig. 17). Note that it cannot assign $\widetilde{\mathsf{sk}}$ here and thus cannot define st′ in line 45 because $\widetilde{\mathsf{sk}}_{\text{sID}}$ is unknown. Thus, it cannot compute the state, but will only draw IV. When $\mathcal{A}$ issues a query to REV-STATE, $\mathcal{C}_b$ checks if the initiator is corrupted, calls OPEN to obtain the corresponding ephemeral secret key $\widetilde{\mathsf{sk}}$ (line 23, Fig. 18) and will then compute and output the complete state. If the initiator is not corrupted and $\mathcal{C}_b$ does not abort, ψ is chosen uniformly at random. If the adversary queries G on (k_i, IV), we patch the random oracle by calling OPEN in line 02 (Fig. 18) and computing the correct value for st′, thus determining output value y.

$\underline{\mathcal{C}_b^{\text{ENCAPS,OPEN},\widetilde{\mathsf{H}}'_1,\ldots,\widetilde{\mathsf{H}}'_S}(\widetilde{\mathsf{pk}}_1,\ldots,\widetilde{\mathsf{pk}}_S)}$
00 $\text{cnt} := 0$
01 $\mathcal{S} := \varnothing$
02 **for** $n \in [N]$
03 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$
04 $\quad k_n \xleftarrow{\$} \{0,1\}^\kappa$
05 $\quad(\mathsf{pk}'_n, \mathsf{sk}'_n) := (\mathsf{pk}_n, (\mathsf{sk}_n, k_n))$
06 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}'_1, \ldots, \mathsf{pk}'_N)$
07 **for** $\text{sID}^* \in \mathcal{S}$
08 $\quad$ **if** $\textsc{Fresh}(\text{sID}^*) = $ **false return** 0
09 $\quad$ **if** $\textsc{Valid}(\text{sID}^*) = $ **false return** 0
10 **return** b'

$\underline{\textsc{Session}_{\mathrm{R}}((i,r) \in [N]^2, I)}$
11 cnt ++
12 $\text{sID} := \text{cnt}$
13 $(\text{init}[\text{sID}], \text{resp}[\text{sID}]) := (i, r)$
14 $\text{type}[\text{sID}] := \text{“Re”}$
15 $(\widetilde{\mathsf{pk}}, c_r) := I$
16 **if** $\exists \text{sID}'$ s. t. $\widetilde{\mathsf{pk}} = \widetilde{\mathsf{pk}}_{\text{sID}'}$
17 $\quad(\tilde{c}, \widetilde{K}) \leftarrow \textsc{Encaps}(\text{sID}')$
18 $\quad\widetilde{\mathcal{CK}}_{\text{sID}'} := \widetilde{\mathcal{CK}}_{\text{sID}'} \cup \{(\tilde{c}, \widetilde{K})\}$
19 **else**
20 $\quad(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}_{\mathsf{CPA}}^{\widetilde{\mathsf{H}}_{\text{sID}}}(\widetilde{\mathsf{pk}})$
21 **if** $\exists K'_r$ s. t. $(c_r, K'_r) \in \mathcal{CK}_r$
22 $\quad K_r := K'_r$
23 **else**
24 $\quad K_r := \mathsf{Decaps}_{\mathsf{CCA}}^{\mathsf{H}_r}(\mathsf{sk}_r, c_r)$
25 $\quad\mathcal{D}_r := \mathcal{D}_r \cup \{c_r\}$
26 $c_i \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_i, \mathsf{sk}_i)$
27 $K_i \xleftarrow{\$} \mathcal{K}$
28 $\mathcal{C}_i := \mathcal{C}_i \cup \{(c_i, \bot)\}$
29 $\mathcal{CK}_i := \mathcal{CK}_i \cup \{(c_i, K_i)\}$
30 $\text{context} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
31 $K := \mathsf{H}(\text{context}, K_i, K_r, \widetilde{K})$
32 $R := (\tilde{c}, c_i)$
33 $(I[\text{sID}], R[\text{sID}], \text{sKey}[\text{sID}]) := (I, R, K)$
34 **return** (sID, R)

$\underline{\textsc{Session}_{\mathrm{I}}((i,r) \in [N]^2)}$
35 cnt ++
36 $\text{sID} := \text{cnt}$
37 $(\text{init}[\text{sID}], \text{resp}[\text{sID}], \text{type}[\text{sID}]) := (i, r, \text{“In”})$
38 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) := (\widetilde{\mathsf{pk}}_{\text{sID}}, \bot)$ // $\widetilde{\mathsf{sk}}_{\text{sID}}$ unknown
39 $c_r \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_r, \mathsf{sk}_r)$
40 $K_r \xleftarrow{\$} \mathcal{K}$
41 $\mathcal{C}_r := \mathcal{C}_r \cup \{(c_r, \bot)\}$
42 $\mathcal{CK}_r := \mathcal{CK}_r \cup \{(c_r, K_r)\}$
43 $I := (\widetilde{\mathsf{pk}}, c_r)$
44 $IV \xleftarrow{\$} \{0,1\}^\kappa$
45 $\text{st}' := (\widetilde{\mathsf{pk}}, \bot, c_r, K_r)$
46 $\text{st} := (IV, \bot)$
47 $(I[\text{sID}], \text{state}[\text{sID}], \text{state}'[\text{sID}]) := (I, \text{st}, \text{st}')$
48 **return** (sID, I)

$\underline{\textsc{Der}_{\mathrm{I}}(\text{sID}, R)}$
49 **if** $\text{state}[\text{sID}] = \bot$ **or** $\text{sKey}[\text{sID}] \neq \bot$
50 $\quad$ **return** $\bot$
51 $(i, r) := (\text{init}[\text{sID}], \text{resp}[\text{sID}])$
52 $(\widetilde{\mathsf{pk}}, \cdot, c_r, K_r) := \text{state}'[\text{sID}]$
53 $(\tilde{c}, c_i) := R$
54 **if** $\exists \widetilde{K}'$ s. t. $(\tilde{c}, \widetilde{K}') \in \widetilde{\mathcal{CK}}_{\text{sID}}$
55 $\quad\widetilde{K} := \widetilde{K}'$
56 **else**
57 $\quad\widetilde{\mathsf{sk}} := \textsc{Open}(\text{sID})$
58 $\quad\widetilde{K} := \mathsf{Decaps}_{\mathsf{CPA}}^{\widetilde{\mathsf{H}}_{\text{sID}}}(\widetilde{\mathsf{sk}}, \tilde{c})$
59 **if** $\exists K'_i$ s. t. $(c_i, K'_i) \in \mathcal{CK}_i$
60 $\quad K_i := K'_i$
61 **else**
62 $\quad K_i := \mathsf{Decaps}_{\mathsf{CCA}}^{\mathsf{H}_i}(\mathsf{sk}_i, c_i)$
63 $\quad\mathcal{D}_i := \mathcal{D}_i \cup \{c_i\}$
64 $\text{context} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_i, c_r, \tilde{c})$
65 $K := \mathsf{H}(\text{context}, K_i, K_r, \widetilde{K})$
66 $(R[\text{sID}], \text{sKey}[\text{sID}]) := (R, K)$
67 **return** ε

Fig. 17. Adversaries $\mathcal{C}_b$ against S-NCKE-CPA for the proof of Eqn. (4). $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, \ldots, \mathsf{H}_N, \widetilde{\mathsf{H}}_1, \ldots, \widetilde{\mathsf{H}}_S\}$, where $\textsc{Reveal}$, $\textsc{Corrupt}$, $\textsc{Test}$, H and H_n for $n \in [N]$ are defined as in Figure 14 resp. 15. $\textsc{Rev-State}$, G and $\widetilde{\mathsf{H}}_{\text{sID}}$ for $\text{sID} \in [S]$ are defined in Figure 18. Lines written in blue color highlight how the adversary simulates $G_{2,b}$ and interpolates to $G_{3,b}$.

If $\mathcal{A}$ queries $\textsc{Session}_{\mathrm{R}}$, $\mathcal{C}_b$ checks whether there exists another session sID' with the same $\widetilde{\mathsf{pk}}$ and if it does, $\mathcal{C}_b$ queries its $\textsc{Encaps}$ oracle on sID' in line 17 (Fig. 17). At that point note why we can only embed the challenge in those sessions with a $\widetilde{\mathsf{pk}}$ output by $\textsc{Session}_{\mathrm{I}}$. An active adversary may query $\textsc{Session}_{\mathrm{R}}$ on any $\widetilde{\mathsf{pk}}$ that it has chosen himself. Thus, $\mathcal{C}_b$ might not be able to query the challenge oracle $\textsc{Encaps}$ on that $\widetilde{\mathsf{pk}}$.

As described before, we save challenges relative to sID' in line 18 to restore them later again when $\textsc{Der}_{\mathrm{I}}$ is called in line 55. If the adversary queries $\textsc{Der}_{\mathrm{I}}$ on a new value $\tilde{c}$, $\mathcal{C}_b$ calls $\textsc{Open}$ to obtain $\widetilde{\mathsf{sk}}$ and compute $\widetilde{K}$. Next, $\mathcal{C}_b$ computes K_i from c_i, retrieves K_r from the state and finally calculates the session key K.

Queries to a random oracle $\widetilde{\mathsf{H}}_{\text{sID}}$ are forwarded to the corresponding oracle $\widetilde{\mathsf{H}}'_{\text{sID}}$ whenever a session is of type “In”.

If $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{real}}$ game, it perfectly simulates $G_{2,b}$. Otherwise, if $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{3,b}$. We have

$$\begin{aligned}\left|\Pr[G^{\mathcal{A}}_{3,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{2,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{C}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{C}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{C}_b) \ . \end{aligned} \tag{4}$$

$\underline{\mathsf{G}(k, IV)}$
00 **if** $\exists \widetilde{k}, IV$ s. t. $(k, IV, y) \in \mathcal{G}$ **return** y
01 **if** $\exists i$ s. t. $k = k_i$ **and**
$\exists(\text{sID}, \psi)$ s. t. state[sID] $= (IV, \psi)$
$\wedge$ revState[sID] $=$ **true**
02 $\widetilde{\mathsf{sk}} \coloneqq \textsc{Open}(\text{sID})$
03 state$'$[sID] $\coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r)$
04 $y \coloneqq \psi \oplus$ state$'$[sID]
05 **else**
06 $y \xleftarrow{\$} \{0,1\}^d$
07 $\mathcal{G} \coloneqq \mathcal{G} \cup \{(k, IV, y)\}$
08 **return** y

$\underline{\widetilde{\mathsf{H}}_{\text{sID}}(M)}$ //sID $\in [S]$
09 **if** $\exists h$ s. t. $(M, h) \in \widetilde{\mathcal{H}}_{\text{sID}}$ **return** h
10 **if** type[sID] $=$ "In"
11 $h \leftarrow \widetilde{\mathsf{H}}'_{\text{sID}}(M)$
12 **else**
13 $h \xleftarrow{\$} \{0,1\}^\kappa$
14 $\widetilde{\mathcal{H}}_{\text{sID}} \coloneqq \widetilde{\mathcal{H}}_{\text{sID}} \cup \{(M, h)\}$
15 **return** h

$\underline{\textsc{Rev-State}(\text{sID})}$
16 **if** revState[sID] $=$ **true**
17 **return** state[sID]
18 **if** type[sID] $\neq$ "In" **return** $\bot$
19 revState[sID] $\coloneqq$ **true**
20 $(IV, \bot) \coloneqq$ state[sID]
21 $i \coloneqq$ init[sID]
22 **if** corrupted$[i]$
23 $\widetilde{\mathsf{sk}} \coloneqq \textsc{Open}(\text{sID})$
24 state$'$[sID] $\coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r)$
25 state[sID] $\coloneqq (IV, \mathsf{G}(IV, k_i) \oplus$ state$'$[sID]$)$
26 **elseif** $\exists y$ s. t. $(k_i, IV, y) \in \mathcal{G}$
27 BAD $\coloneqq$ **true**
28 **abort**
29 **else**
30 $\psi \xleftarrow{\$} \{0,1\}^d$
31 state[sID] $\coloneqq (IV, \psi)$
32 **return** state[sID]

Fig. 18. Oracles G, $\widetilde{\mathsf{H}}_{\text{sID}}$ for sID $\in [S]$ and Rev-State for adversaries $\mathcal{C}_b$ against S-NCKE-CPA in Figure 17.

Games $G_{4,b}$. In games $G_{4,b}$, we change the output for K_0^* in the Test oracle to a random key in line 49. Now games $G_{4,0}$ and $G_{4,1}$ are equal as well as games $G_{4,1}$ and $G_{3,1}$, hence

$$\begin{aligned}\left|\Pr[G^{\mathcal{A}}_{3,1} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,0} \Rightarrow 1]\right| &= \left|\Pr[G^{\mathcal{A}}_{4,1} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,0} \Rightarrow 1]\right| \\ &= \left|\Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,0} \Rightarrow 1]\right| \ . \end{aligned} \tag{5}$$

It remains to bound $\left|\Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,0} \Rightarrow 1]\right|$. Therefore, we will now consider the different attacks for IND-wFS-St as described in Table 1. Depending on which queries the adversary makes, each test session must belong to at least one of the attacks or the game will return 0 anyway. For the analysis, we consider the worst case scenario where the adversary queries as much information as it is allowed to obtain.

When referring to a particular test session sID*, we will denote all values used with an asterisk, i. e. context$^* = (\mathsf{pk}_{i^*}, \mathsf{pk}_{r^*}, \widetilde{\mathsf{pk}}^*, c_{i^*}, c_{r^*}, \tilde{c}^*)$ and $IV^*, k_{i^*}, K_{i^*}, K_{r^*}, \widetilde{K}^*$. As we assumed in the beginning that ciphertexts and long-term as well as ephemeral key pairs are all different, it is not possible to recreate a particular session. In particular, this means that there is no partially matching session and attack (0) of Table 1 will return false.

Now the only possibility to learn any test key K_0^* is through random oracle queries. Let QUERY be the event that $(\text{context}, K_i, K_r, \widetilde{K})$ of any test session is queried to H and QUERY* be the event that $(\text{context}^*, K_{i^*}, K_{r^*}, \widetilde{K}^*)$ of a specific test session is queried to H. We have

$$\left|\Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,0} \Rightarrow 1]\right| \leq \Pr[\text{QUERY}] \leq T \cdot \Pr[\text{QUERY}^*] \ ,$$

where the last inequality uses union bound over the number of test sessions T.

We will now focus on the event QUERY* and iterate over the attacks in Table 1. An overview on the argumentation is given in Figure 19.

Attack	**Security relies on ...**
$(1 \vee 2)$, (10)	IV^* unknown $\Rightarrow \widetilde{\mathsf{sk}}^*$ unknown $\Rightarrow \widetilde{K}^*$ unknown
$(7 \vee 8)$, (16)	sk_{i^*} unknown $\Rightarrow K_{i^*}$ unknown
(19)	sk_{r^*} and k_{i^*} unknown $\Rightarrow K_{r^*}$ unknown
(21)	sk_{r^*} and IV^* unknown $\Rightarrow K_{r^*}$ unknown
(24)	sk_{i^*} unknown $\Rightarrow K_{i^*}$ unknown

Fig. 19. Overview of attacks for the proof of Theorem 2.

ATTACK $(1 \vee 2)$, (10). If $(1 \vee 2) \Rightarrow \mathbf{true}$, the test session has a matching session and both long-term secret keys $(\mathsf{sk}_{i^*}, k_{i^*})$ and $(\mathsf{sk}_{r^*}, k_{r^*})$ are revealed. Hence, $\mathcal{A}$ can compute K_{i^*} and K_{r^*}. However, $\mathcal{A}$ is not allowed to query the test session's state or the state of the matching session, depending on the type of the test session. Thus, $\mathcal{A}$ has no information about $\widetilde{\mathsf{sk}}^*$. As there is a matching session for this test session, $\widetilde{\mathsf{pk}}^*$ was generated by SESSION_I, which means that $\widetilde{K}^*$ is chosen uniformly at random and thus independent of $\widetilde{\mathsf{pk}}^*$ and $\tilde{c}^*$.

If $(10) \Rightarrow \mathbf{true}$, the test session has a partially matching and is of type "Re". Here, $\mathcal{A}$ is not allowed to query the state of the partially matching session. Except for that, everything remains the same as explained above. Hence,

$$\Pr[\text{QUERY}^* \mid (1 \vee 2) \Rightarrow \mathbf{true}] = \Pr[\text{QUERY}^* \mid (10) \Rightarrow \mathbf{true}] \leq \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

ATTACK $(7 \vee 8)$, (16). If $(7 \vee 8) \Rightarrow \mathbf{true}$, the test session has a matching session and the state (IV^*, ψ^*) is revealed. Furthermore, $\mathcal{A}$ can obtain $(\mathsf{sk}_{r^*}, k_{r^*})$ and thus K_{r^*}. As secret key sk_{i^*} is unknown to $\mathcal{A}$, K_{i^*} which is chosen uniformly at random from the key space of $\mathsf{KEM}_\mathsf{CCA}$ is also unknown to $\mathcal{A}$. $\widetilde{K}^*$ is also chosen uniformly at random and unknown as long as $\mathcal{A}$ does not obtain $\widetilde{\mathsf{sk}}^*$ through queries to G or guesses $\widetilde{K}^*$ correctly. However, to bound event QUERY^*, we will only make use of the fact that K_{i^*} is a uniformly random key.

If (16) $\Rightarrow \mathbf{true}$, the test session has a partially matching session and is of type "Re". Here, $\mathcal{A}$ can reveal the state of the partially matching session (IV, ψ). The rest remains the same. It follows that

$$\Pr[\text{QUERY}^* \mid (7 \vee 8) \Rightarrow \mathbf{true}] = \Pr[\text{QUERY}^* \mid (16) \Rightarrow \mathbf{true}] \leq \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

ATTACK (19). If $(19) \Rightarrow \mathbf{true}$, the test session has no matching session and the type of this test session is "In". $\mathcal{A}$ is allowed to obtain the initiator's state (IV^*, ψ^*) and can choose $(\tilde{c}^*, \widetilde{K}^*)$ and (c_{i^*}, K_{i^*}) itself. K_{r^*} is unknown to $\mathcal{A}$, unless it obtains it through queries to G. Therefore, $\mathcal{A}$ can either guess k_{i^*} and query G or it can query H directly. Hence,

$$\Pr[\text{QUERY}^* \mid (19) \Rightarrow \mathbf{true}] \leq \frac{q_\mathsf{G}}{2^\kappa} + \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

ATTACK (21). If $(21) \Rightarrow \mathbf{true}$, the test session has no matching session and the type of this test session is "In". $\mathcal{A}$ is allowed to obtain the initiator's long-term secret key $(\mathsf{sk}_{i^*}, k_{i^*})$ and can choose $(\tilde{c}^*, \widetilde{K}^*)$ and (c_{i^*}, K_{i^*}) itself. However, K_r^* is unknown to $\mathcal{A}$ and as it is chosen uniformly at random from the key space of $\mathsf{KEM}_\mathsf{CCA}$, $(\text{context}^*, K_{i^*}, K_{r^*}, \widetilde{K}^*)$ is queried to H with probability at most $q_\mathsf{H}/|\mathcal{K}|$. This yields

$$\Pr[\text{QUERY}^* \mid (21) \Rightarrow \mathbf{true}] \leq \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

ATTACK (24). If $(24) \Rightarrow \mathbf{true}$, the test session has no matching session and the type of the test session is "Re", which means that $\mathcal{A}$ can reveal the responder's long-term secret key $(\mathsf{sk}_{r^*}, k_{r^*})$. Here, $\mathcal{A}$ can choose $(\widetilde{\mathsf{pk}}^*, \widetilde{\mathsf{sk}}^*)$ and (c_{r^*}, K_{r^*}) itself and thus is able to compute $\widetilde{K}^*$. As $(\mathsf{sk}_{i^*}, k_{i^*})$ is unknown, so is K_{i^*} and we have

$$\Pr[\text{QUERY}^* \mid (24) \Rightarrow \mathbf{true}] \leq \frac{q_\mathsf{H}}{|\mathcal{K}|} \ .$$

Taking the maximum over the conditional probabilities, it follows that

$$\left|\Pr[G_{4,0}^\mathcal{A} \Rightarrow 1] - \Pr[G_{3,0}^\mathcal{A} \Rightarrow 1]\right| \leq T \cdot \Pr[\text{QUERY}^*] \leq T \cdot \left(\frac{q_\mathsf{G}}{2^\kappa} + \frac{q_\mathsf{H}}{|\mathcal{K}|}\right) \ . \tag{6}$$

Finally, folding both adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$ into one adversary $\mathcal{B}$ and $\mathcal{C}_0$ and $\mathcal{C}_1$ into one adversary $\mathcal{C}$ yields

$$\begin{aligned}
\text{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_\mathsf{CCA},\mathsf{Sim}_\mathsf{CCA}}(\mathcal{B}_0) + \text{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_\mathsf{CCA},\mathsf{Sim}_\mathsf{CCA}}(\mathcal{B}_1) &= 2 \cdot \text{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_\mathsf{CCA},\mathsf{Sim}_\mathsf{CCA}}(\mathcal{B}) \text{ and} \\
\text{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_\mathsf{CPA},\mathsf{Sim}_\mathsf{CPA}}(\mathcal{C}_0) + \text{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_\mathsf{CPA},\mathsf{Sim}_\mathsf{CPA}}(\mathcal{C}_1) &= 2 \cdot \text{Adv}^{S\text{-NCKE-CPA}}_{\mathsf{KEM}_\mathsf{CPA},\mathsf{Sim}_\mathsf{CPA}}(\mathcal{C}) \ .
\end{aligned}$$

The proof of Theorem 2 follows by collecting the probabilities from Eqns. (2)-(6). □

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

6 AKE with Full Forward Security

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

6.1 Digital Signatures

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

In Figure 20, we define the security game N user Strong UnForgeability under Chosen Message Attacks with corruptions (N-SUF-CMA). The definition is similar to the one given in [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

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

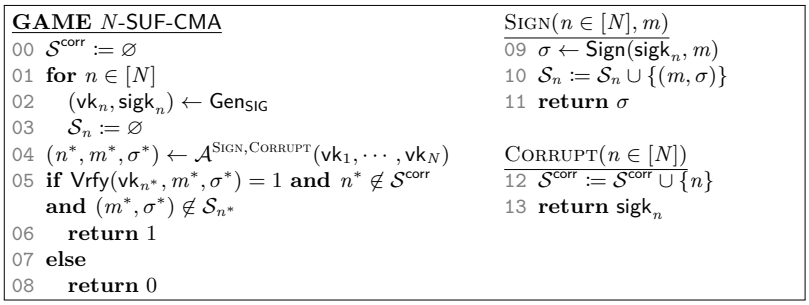

Fig. 20. Game N-SUF-CMA for SIG.

6.2 Transformation using NCKE and a Signature Scheme

From two key encapsulation mechanisms $\mathsf{KEM}_{\mathsf{CPA}} = (\mathsf{Gen}_{\mathsf{CPA}}, \mathsf{Encaps}_{\mathsf{CPA}}, \mathsf{Decaps}_{\mathsf{CPA}})$ and $\mathsf{KEM}_{\mathsf{CCA}} = (\mathsf{Gen}_{\mathsf{CCA}}, \mathsf{Encaps}_{\mathsf{CCA}}, \mathsf{Decaps}_{\mathsf{CCA}})$ with key space $\mathcal{K}$ and a digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}_{\mathsf{SIG}}, \mathsf{Sign}, \mathsf{Vrfy})$, we construct a two-message authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{FS}} = (\mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathrm{I}}, \mathsf{Der}_{\mathrm{R}}, \mathsf{Der}_{\mathrm{I}})$ with key space $\mathcal{K}$ as shown in Figures 21 and 22. Each party has a key pair $(\mathsf{vk}, \mathsf{sigk})$ for SIG, a key pair $(\mathsf{pk}, \mathsf{sk})$ for $\mathsf{KEM}_{\mathsf{CCA}}$ and a symmetric key k to encrypt the secret state information which has to be stored by the initiating party (cf. Section 5). The protocol uses additional cryptographic hash functions $\mathsf{F} : \{0,1\}^* \to \{0,1\}^{\kappa}$ to compute value π and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ to output the session key.

The initiating party computes an ephemeral key pair for $\mathsf{KEM}_{\mathsf{CPA}}$, runs the $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithm on the intended receiver's public key pk_r to obtain a ciphertext c_r and a key K_r and signs both the

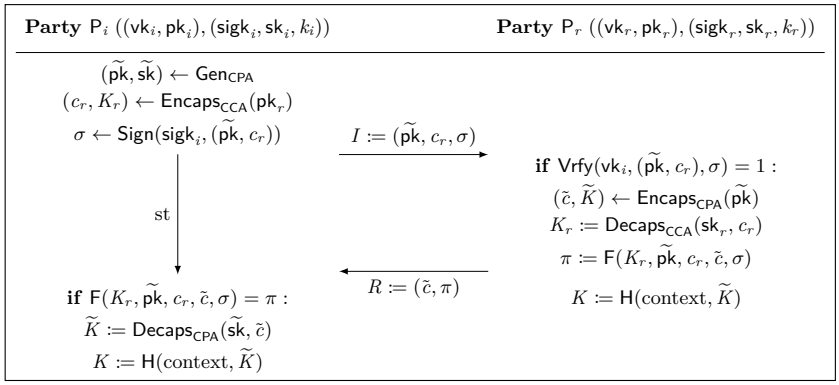

Fig. 21. Visualization: Running $\mathsf{AKE}_{\mathsf{FS}}$ between two parties, where K is the resulting session key and context $:= (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$

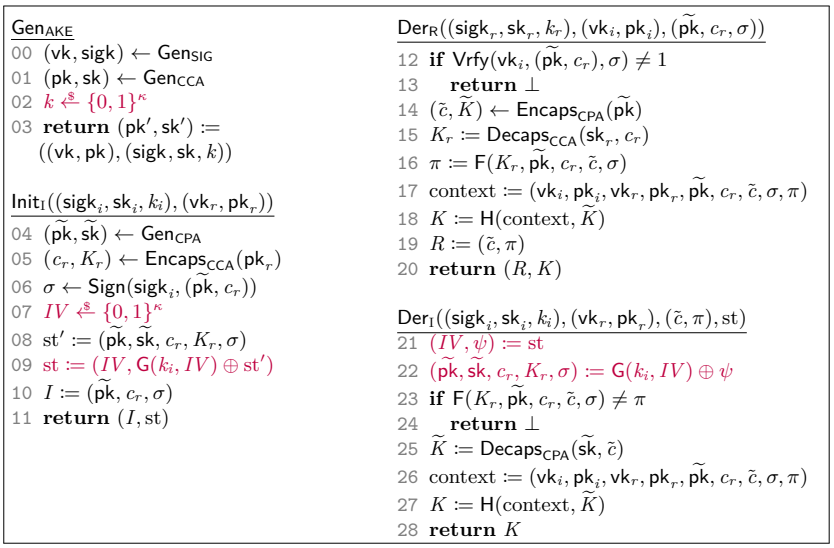

Fig. 22. Authenticated key exchange protocol $\mathsf{AKE}_{\mathsf{FS}}$ from $\mathsf{KEM}_{\mathsf{CPA}}$, $\mathsf{KEM}_{\mathsf{CCA}}$ and SIG. Lines written in purple color are only used to encrypt the state.

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 sk_r. It then tags the received message together with $\tilde{c}$ and K_r by evaluating hash function F and sends the output together with $\tilde{c}$ to the initiator. The initiator retrieves K_r from the secret state and also evaluates F. If the output is the same, it computes $\widetilde{K}$ using the ephemeral secret key. The session key is computed evaluating hash function H on all public context and key $\widetilde{K}$.

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

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

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

Proof. Let $\mathcal{A}$ be an adversary against IND-FS-St security of $\mathsf{AKE}_{\mathsf{FS}}$, where N is the number of parties, S is the maximum number of sessions that $\mathcal{A}$ establishes and T is the maximum number of test sessions. Consider the sequence of games in Figures 23 and 24.

GAMES $G_{0,b}$-$G_{5,b}$

00 $\mathrm{cnt} \coloneqq 0$
01 $\mathcal{S} \coloneqq \varnothing$
02 **for** $n \in [N]$
03 $\quad(\mathsf{vk}_n, \mathsf{sigk}_n) \leftarrow \mathsf{Gen}_{\mathsf{SIG}}$
04 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_{\mathsf{CCA}}$ // $G_{0\text{-}1}$
05 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{SimGen}_{\mathsf{CCA}}$ // $G_{2\text{-}5}$
06 $\quad k_n \xleftarrow{\$} \{0,1\}^\kappa$
07 $\quad(\mathsf{pk}'_n, \mathsf{sk}'_n) \coloneqq ((\mathsf{vk}_n, \mathsf{pk}_n), (\mathsf{sigk}_n, \mathsf{sk}_n, k_n))$
08 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{pk}'_1, ..., \mathsf{pk}'_N)$
09 **for** $\mathrm{sID}^* \in \mathcal{S}$
10 $\quad$ **if** $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$ **return** 0
11 $\quad$ **if** $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$ **return** 0
12 **return** b'

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

13 cnt ++
14 $\mathrm{sID} \coloneqq \mathrm{cnt}$
15 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) \coloneqq (i, r)$
16 $\mathrm{type}[\mathrm{sID}] \coloneqq$ "Re"
17 $\mathrm{peerCorrupted}[\mathrm{sID}] \coloneqq \mathrm{corrupted}[i]$
18 $(\widetilde{\mathsf{pk}}, c_r, \sigma) \coloneqq I$
19 **if** $\mathsf{Vrfy}(\mathsf{vk}_i, (\widetilde{\mathsf{pk}}, c_r), \sigma) \neq 1$
20 $\quad$ **return** $\bot$
21 **if** $\mathrm{peerCorrupted}[\mathrm{sID}] = \mathbf{false}$ **and** $\nexists \mathrm{sID}'$ s. t. $(\mathrm{init}[\mathrm{sID}'], \mathrm{type}[\mathrm{sID}'], I[\mathrm{sID}']) = (i, \text{"In"}, I)$ // $G_{1\text{-}5}$
22 $\quad\mathsf{BREAK}_{\mathsf{SIG}} \coloneqq \mathbf{true}$ // $G_{1\text{-}5}$
23 $\quad$ **abort** // $G_{1\text{-}5}$
24 $(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$ // $G_{0\text{-}3}$
25 **if** $\exists \mathrm{sID}'$ s. t. $\mathrm{state}'[\mathrm{sID}'] = (\widetilde{\mathsf{pk}}, \cdot, \cdot, \cdot, \cdot)$ // $G_{4\text{-}5}$
26 $\quad(\cdot, \widetilde{\mathsf{sk}}, \cdot, \cdot, \cdot) \coloneqq \mathrm{state}'[\mathrm{sID}']$ // $G_{4\text{-}5}$
27 $\quad\tilde{c} \leftarrow \mathsf{SimEncaps}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ // $G_{4\text{-}5}$
28 $\quad\widetilde{K} \xleftarrow{\$} \mathcal{K}$ // $G_{4\text{-}5}$
29 $\quad\widetilde{\mathcal{C}}_{\mathrm{sID}'} \coloneqq \widetilde{\mathcal{C}}_{\mathrm{sID}'} \cup \{(\tilde{c}, \bot)\}$ // $G_{4\text{-}5}$
30 $\quad\widetilde{\mathcal{CK}}_{\mathrm{sID}'} \coloneqq \widetilde{\mathcal{CK}}_{\mathrm{sID}'} \cup \{(\tilde{c}, \widetilde{K})\}$ // $G_{4\text{-}5}$
31 **else** // $G_{4\text{-}5}$
32 $\quad(\tilde{c}, \widetilde{K}) \leftarrow \mathsf{Encaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{pk}})$ // $G_{4\text{-}5}$
33 $K_r \coloneqq \mathsf{Decaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$ // $G_{0\text{-}1}$
34 **if** $\exists K'_r$ s. t. $(c_r, K'_r) \in \mathcal{CK}_r$ // $G_{2\text{-}5}$
35 $\quad K_r \coloneqq K'_r$ // $G_{2\text{-}5}$
36 **else** // $G_{2\text{-}5}$
37 $\quad K_r \coloneqq \mathsf{Decaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{sk}_r, c_r)$ // $G_{2\text{-}5}$
38 $\quad\mathcal{D}_r \coloneqq \mathcal{D}_r \cup \{c_r\}$ // $G_{2\text{-}5}$
39 $\pi \coloneqq \mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma)$
40 $\mathrm{context} \coloneqq (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$
41 $K \coloneqq \mathsf{H}(\mathrm{context}, \widetilde{K})$
42 $R \coloneqq (\tilde{c}, \pi)$
43 $(I[\mathrm{sID}], R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) \coloneqq (I, R, K)$
44 **return** (sID, R)

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

45 cnt ++
46 $\mathrm{sID} \coloneqq \mathrm{cnt}$
47 $(\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}]) \coloneqq (i, r)$
48 $\mathrm{type}[\mathrm{sID}] \coloneqq$ "In"
49 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_{\mathsf{CPA}}$ // $G_{0\text{-}3}$
50 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{SimGen}_{\mathsf{CPA}}$ // $G_{4\text{-}5}$
51 $(c_r, K_r) \leftarrow \mathsf{Encaps}^{\mathsf{H}_r}_{\mathsf{CCA}}(\mathsf{pk}_r)$ // $G_{0\text{-}1}$
52 $c_r \leftarrow \mathsf{SimEncaps}_{\mathsf{CCA}}(\mathsf{pk}_r, \mathsf{sk}_r)$ // $G_{2\text{-}5}$
53 $K_r \xleftarrow{\$} \mathcal{K}$ // $G_{2\text{-}5}$
54 $\mathcal{C}_r \coloneqq \mathcal{C}_r \cup \{(c_r, \bot)\}$
55 $\mathcal{CK}_r \coloneqq \mathcal{CK}_r \cup \{(c_r, K_r)\}$
56 $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_i, (\widetilde{\mathsf{pk}}, c_r))$
57 $I \coloneqq (\widetilde{\mathsf{pk}}, c_r, \sigma)$
58 $IV \xleftarrow{\$} \{0,1\}^\kappa$
59 $\mathrm{st}' \coloneqq (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r, \sigma)$
60 $\mathrm{st} \coloneqq (IV, \mathsf{G}(k_i, IV) \oplus \mathrm{st}')$ // $G_{0\text{-}2}$
61 $\mathrm{st} \coloneqq (IV, \bot)$ // $G_{3\text{-}5}$
62 $(I[\mathrm{sID}], \mathrm{state}[\mathrm{sID}]) \coloneqq (I, \mathrm{st})$
63 $\mathrm{state}'[\mathrm{sID}] \coloneqq \mathrm{st}'$
64 **return** (sID, I)

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID}, R)$

65 **if** $\mathrm{state}[\mathrm{sID}] = \bot$ **or** $\mathrm{sKey}[\mathrm{sID}] \neq \bot$
66 $\quad$ **return** $\bot$
67 $(i, r) \coloneqq (\mathrm{init}[\mathrm{sID}], \mathrm{resp}[\mathrm{sID}])$
68 $\mathrm{peerCorrupted}[\mathrm{sID}] \coloneqq \mathrm{corrupted}[r]$
69 $(\tilde{c}, \pi) \coloneqq R$
70 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, c_r, K_r, \sigma) \coloneqq \mathrm{state}'[\mathrm{sID}]$
71 **if** $\mathsf{F}(K_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma) \neq \pi$
72 $\quad$ **return** $\bot$
73 $\widetilde{K} \coloneqq \mathsf{Decaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$ // $G_{0\text{-}3}$
74 **if** $\exists \widetilde{K}'$ s. t. $(\tilde{c}, \widetilde{K}') \in \widetilde{\mathcal{CK}}_{\mathrm{sID}}$ // $G_{4\text{-}5}$
75 $\quad\widetilde{K} \coloneqq \widetilde{K}'$ // $G_{4\text{-}5}$
76 **else** // $G_{4\text{-}5}$
77 $\quad\widetilde{K} \coloneqq \mathsf{Decaps}^{\widetilde{\mathsf{H}}_{\mathrm{sID}}}_{\mathsf{CPA}}(\widetilde{\mathsf{sk}}, \tilde{c})$ // $G_{4\text{-}5}$
78 $\mathrm{context} \coloneqq (\mathsf{vk}_i, \mathsf{pk}_i, \mathsf{vk}_r, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, c_r, \tilde{c}, \sigma, \pi)$
79 $K \coloneqq \mathsf{H}(\mathrm{context}, \widetilde{K})$
80 $(R[\mathrm{sID}], \mathrm{sKey}[\mathrm{sID}]) \coloneqq (R, K)$
81 **return** ε

$\mathsf{H}(x)$

82 **if** $\exists K$ s. t. $(x, K) \in \mathcal{H}$
83 $\quad$ **return** K
84 $K \xleftarrow{\$} \mathcal{K}$
85 $\mathcal{H} \coloneqq \mathcal{H} \cup \{(x, K)\}$
86 **return** K

Fig. 23. Games $G_{0,b}$-$G_{5,b}$ for the proof of Theorem 3. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, ..., \mathsf{H}_N, \widetilde{\mathsf{H}}_1, ..., \widetilde{\mathsf{H}}_S\}$, where Corrupt and Reveal are defined as in the original IND-FS-St game (Fig. 10) and oracles Rev-State, F, G, H_n for $n \in [N]$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ for $\mathrm{sID} \in [S]$ and Test are defined in Figure 24.

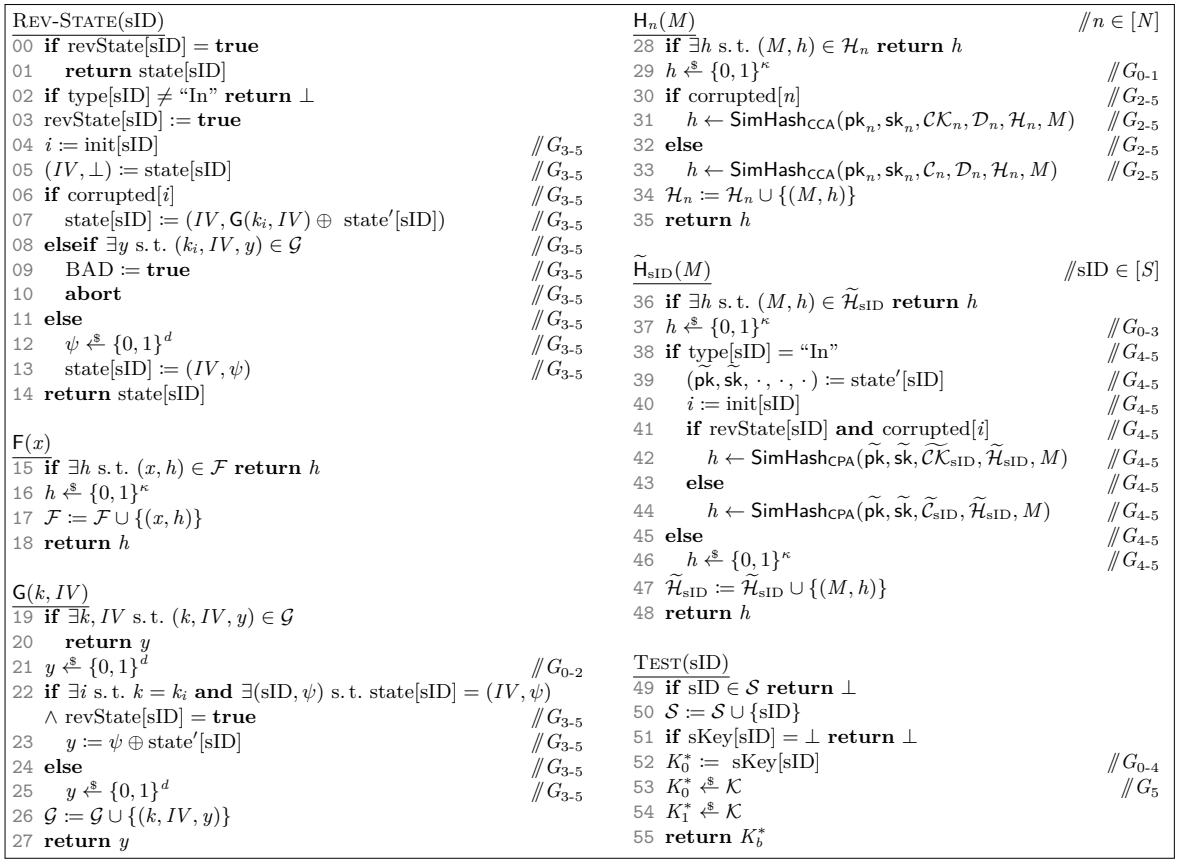

Fig. 24. Oracles REV-STATE, F, G, H_n for $n \in [N]$, $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ and TEST for $\mathrm{sID} \in [S]$ for games $G_{0,b}$-$G_{5,b}$ in Fig. 23.

GAMES $G_{0,b}$. These are the original $\mathsf{IND\text{-}FS\text{-}St}_b$ games. Similar to Equation (2) in the proof of Theorem 2, we assume all key pairs, N long-term keys generated by $\mathsf{Gen}_{\mathsf{SIG}}$ and $\mathsf{Gen}_{\mathsf{CCA}}$ as well as ephemeral keys (at most S) generated by $\mathsf{Gen}_{\mathsf{CPA}}$, and all ciphertexts (at most S) output by the $\mathsf{Encaps}_{\mathsf{CPA}}$ and $\mathsf{Encaps}_{\mathsf{CCA}}$ algorithms to be distinct. We also assume that values k_n, $n \in [N]$, and IV (at most S) are distinct. This yields

$$\begin{aligned}\Big|\Pr[\mathsf{IND\text{-}FS\text{-}St}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{IND\text{-}FS\text{-}St}_0^{\mathcal{A}} \Rightarrow 1]\Big| \leq \big|\Pr[G_{0,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{0,0}^{\mathcal{A}} \Rightarrow 1]\big| \\ + N^2\left(2^{-\mu_{\mathsf{SIG}}} + 2^{-\mu_{\mathsf{CCA}}} + 2^{-\kappa}\right) + S^2\left(2^{-\mu_{\mathsf{CPA}}} + 2^{-\gamma_{\mathsf{CCA}}} + 2^{-\gamma_{\mathsf{CPA}}} + 2^{-\kappa}\right)\ , \quad (7)\end{aligned}$$

where μ_{SIG}, μ_{CPA}, μ_{CCA} are collision probabilities of the key generation algorithms $\mathsf{Gen}_{\mathsf{SIG}}$, $\mathsf{Gen}_{\mathsf{CPA}}$ and $\mathsf{Gen}_{\mathsf{CCA}}$ and γ_{CPA}, γ_{CCA} are the spreadness parameters of $\mathsf{Encaps}_{\mathsf{CPA}}$ and $\mathsf{Encaps}_{\mathsf{CCA}}$.

GAMES $G_{1,b}$. In games $G_{1,b}$, we raise flag $\mathsf{BREAK}_{\mathsf{SIG}}$ in line 22 (Fig. 24) and abort when $\mathcal{A}$ has not queried CORRUPT to obtain sigk_i yet and has not only forwarded a message and a signature output by $\mathrm{SESSION}_{\mathrm{I}}$. Note that if the game aborts, the signature must be valid because this is checked before in line 19. Due to the difference lemma [35],

$$\big|\Pr[G_{1,b}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{0,b}^{\mathcal{A}} \Rightarrow 1]\big| \leq \Pr[\mathsf{BREAK}_{\mathsf{SIG}}]\ . \quad (8)$$

To bound $\Pr[\mathsf{BREAK}_{\mathsf{SIG}}]$, we construct adversaries $\mathcal{B}_b$ for $b \in \{0,1\}$ against N-$\mathsf{SUF\text{-}CMA}$ security of SIG in Figure 25.

$\mathcal{B}_b$ inputs N verification keys $(\mathsf{vk}_1, ..., \mathsf{vk}_N)$ and has access to signing oracle SIGN and corruption oracle CORRUPT$'$. It then generates N key pairs for $\mathsf{KEM}_{\mathsf{CCA}}$ and N symmetric keys k_n which are part of the long-term secret key. It forwards the public keys to adversary $\mathcal{A}$. Whenever $\mathcal{A}$ queries $\mathrm{SESSION}_{\mathrm{I}}$ on a pair (i, r), $\mathcal{B}_b$ queries SIGN on user i in line 37 to obtain a signature σ to message $(\widetilde{\mathsf{pk}}, c_r)$. It then outputs $(\widetilde{\mathsf{pk}}, c_r, \sigma)$ to $\mathcal{A}$.

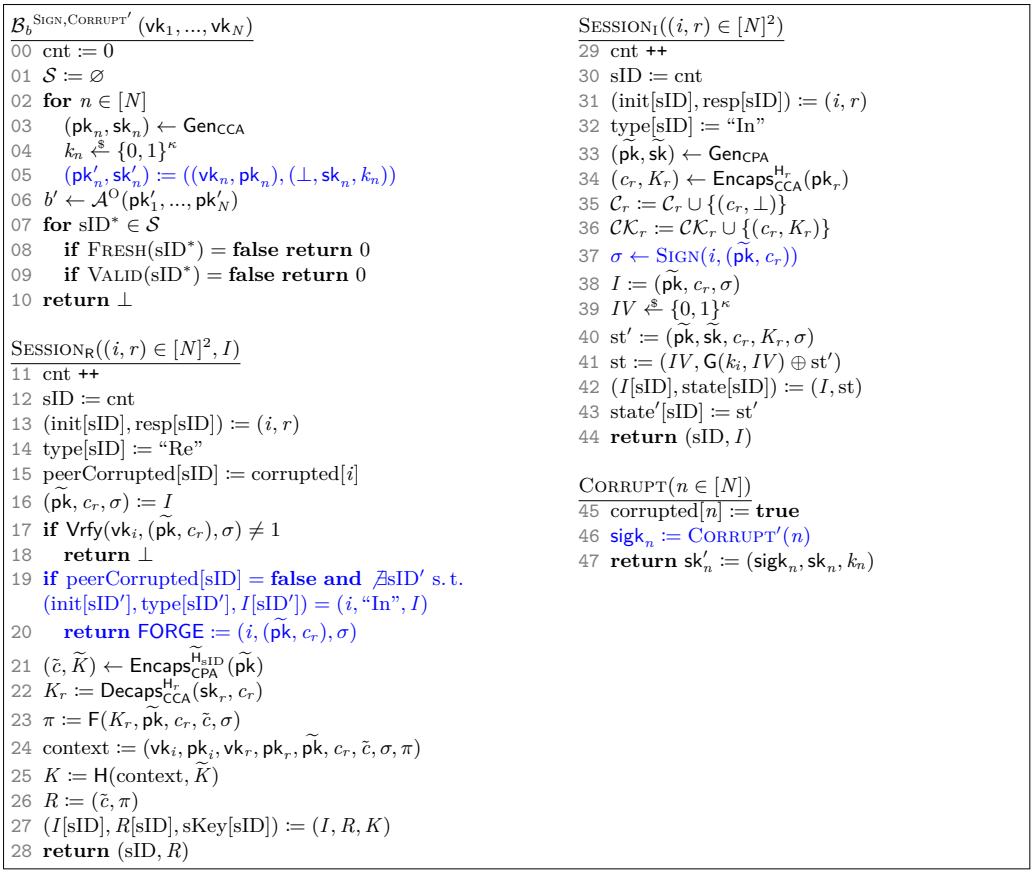

Fig. 25. Adversaries $\mathcal{B}_b$ against N-SUF-CMA for the proof of Eqn. (9). $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{SESSION}_{\text{I}}, \text{SESSION}_{\text{R}}, \text{DER}_{\text{I}}, \text{REVEAL}, \text{REV-STATE}, \text{CORRUPT}, \text{TEST}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, ..., \mathsf{H}_N, \widetilde{\mathsf{H}}_1, ..., \widetilde{\mathsf{H}}_S\}$, where DER_{I}, REVEAL, REV-STATE, TEST, $\mathsf{F}, \mathsf{G}, \mathsf{H}$, H_n for $n \in [N]$ and $\widetilde{\mathsf{H}}_{\text{sID}}$ for $\text{sID} \in [S]$ are defined as in games $G_{0,b}$ in Figure 23 resp. 24. Lines written in blue color highlight how the adversary simulates $G_{0,b}$ and how event $\mathsf{BREAK}_{\mathsf{SIG}}$ leads to a forgery.

To answer a CORRUPT query on user n, $\mathcal{B}_b$ queries its own oracle $\text{CORRUPT}'$ to obtain signing key sigk_n in line 46 and outputs secret keys sigk_n, sk_n and k_n to $\mathcal{A}$.

Recall that flag $\mathsf{BREAK}_{\mathsf{SIG}}$ is raised when $\mathcal{A}$ has not queried CORRUPT to obtain sigk_i, but computes a valid signature which was not output by $\text{SESSION}_{\text{I}}$. If $\mathcal{A}$ queries $\text{SESSION}_{\text{R}}$ on pair (i,r) and $I = (\widetilde{\mathsf{pk}}, c_r, \sigma)$ such that this is the case, $\mathcal{B}_b$ wins the N-SUF-CMA game by returning $(i, (\widetilde{\mathsf{pk}}, c_r), \sigma)$. It follows that

$$\Pr[\mathsf{BREAK}_{\mathsf{SIG}}] = \mathrm{Adv}_{\mathsf{SIG}}^{N\text{-SUF-CMA}}(\mathcal{B}_b) \ . \tag{9}$$

GAMES $G_{2,b}$. In games $G_{2,b}$, we use the $\mathsf{SimGen}_{\mathsf{CCA}}$ algorithm to generate long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$ in line 05 (Fig. 23). Next, $\text{SESSION}_{\text{I}}$ uses the $\mathsf{SimEncaps}_{\mathsf{CCA}}$ algorithm to compute ciphertext c_r in line 52 and draws a random key K_r in line 50. K_r is then retrieved in line 35 (Fig. 24) when the same c_r is issued to $\text{SESSION}_{\text{R}}$. Furthermore, the $\mathsf{SimHash}_{\mathsf{CCA}}$ algorithm is used in all random oracles H_n, where $n \in [N]$. In case party n is corrupted, i. e. sk_n is known to the adversary $\mathcal{A}$, we call $\mathsf{SimHash}_{\mathsf{CCA}}$ with set $\mathcal{CK}_n$, otherwise with set $\mathcal{C}_n$.

For $b \in \{0,1\}$, we construct adversaries $\mathcal{C}_b$ against N-NCKE-CCA security of $\mathsf{KEM}_{\mathsf{CCA}}$ in Figure 26, similar to adversaries $\mathcal{B}_b$ in Figure 16, only that here we have only ciphertexts generated by initiating sessions and that signatures are simulated as well. If $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CCA}_{\mathsf{real}}$ game, it perfectly simulates $G_{1,b}$. Otherwise, if $\mathcal{C}_b$ is in the $\mathsf{NCKE\text{-}CCA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{2,b}$. We have

$$\begin{aligned} \left|\Pr[G_{2,b}^{\mathcal{A}} \Rightarrow 1] - \Pr[G_{1,b}^{\mathcal{A}} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CCA}_{\mathsf{sim}}^{\mathcal{C}_b} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CCA}_{\mathsf{real}}^{\mathcal{C}_b} \Rightarrow 1]\right| \\ &= \mathrm{Adv}_{\mathsf{KEM,Sim}}^{N\text{-NCKE-CCA}}(\mathcal{C}_b) \end{aligned} \tag{10}$$

Fig. 26. Adversaries $\mathcal{C}_b$ against N-NCKE-CCA for the proof of Eqn. (10). $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{SESSION}_{\text{I}}, \text{SESSION}_{\text{R}}, \text{DER}_{\text{I}}, \text{REVEAL}, \text{REV-STATE}, \text{CORRUPT}, \text{TEST}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, \ldots, \mathsf{H}_N, \widetilde{\mathsf{H}}_1, \ldots, \widetilde{\mathsf{H}}_S\}$, where DER_{I}, REVEAL, REV-STATE, TEST, $\mathsf{F}, \mathsf{G}, \mathsf{H}$ and $\widetilde{\mathsf{H}}_{\text{sID}}$ for $\text{sID} \in [S]$ are defined as in games $G_{1,b}$ in Figure 23 resp. 24. Lines written in blue color highlight how the adversary simulates $G_{1,b}$ and interpolates to $G_{2,b}$.

We make the same preparations as in the proof of Theorem 2, before switching to the algorithms of the simulator $\mathsf{Sim}_{\mathsf{CPA}} = (\mathsf{SimGen}_{\mathsf{CPA}}, \mathsf{SimEncaps}_{\mathsf{CPA}}, \mathsf{SimHash}_{\mathsf{CPA}})$ in games $G_{4,b}$. In particular, we introduce an intermediate game which delays the computation and encryption of the state.

GAMES $G_{3,b}$. We move the encryption of the state to the REV-STATE oracle and choose only IV when $\text{SESSION}_{\text{I}}$ is called. Then, when REV-STATE is queried and the initiator i is corrupted, we honestly compute the state in line 07 (Fig. 23). If the adversary already made a query to G, where (k_i, IV) are as in the corresponding session, we raise flag BAD in line 09 and abort in line 10. Otherwise, we choose a random string ψ in line 12 and patch the random oracle G for the case that $\mathcal{A}$ issues a query with the correct symmetric key k_i and IV. If BAD is not raised, the adversary's view does not change. We have

$$\left|\Pr[G_{3,b} \Rightarrow 1] - \Pr[G_{2,b} \Rightarrow 1]\right| \leq \Pr[\text{BAD}] \leq S \cdot \frac{q_{\mathsf{G}}}{2^{2\kappa}} \ .$$

GAMES $G_{4,b}$. Following the previous proofs, we now use the $\mathsf{SimGen}_{\mathsf{CPA}}$ algorithm to generate ephemeral key pairs $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$, the $\mathsf{SimEncaps}_{\mathsf{CPA}}$ algorithm to compute ciphertext $\tilde{c}$ and choose a uniformly random key $\widetilde{K}$ whenever the ephemeral public key $\widetilde{\mathsf{pk}}$ was output by $\text{SESSION}_{\text{I}}$. Random oracles $\widetilde{\mathsf{H}}_{\text{sID}}$ will then use the $\mathsf{SimHash}_{\mathsf{CPA}}$ algorithm.

In Figure 27, we construct adversaries $\mathcal{D}_b$ for $b \in \{0,1\}$ against S-NCKE-CPA which are similar to adversaries $\mathcal{C}_b$ defined in Figure 17. Here, $\mathcal{D}_b$ generates N key pairs with $\mathsf{Gen}_{\mathsf{SIG}}$ and simulates the signatures.

If $\mathcal{D}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{real}}$ game, it perfectly simulates $G_{3,b}$. Otherwise, if $\mathcal{D}_b$ is in the $\mathsf{NCKE\text{-}CPA}_{\mathsf{sim}}$ game, it perfectly simulates $G_{4,b}$. We have

$$\begin{aligned}\left|\Pr[G^{\mathcal{A}}_{4,b} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{3,b} \Rightarrow 1]\right| &= \left|\Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{D}_b}_{\mathsf{sim}} \Rightarrow 1] - \Pr[\mathsf{NCKE\text{-}CPA}^{\mathcal{D}_b}_{\mathsf{real}} \Rightarrow 1]\right| \\ &= \mathrm{Adv}^{S\text{-}\mathsf{NCKE\text{-}CPA}}_{\mathsf{KEM}_{\mathsf{CPA}},\mathsf{Sim}_{\mathsf{CPA}}}(\mathcal{D}_b) \ . \end{aligned} \tag{11}$$

Games $G_{5,b}$. In games $G_{5,b}$, we change the output for K^*_0 in the Test oracle to a random key in line 53 (Fig. 23). Now games $G_{5,0}$ and $G_{5,1}$ are equal as well as games $G_{5,1}$ and $G_{4,1}$, hence

$$\begin{aligned}\left|\Pr[G^{\mathcal{A}}_{4,1} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| &= \left|\Pr[G^{\mathcal{A}}_{5,1} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| \\ &= \left|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| \ . \end{aligned} \tag{12}$$

It remains to bound $|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]|$. Therefore, we will now consider the different attacks for $\mathsf{IND\text{-}FS\text{-}St}$ as described in Table 2. Depending on which queries the adversary makes, each test session must belong to at least one of the attacks or the game will return 0 anyway.

Again, we will assume that the adversary queries as much information as possible. Variables of a particular test session sID^* are denoted by $\mathrm{context}^* = (\mathsf{vk}_{i^*}, \mathsf{pk}_{i^*}, \mathsf{vk}_{r^*}, \mathsf{pk}_{r^*}, \widetilde{\mathsf{pk}}^*, c_{r^*}, \tilde{c}^*, \sigma^*, \pi^*)$ and $IV^*, k_{i^*}, \widetilde{K}^*$. As we assumed in the beginning that ciphertexts and long-term as well as ephemeral key pairs are all different, it is not possible to recreate a particular session. In particular, this means that there is no partially matching session and row (0) will return false.

Now the only possibility to learn any test key K^*_0 is through random oracle queries. Let QUERY be the event that $(\mathrm{context}, \widetilde{K})$ of any test session is queried to H and QUERY^* be the event that $(\mathrm{context}^*, \widetilde{K}^*)$ of a specific test session is queried to H. We have

$$\left|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| \leq \Pr[\mathrm{QUERY}] \ .$$

Union bound over the maximum number of test sessions T yields

$$\Pr[\mathrm{QUERY}] \leq T \cdot \Pr[\mathrm{QUERY}^*] \ .$$

We will now focus on the event QUERY^* and iterate over the attacks in Table 2. An overview is given in Figure 28.

Attack $(1 \vee 2)$, (10). If $(1 \vee 2) \Rightarrow \mathbf{true}$, the test session has a matching session and both long-term secret keys $(\mathsf{sigk}_{i^*}, \mathsf{sk}_{i^*}, k_{i^*})$ and $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$ are revealed. However, $\mathcal{A}$ is not allowed to query the test session's state or the state of the matching session, depending on the type of the test session. Thus, $\mathcal{A}$ has no information about $\widetilde{\mathsf{sk}}^*$. As there is a matching session for this test session, $\widetilde{\mathsf{pk}}^*$ was generated by $\textsc{Session}_{\mathrm{I}}$, which means that $\widetilde{K}^*$ is chosen uniformly at random and thus independent of $\widetilde{\mathsf{pk}}^*$ and $\tilde{c}^*$. Hence, the probability that $\mathcal{A}$ queries H on $(\mathrm{context}^*, \widetilde{K}^*)$ is $q_{\mathsf{H}}/|\mathcal{K}|$.

If (10) $\Rightarrow \mathbf{true}$, the test session has a partially matching session and it is of type "Re". $\mathcal{A}$ is allowed to obtain both long-term secret keys as the test session is completed. $\mathcal{A}$ is not allowed to query the state of the partially matching session. This yields the same scenario as described above. It follows that

$$\Pr[\mathrm{QUERY}^* \mid (1 \vee 2) \Rightarrow \mathbf{true}] = \Pr[\mathrm{QUERY}^* \mid (10) \Rightarrow \mathbf{true}] \leq \frac{q_{\mathsf{H}}}{|\mathcal{K}|} \ .$$

Attack $(7 \vee 8)$, (16). If $(7 \vee 8) \Rightarrow \mathbf{true}$, the test session has a matching session and the state (IV^*, ψ^*) is revealed. Furthermore, $\mathcal{A}$ can obtain $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$. As the initiator is not corrupted, $\widetilde{\mathsf{sk}}^*$ is unknown to $\mathcal{A}$, unless it issues a query to G on a correct value k_{i^*}. The probability that this happens is upper bounded by $q_{\mathsf{G}}/2^{\kappa}$. Another way to learn the session key is to query H directly, where $\mathcal{A}$ succeeds with probability $q_{\mathsf{H}}/|\mathcal{K}|$, as $\widetilde{K}^*$ is chosen uniformly at random.

If (16) $\Rightarrow \mathbf{true}$, the test session has a partially matching session and it is of type "Re". $\mathcal{A}$ can reveal the state (IV, ψ) of the partially matching session, the rest remains unchanged to before. Thus,

$$\Pr[\mathrm{QUERY}^* \mid (7 \vee 8) \Rightarrow \mathbf{true}] = \Pr[\mathrm{QUERY}^* \mid (16) \Rightarrow \mathbf{true}] \leq \frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|} \ .$$

Fig. 27. Adversaries $\mathcal{D}_b$ against S-NCKE-CPA for the proof of Eqn. (11). $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathsf{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathsf{I}}, \textsc{Reveal}, \textsc{Rev-State}, \textsc{Corrupt}, \textsc{Test}, \mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{H}_1, \ldots, \mathsf{H}_N, \widetilde{\mathsf{H}}_1, \ldots, \widetilde{\mathsf{H}}_S\}$, where Reveal, Corrupt, Test, F, H and H_n for $n \in [N]$ are defined as in games $G_{3,b}$ in Figure 23 resp. 24. Lines written in blue color highlight how the adversary simulates $G_{3,b}$ and interpolates to $G_{4,b}$.

Attack	Security relies on ...
$(1 \vee 2)$, (10)	IV^* unknown $\Rightarrow \widetilde{\mathsf{sk}}^*$ unknown $\Rightarrow \widetilde{K}^*$ unknown
$(7 \vee 8)$, (16)	k_{i^*} unknown $\Rightarrow \widetilde{\mathsf{sk}}^*$ unknown $\Rightarrow \widetilde{K}^*$ unknown
(17), (23)	sk_{r^*} unknown before session completed $\Rightarrow \pi^*$ cannot be computed
(18)	sigk_{i^*} unknown before session completed $\Rightarrow \sigma^*$ cannot be forged

Fig. 28. Overview of attacks for the proof of Theorem 3.

Attack (17), (23). If (17) $\Rightarrow$ **true**, the test session has no matching session and it is of type "In". $\mathcal{A}$ is allowed to obtain the initiator's long-term secret key $(\mathsf{sigk}_{i^*}, \mathsf{sk}_{i^*}, k_{i^*})$ and can choose $(\tilde{c}^*, \widetilde{K}^*)$ itself, but the responder's long-term secret key $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$ will only be available after the session key is established. Thus, $\mathcal{A}$ does not know K_{r^*}, which is a uniformly random key. $\mathcal{A}$ can only complete the session if it manages to compute π by querying F on K_{r^*}, where the probability for that is upper bounded by $q_{\mathsf{F}}/|\mathcal{K}|$.

If (23) $\Rightarrow$ **true**, instead of obtaining the initiator's long-term secret key, $\mathcal{A}$ is allowed to obtain the initiator's state (IV^*, ψ^*). The rest remains the same. $\mathcal{A}$ can only complete the session if it manages to forge π, for which it has to query F on the correct key K_{r^*}. Thus,

$$\Pr[\text{QUERY}^* \mid (17) \Rightarrow \textbf{true}] = \Pr[\text{QUERY}^* \mid (23) \Rightarrow \textbf{true}] \leq \frac{q_{\mathsf{F}}}{|\mathcal{K}|} \ .$$

Attack (18). If (18) $\Rightarrow$ **true**, the test session has no matching session and it is of type "Re", which means that $\mathcal{A}$ can reveal the responder's long-term secret key $(\mathsf{sigk}_{r^*}, \mathsf{sk}_{r^*}, k_{r^*})$. Here, $\mathcal{A}$ has two possibilities. First, it can choose $(\widetilde{\mathsf{pk}}^*, \widetilde{\mathsf{sk}}^*)$ and (c_{r^*}, K_{r^*}) itself. However, the initiator's long-term secret key $(\mathsf{sigk}_{i^*}, \mathsf{sk}_{i^*}, k_{i^*})$ will only be available after the session key is established and $\mathcal{A}$ has to forge σ_{i^*} to call $\textsc{Session}_{\mathsf{R}}$ in the first place. As the game aborts if that happens, the session key will never be computed.

$$\Pr[\text{QUERY}^* \mid (18) \Rightarrow \textbf{true}] = 0 \ .$$

Taking the maximum over the conditional probabilities and assuming that $q_{\mathsf{H}} \approx q_{\mathsf{F}}$, it follows that

$$\begin{aligned} \left|\Pr[G^{\mathcal{A}}_{5,0} \Rightarrow 1] - \Pr[G^{\mathcal{A}}_{4,0} \Rightarrow 1]\right| &\leq \Pr[\text{QUERY}] \leq T \cdot \Pr[\text{QUERY}^*] \\ &\leq T \cdot \left(\frac{q_{\mathsf{G}}}{2^{\kappa}} + \frac{q_{\mathsf{H}}}{|\mathcal{K}|}\right) \end{aligned} \tag{13}$$

The proof of Theorem 3 follows by collecting the probabilities from Equations (7)-(13) and by folding adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$, $\mathcal{C}_0$ and $\mathcal{C}_1$ as well as $\mathcal{D}_0$ and $\mathcal{D}_1$ into single adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. □

7 Concrete Instantiation of AKE Protocols

7.1 NCKE from the DDH Assumption

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

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

We define $\mathsf{KEM}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Encaps}_{\mathsf{DDH}}, \mathsf{Decaps}_{\mathsf{DDH}})$ with global parameters $\mathsf{par} := (\mathbb{G}, p, g_1, g_2)$ as shown in Figure 29.

$\underline{\mathsf{Gen}_{\mathsf{DDH}}(\mathsf{par})}$	$\underline{\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{DDH}}(\mathsf{pk}, m)}$	$\underline{\mathsf{Decaps}^{\mathsf{H}}_{\mathsf{DDH}}(\mathsf{sk}, (c_1, c_2))}$
00 $(x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2$	03 $r \xleftarrow{\$} \mathbb{Z}_p$	07 $K := \mathsf{H}(c_1, c_2, c_1^{x_1} c_2^{x_2})$
01 $X := g_1^{x_1} g_2^{x_2}$	04 $(c_1, c_2) := (g_1^r, g_2^r)$	08 **return** K
02 **return** $(\mathsf{pk} := X,$	05 $K := \mathsf{H}(c_1, c_2, X^r)$	
$\mathsf{sk} := (x_1, x_2))$	06 **return** $((c_1, c_2), K)$	

Fig. 29. Key encapsulation mechanism $\mathsf{KEM}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{Encaps}_{\mathsf{DDH}}, \mathsf{Decaps}_{\mathsf{DDH}})$.

Definition 3 (m-fold DDH Problem). *Let* GGen *be a PPT algorithm that on input* 1^κ *outputs a cyclic group* $\mathbb{G}$ *of prime order* $2^{k-1} \leq p \leq 2^k$ *with generator* g_1*. Furthermore let* $g_2 = g_1^\omega$ *for* $\omega \xleftarrow{\$} \mathbb{Z}_p$*. The* m-DDH *problem requires to distinguish* m DDH *tuples from* m *uniformly random tuples:*

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

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*

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

The following theorem establishes that the construction given in Figure 29 is an N-receiver non-committing encapsulation mechanism under the DDH assumption.

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

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

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

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

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

If $r \neq r'$, then $\det M = w(r' - r) \neq 0$, which implies that pk and Z are random and independent group elements as long as x_1, x_2 are unknown. Thus, for all $Z' \in \mathcal{Z}$, holds that $\Pr[Z = Z'] = 1/p$. In Definition 3, all values r_i and r'_i are drawn uniformly at random from $\mathbb{Z}_p$. The probability that $r_i = r'_i$ for any $i \in [N \cdot q_E]$ is upper bounded by $N \cdot q_E/p$. Furthermore, the probability that a specific challenge ciphertext is issued to Decaps before it is output by Encaps is at most q_D/p. It follows that

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

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 pk_i in the second message, as $\mathsf{KEM}_{\mathsf{DDH}}$ is a multi-recipient KEM. We establish Theorem 5 and give a proof sketch.

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

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

where κ *is a security parameter.*

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

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

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

Union bound yields $(N+S)^2/p$, as we have N long-term public keys and at most S ephemeral public keys. For ciphertexts $(c_1, c_2) \in \mathcal{C}$ output by the encapsulation algorithm $\mathsf{Encaps}_{\mathsf{DDH}}$, it holds that

$$\Pr[r \xleftarrow{\$} \mathbb{Z}_p : (c_1, c_2) = (g_1^r, g_2^r)] = 1/p\ ,$$

which yields an upper bound for collisions of S^2/p, as there are at most S sessions with one ciphertext. We also assume that values IV are different in all sessions and keys k_n are different for all parties.

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

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

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

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

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

oracle F*,* $q_{\widetilde{\mathsf{H}}}$*,* q_{H_n} *queries to each random oracle* $\widetilde{\mathsf{H}}_{\mathrm{sID}}$ *and* H_n *and at most* q_{H} *queries to random oracle* H*, there exists an adversary* $\mathcal{B}$ *against* DDH *and an adversary* $\mathcal{C}$ *against* N-SUF-CMA *such that*

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

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

The signature scheme can be instantiated with the tight scheme based on the DDH and CDH assumption proposed by Gjøsteen and Jager in [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).

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)
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)
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)
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)
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)
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)
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)
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)
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)
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)
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)

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)
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)
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)
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)
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)
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)
22. Hofheinz, D., Kiltz, E.: The group of signed quadratic residues and applications. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 637–653. Springer, Heidelberg (Aug 2009)
23. 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)
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)
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)
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)
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)
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)
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)
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)
32. Liu, X., Liu, S., Gu, D., Weng, J.: Two-Pass Authenticated Key Exchange with Explicit Authentication and Tight Security. In: ASIACRYPT 2020. p. ??? LNCS (2020)
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)
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)
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.) Advances in Information and Computer Security - 7th International Workshop on Security, IWSEC 2012, Fukuoka, Japan, November 7-9, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7631, pp. 69–86. Springer (2012), https://doi.org/10.1007/978-3-642-34117-5_5

A NCKE from the Higher Residuosity Assumption

We show how to construct an NCKE scheme using the hash proof system based on the higher residuosity (HR) problem described in [22]. Therefore, we will first recall some definitions.

Quadratic Residues. An n-bit integer $N = PQ$, where P, Q are two distinct $n/2$-bit odd primes is called an RSA modulus. We assume that N is a Blum integer, i. e., both P and Q are congruent 3 modulo 4. By $\phi(N)$ we denote Euler's totient function, i. e. $\phi(N) = (P-1)(Q-1)$. By $\mathbb{J}_N$ we denote the subgroup of all elements from $\mathbb{Z}_N^*$ with Jacobi symbol 1 and by $\mathbb{QR}_N$ the group of quadratic residues modulo N, which is a subgroup of $\mathbb{J}_N$ with order $(P-1)(Q-1)/4$.

Signed Quadratic Residues. For $x \in \mathbb{Z}_N$, let $|x|$ denote the absolute value of x, where x is represented as a signed integer in the set $\{-(N-1)/2, ..., (N-1)/2\}$. The signed group $\mathbb{G}^+$, where $\mathbb{G}$ is a subgroup of $\mathbb{Z}_N^*$, is defined as $\mathbb{G}^+ := \{|x| : x \in \mathbb{G}\}$. For $g, h \in \mathbb{G}^+$ and integer x, we define $g \circ h := |g \cdot h \bmod N|$ and $g^{\underline{x}} := |g^x \bmod N|$. Our focus will be on the group of signed quadratic residues $\mathbb{QR}_N^+$.

Lemma 2. [22] *Let N be a Blum integer. Then:*
(i) $(\mathbb{QR}_N^+, \circ)$ is a group of order $\phi(N)/4$.
(ii) $\mathbb{QR}_N^+ = \mathbb{J}_N^+$. In particular, $\mathbb{QR}_N^+$ is efficiently recognizable (given only N).
(iii) If $\mathbb{QR}_N$ is cyclic, so is $\mathbb{QR}_N^+$.

RSA Instance Generator. Let $0 \le \delta \le 1/4$ be a constant and $n(\kappa)$ be a function. Let RSAgen be an algorithm that generates elements (N, P, Q, S) such that $N = PQ$ is an n-bit Blum integer. The prime factors of $\phi(N)/4$ are pairwise distinct and at least δn-bit integers. Furthermore, $S > 1$ is a divisor of $\phi(N)/4$ with $1 < \gcd(S, (P-1)/2) < (P-1)/2$ and $1 < \gcd(S, (Q-1)/2) < (Q-1)/2$.

Statistical Distance. The statistical distance between two random variables X and Y having a common domain $\mathcal{X}$ is defined as $\Delta[X, Y] = \frac{1}{2}\sum_{x\in\mathcal{X}} |\Pr[X = x] - \Pr[Y = x]|$. The min-entropy of a random variable X is defined as $H_\infty(X) = -\log(\max_{x\in\mathcal{X}} \Pr[X = x])$.

Now we describe our HPS. Define $group = (N, g)$, where $(N, P, Q, S) \leftarrow \mathsf{RSAgen}(1^\kappa)$ and g is a uniform generator of $\mathbb{G}_S^+$. Define $\mathcal{Y} = \mathbb{QR}_N^+$ and $\mathcal{X} = \mathbb{G}_S^+ = \{g^{\underline{r}} : r \in \mathbb{Z}_S\}$. A value r is a witness that $c \in \mathcal{X}$. It is possible to sample an almost uniform element from $\mathcal{X}$ together with a witness by first picking $r \in [N/4]$ and defining $c = g^{\underline{r}} \in \mathbb{G}_S^+$. We will determine the statistical distance below in Equation (14). Membership in $\mathcal{Y}$ can be efficiently checked by Lemma 2. Define $\mathcal{SK} = [N/4]$, $\mathcal{PK} = \mathbb{G}_S^+$ and $\mathcal{Z} = \mathbb{QR}_N^+$. For $\mathsf{sk} = x \in [N/4]$, define $\mu(\mathsf{sk}) = X = g^{\underline{x}} \in \mathbb{G}_S^+$. This defines the output of the parameter generation algorithm Par.

For $c \in \mathcal{Y}$ define $\Lambda_{\mathsf{sk}}(c) := Z = c^{\underline{x}}$. This defines the private evaluation algorithm $\mathsf{Priv}(\mathsf{sk}, c)$. Given $\mathsf{pk} = \mu(\mathsf{sk}) = X$, $c \in \mathcal{X}$ and a witness $r \in \mathbb{Z}$ such that $c = g^{\underline{r}}$, the public evaluation algorithm $\mathsf{Pub}(\mathsf{pk}, c, r)$ computes $Z = \Lambda_{\mathsf{sk}}(c)$ as $Z = X^{\underline{r}}$.

We want to analyze the statistical distance between the two distributions defined by sampling from $X_1 = [\phi(N)/4]$ and sampling from $X_2 = [N/4]$. As $\phi(N)/4 = ST$ and $N/4 = ST + (P + Q - 1)/4$, we can write the statistical distance as

$$\Delta[X_1, X_2] = \frac{(P+Q-1)/4}{N/4} = \frac{1}{P} + \frac{1}{Q} - \frac{1}{PQ} \le \frac{1}{P} + \frac{1}{Q} = O(2^{-n/2}) \ , \tag{14}$$

where the last equation holds because P and Q are both $n/2$-bit primes.

Definition 4 (m-fold Higher Residuosity Problem). *Let (N, P, Q, S) be generated by* RSAgen*. The higher residuosity (HR) problem requires to distinguish m random elements from $\mathbb{G}_S^+$ from m random elements from $\mathbb{QR}_N^+$. The advantage of an adversary $\mathcal{A}$ against the* HR *problem is defined as*

$$\mathrm{Adv}^{m\text{-HR}}_{\mathsf{RSAgen}}(\mathcal{A}) := |\Pr[\mathcal{A}(N, g, c_1, ..., c_m) \Rightarrow 1] - \Pr[\mathcal{A}(N, g, c'_1, ..., c'_m) \Rightarrow 1]| \ ,$$

where the probability is taken over $(N, P, Q, S) \leftarrow \mathsf{RSAgen}$, $c_1, ..., c_m \xleftarrow{\$} \mathbb{G}_S^+$ and $c'_1, ..., c'_m \xleftarrow{\$} \mathbb{QR}_N^+$ as well as the coin tosses of $\mathcal{A}$.

Next, we want to show that the m-fold HR problem tightly reduces to the (1-)HR problem using random self-reducibility.

Lemma 3 (Random self-reducibility of HR). *For any adversary $\mathcal{A}$ against the m-fold* HR *problem, there exists an adversary $\mathcal{B}$ against the* HR *problem with roughly the same running time such that*

$$\mathrm{Adv}^{m\text{-HR}}_{\mathsf{RSAgen}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{B}) + m \cdot O(2^{-\delta n(\kappa)}) + O(2^{-n/2}) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against the m-HR problem. We construct adversary $\mathcal{B}$ against the (1-)HR problem as shown in Figure 30.

$\underline{\mathcal{B}(N, g, c)}$
00 **for** $i \in [m]$
01 $\quad a_i \xleftarrow{\$} [N/4]$
02 $\quad c_i := c^{a_i}$
03 $b' \leftarrow \mathcal{A}(N, g, c_1, ..., c_m)$
04 **return** b'

Fig. 30. Adversary $\mathcal{B}$ against the HR problem for the proof of Lemma 3.

$\mathcal{B}$ inputs (N, g, c), where g is a generator of $\mathbb{G}_S^+$ and c is either a random element from $\mathbb{G}_S^+$ or from $\mathbb{QR}_N^+$. It samples m random elements a_i from $[N/4]$, computes c_i as c^{a_i} and runs adversary $\mathcal{A}$ on input $(N, g, c_1, ..., c_m)$.

First, note that $|\mathbb{G}_S^+| = S$ and $|\mathbb{QR}_N^+| = ST$, where S and T have only prime factors that are distinct and greater than $2^{-\delta n(\kappa)}$. Thus, c is a generator of $\mathbb{G}_S^+$ or $\mathbb{QR}_N^+$ with high probability $1 - O(2^{-\delta n(\kappa)})$.

Second, if c is a generator of $\mathbb{G}_S^+$ resp. $\mathbb{QR}_N^+$, then $c_i := c^{a_i}$, where $a_i \xleftarrow{\$} [\phi(N)/4]$ and $i \in [m]$, are m random and independent elements in the corresponding group. As $\mathcal{B}$ does not know $\phi(N)$, it samples exponents a_i from $[N/4]$. As shown above, the statistical distance between these $(c_1, ..., c_m)$ and the input of $\mathcal{A}$ in the original m-HR experiment is bounded by $m \cdot O(2^{-\delta n(\kappa)})$.

This yields the bound stated in Lemma 3. □

$\underline{\mathsf{Gen}_{\mathsf{HR}}(\mathsf{par})}$	$\underline{\mathsf{Encaps}^{\mathsf{H}}_{\mathsf{HR}}(\mathsf{pk})}$	$\underline{\mathsf{Decaps}^{\mathsf{H}}_{\mathsf{HR}}(\mathsf{sk}, c)}$
00 $x \xleftarrow{\$} [N/4]$	03 $r \xleftarrow{\$} [N/4]$	07 $K := \mathsf{H}(c, c^x)$
01 $X := g^x$	04 $c := g^r$	08 **return** K
02 **return** $(\mathsf{pk} := X,$	05 $K := \mathsf{H}(c, X^r)$	
$\quad \mathsf{sk} := x)$	06 **return** (c, K)	

Fig. 31. Key encapsulation mechanism $\mathsf{KEM}_{\mathsf{HR}} = (\mathsf{Gen}_{\mathsf{HR}}, \mathsf{Encaps}_{\mathsf{HR}}, \mathsf{Decaps}_{\mathsf{HR}})$.

Theorem 7. *Under the* HR *assumption and in the random oracle model,* $\mathsf{KEM}_{\mathsf{HR}}$ *is an N-receiver non-committing key encapsulation mechanism. In particular, for any N-*NCKE-CCA *adversary $\mathcal{A}$ against* $\mathsf{KEM}_{\mathsf{HR}}$ *and* $\mathsf{Sim}_{\mathsf{HR}}$ *that issues at most q_E queries per user to* Encaps*, q_D queries to* Decaps *and at most q_{H} queries to each random oracle H_n, $n \in [N]$, there exists an adversary $\mathcal{B}$ against* HR *with roughly the same running time such that*

$$\mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{HR}},\mathsf{Sim}_{\mathsf{HR}}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{HR}}_{\mathsf{RSAgen}}(\mathcal{B}) + (N \cdot q_E)(q_{\mathsf{H}} + q_D + 1) \cdot O(2^{-\delta n(\kappa)}) + O(2^{-n/2}) \ .$$

Proof. As shown in [22], HPS is $\delta n(\kappa)$-entropic. Furthermore, the m-fold subset membership problem is hard in HPS by definition of the m-HR assumption. Thus, Theorem 1 yields

$$\mathrm{Adv}^{N\text{-NCKE-CCA}}_{\mathsf{KEM}_{\mathsf{HR}},\mathsf{Sim}_{\mathsf{HR}}}(\mathcal{A}) \leq \mathrm{Adv}^{m\text{-HR}}_{\mathsf{RSAgen}}(\mathcal{B}) + (N \cdot q_E \cdot q_{\mathsf{H}}) \cdot 2^{-\delta n(\kappa)} + (N \cdot q_E \cdot q_D) \cdot 2^{-\delta n(\kappa)} \ .$$

Applying Lemma 3 yields the bound stated in Theorem 7. □

B Full Attack Tables for our AKE Model

In the following, we want to give the complete tables of possible attacks in the FS and wFS security model. Note that these tables contains a large number of redundant rows. We do this to justify completeness and point out the security properties. In the next step, we will distill the tables for the special case of two-message protocols, thus reducing complexity. We also point out trivial attacks.

B.1 Overview of Allowed Attacks for Full Forward Security

We begin with the complete table of possible attacks for full forward security, which is given in Table 3. The variables used are explained in Section 4. The structure is as follows:

- Attack (0) covers that it is considered a valid attack when a session is recreated due to insufficient randomness. Therefore, if there is more than one partially matching session to a test session, the adversary may also run a trivial attack.
- Attacks (1)-(8) capture all attacks, where a matching session exists.
- Attacks (9)-(16) capture all attacks, where a partially matching session exists, but no full matching session.
- Attacks (17)-(24) capture all attacks, where neither a partially nor a full matching session exists.

We cover all possible combinations of long-term key corruptions and state reveals, also taking into account when a corruption may happen (modeled by variable peerCorrupted). Thereby, the allowed attacks include forward security, KCI security, and security against maximal exposure:

Forward Security is covered, for instance, by attacks (17) and (18), which enable an active adversary (i.e., $|\mathfrak{M}(\text{sID}^*)| = 0$) to obtain the long-term secret of one or both parties. The peer's long-term secret key will be available after the session key has been computed.

Key Compromise Impersonation is covered by attacks (21)-(24), where the adversary obtains at least the long-term secret of one party.

Maximal Exposure Attacks are covered by the fact that all combinations of long-term secret and state reveals are allowed, except for those that lead to trivial attacks (e.g., reveal of long-term key and state of the same party). In particular, we allow the adversary to obtain both states in attacks (19) and (20).

Distilled Table for Two-Message Protocols. We consider the full table and give a distilled version in Table 2 in Section 4. The simplifications are due to the following reasons:

- Attacks (1) and (2) can be merged by setting the type to arbitrary.
- Attacks (3) and (4) can be removed as the responder does not have a state and thus this is already captured by attacks (7) and (8).
- Attacks (5) and (6) can be removed for the same reason, they are already captured by attacks (1) and (2).
- Attacks (7) and (8) can be merged by setting the type to arbitrary and allowing to reveal both states as only the initiator's state contains meaningful information.
- Attacks (9), (11), (13) and (15) can be removed as by definition, a partially matching session can never be of type "In".
- Attacks (12) and (14) can be removed as the responder does not have a state and thus these attacks are already captured by attacks (16) and (10).
- Attacks (19) and (22) can be removed as the responder does not have a state and thus this is already captured by attacks (23) and (18).
- Attack (20) can be removed as the adversary can compute the state on his own and thus this is already captured by attack (22).
- Attacks (21) and (24) equal attacks (17) and (18) and can be removed as the adversary can compute the state on his own.

B.2 Overview of Allowed Attacks for Weak Forward Security

Compared to full forward security, weak forward security only provides security against a passive adversary (i.e., it must hold that $|\mathfrak{M}(\text{sID}^*)| = 1$) when both secret keys are revealed. This is the strongest form of forward security that implicitly authenticated protocols can achieve. Our model covers this in attacks (1) and (2).

	$\mathcal{A}$ gets (Initiator, Responder)	corrupted[i^*]	corrupted[r^*]	peerCorrupted[sID*]	type[sID*]	revState[sID*]	$\exists$sID $\in \mathfrak{M}$(sID*) : revState[sID]	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$	$\exists$sID $\in \mathfrak{P}$(sID*) : revState[sID]	$\lvert\mathfrak{P}(\text{sID}^*)\rvert$
(0)	**multiple partially matching sessions**	–	–	–	–	–	–	–	–	> 1
(1)	**(long-term, long-term)**	–	–	–	"In"	**F**	**F**	1	–	–
(2)	**(long-term, long-term)**	–	–	–	"Re"	**F**	**F**	1	–	–
(3)	**(state, state)**	**F**	**F**	–	"In"	–	–	1	–	–
(4)	**(state, state)**	**F**	**F**	–	"Re"	–	–	1	–	–
(5)	**(long-term, state)**	–	**F**	–	"In"	**F**	–	1	–	–
(6)	**(long-term, state)**	–	**F**	–	"Re"	–	**F**	1	–	–
(7)	**(state, long-term)**	**F**	–	–	"In"	–	**F**	1	–	–
(8)	**(state, long-term)**	**F**	–	–	"Re"	**F**	–	1	–	–
(9)	**(long-term, long-term)**	–	–	**F**	"In"	**F**	n/a	0	**F**	1
(10)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	**F**	1
(11)	**(state, state)**	**F**	**F**	–	"In"	–	n/a	0	–	1
(12)	**(state, state)**	**F**	**F**	–	"Re"	–	n/a	0	–	1
(13)	**(long-term, state)**	–	**F**	–	"In"	**F**	n/a	0	–	1
(14)	**(long-term, state)**	–	**F**	**F**	"Re"	–	n/a	0	**F**	1
(15)	**(state, long-term)**	**F**	–	**F**	"In"	–	n/a	0	**F**	1
(16)	**(state, long-term)**	**F**	–	–	"Re"	**F**	n/a	0	–	1
(17)	**(long-term, long-term)**	–	–	**F**	"In"	**F**	n/a	0	n/a	0
(18)	**(long-term, long-term)**	–	–	**F**	"Re"	**F**	n/a	0	n/a	0
(19)	**(state, state)**	**F**	**F**	–	"In"	–	n/a	0	n/a	0
(20)	**(state, state)**	**F**	**F**	–	"Re"	–	n/a	0	n/a	0
(21)	**(long-term, state)**	–	**F**	–	"In"	**F**	n/a	0	n/a	0
(22)	**(long-term, state)**	–	**F**	**F**	"Re"	–	n/a	0	n/a	0
(23)	**(state, long-term)**	**F**	–	**F**	"In"	–	n/a	0	n/a	0
(24)	**(state, long-term)**	**F**	–	–	"Re"	**F**	n/a	0	n/a	0

Table 3. Full table of attacks for full and weak FS adversaries. For two-message protocols, a partial matching session can only be of type "Re" and we can exclude attacks highlighted in green color. Furthermore, for weak FS adversaries we exclude trivial attacks which are highlighted in blue color. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value. "**F**" means "false", "n/a" indicates that there is no state which can be revealed as no (partially) matching session exists.

Distilled Table for wFS and Two-Message Protocols. We use Table 3 and distill it such that it considers weak forward security for two-message protocols. The result is given in Table 1 in Section 4. Column peerCorrupted has been removed as we no longer consider the time when a corruption happens. We justify the removal of trivial attacks as follows:

- Attacks (9), (11), (13) and (15) can be removed as by definition, a partial matching session can never be of type "In".
- Attacks (17) and (23) have to be removed as the adversary can trivially win by obtaining the responder's long-term secret and computing the last message.
- Attacks (18) and (22) have to be removed as the active adversary can trivially win by impersonating the initiator and choosing its own state for the first message.

Furthermore, we do some optimizations:

- Attacks (1) and (2) can be merged by setting the type to arbitrary.
- Attacks (3) and (4) can be removed as the responder does not have a state and thus this is already captured by attacks (7) and (8).
- Attacks (5) and (6) can be removed for the same reason, they are already captured by attacks (1) and (2).

– Attacks (7) and (8) can be merged by setting the type to arbitrary and allowing to reveal both states as only the initiator's state contains meaningful information.
– Attacks (12) and (14) can be removed as the responder does not have a state and thus these attacks are already captured by attacks (16) and (10).
– Attack (20) can be removed as the the responder does not have a state and thus this is already captured by attack (24).