The Pre-Shared Key Modes of HPKE

Joël Alwen[1], Jonas Janneck[2], Eike Kiltz[2], and Benjamin Lipp[3]

[1] AWS-Wickr, Seattle, USA
alwenjo@amazon.com
[2] Ruhr-Universität Bochum, Germany
{jonas.janneck,eike.kiltz}@rub.de
[3] Max Planck Institute for Security and Privacy, Bochum, Germany
benjamin.lipp@mpi-sp.org

October 17, 2023

Abstract. The Hybrid Public Key Encryption (HPKE) standard was recently published as RFC 9180 by the Crypto Forum Research Group (CFRG) of the Internet Research Task Force (IRTF). The RFC specifies an efficient public key encryption scheme, combining asymmetric and symmetric cryptographic building blocks.

Out of HPKE's four modes, two have already been formally analyzed by Alwen et al. (EUROCRYPT 2021). This work considers the remaining two modes: $\mathsf{HPKE}_{\mathsf{PSK}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$. Both of them are "pre-shared key" modes that assume the sender and receiver hold a symmetric pre-shared key. We capture the schemes with two new primitives which we call pre-shared key public-key encryption (pskPKE) and pre-shared key authenticated public-key encryption (pskAPKE). We provide formal security models for pskPKE and pskAPKE and prove (via general composition theorems) that the two modes $\mathsf{HPKE}_{\mathsf{PSK}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ offer active security (in the sense of insider privacy and outsider authenticity) under the Gap Diffie-Hellman assumption.

We furthermore explore possible post-quantum secure instantiations of the HPKE standard and propose new solutions based on lattices and isogenies. Moreover, we show how HPKE's basic $\mathsf{HPKE}_{\mathsf{PSK}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ modes can be used black-box in a simple way to build actively secure post-quantum/classic-hybrid (authenticated) encryption schemes. Our hybrid constructions provide a cheap and easy path towards a practical post-quantum secure drop-in replacement for the basic HPKE modes $\mathsf{HPKE}_{\mathsf{Base}}$ and $\mathsf{HPKE}_{\mathsf{Auth}}$.

Keywords. Authenticated Public Key Encryption, Post-Quantum Hybrid, Open Standards, HPKE

A preliminary version of this paper appears in the proceedings of the 29th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2023). This is the full version.

Table of Contents

1 Introduction

The Hybrid Public Key Encryption (HPKE) standard was published as RFC 9180 [4] by the Crypto Forum Research Group (CFRG) of the Internet Research Task Force (IRTF)[4] in February 2022. The RFC specifies an efficient public key encryption scheme, combining asymmetric and symmetric cryptographic building blocks. While this an old and relatively well understood paradigm, the new standard was developed in an effort to address issues in previous standardizations of hybrid public key encryption. For example, HPKE relies on modern cryptographic building blocks, provides test vectors to ease development of interoperable implementations, and already received some cryptographic analysis during its development, inspired by the "analysis-prior-to-deployment" design philosophy adopted for the development of the TLS 1.3 protocol [20]. At the time of development of HPKE, two IETF standardization efforts already started building upon it: the Messaging Layer Security (MLS) protocol [3], and the Encrypted Client Hello privacy extension of TLS 1.3 [21]. Since its publication, HPKE has also been adopted by other higher-level protocols, like the published RFC 9230 Oblivious DNS over HTTPS [16], and the Distributed Aggregation Protocol for Privacy Preserving Measurement [15], and thus, has become an important building block of today's and the future Internet.

The HPKE standard may appear to resemble a "public key encryption" approach, aligning with the KEM/DEM paradigm [9]. Indeed, it incorporates a Key Encapsulation Mechanism (KEM) and an Authenticated Encryption with Associated Data (AEAD), functioning as a Data Encapsulation Mechanism (DEM) based on the KEM/DEM paradigm. However, upon closer inspection HPKE turns out to be more complex than this perfunctory description implies. First, HPKE actually consists of 2 different KEM/DEM constructions. Moreover, each construction can also be instantiated with a pre-shared key (*psk*) known to both sender and receiver, which is used in the key schedule KS to derive the DEM key. In total this gives rise to 4 different *modes* for HPKE.

The *basic* mode $\mathsf{HPKE}_{\mathsf{Base}}$ makes use of a standard KEM to obtain a "message privacy and integrity" only mode. This mode can be extended to $\mathsf{HPKE}_{\mathsf{PSK}}$ to support authentication of the sender via a *psk*. The remaining 2 HPKE modes make use of a different KEM/DEM construction built from a rather non-standard KEM variant called *Authenticated KEM* (AKEM) [1]. An AKEM can be thought of the KEM analogue of signcryption [22]. In particular, sender and receiver both have their own public/private keys. Each party requires their own private and the other party's public key to perform en/decryption. The AKEM-based HPKE modes also intend to authenticate the sender to the receiver. Just as in the KEM-based case, the AKEM/DEM construction can be instantiated in modes either without *psk* ($\mathsf{HPKE}_{\mathsf{Auth}}$) or with a *psk* ($\mathsf{HPKE}_{\mathsf{AuthPSK}}$). The HPKE RFC constructs a KEM and an AKEM based on specific Diffie-Hellman groups (such as P-256, P-384, P-521 NIST curves [19], Curve25519, or Curve448 [17]). Alwen, Blanchet, Hauck, Kiltz, Lipp, and Riepel [1] have analyzed the security of the Diffie-Hellman AKEM and showed that it can be securely combined with a key schedule KS and an AEAD to obtain concrete security bounds for the $\mathsf{HPKE}_{\mathsf{Auth}}$ modes, as defined in the HPKE standard, see Table 1. Their work explicitly leaves analyzing the remaining two HPKE modes $\mathsf{HPKE}_{\mathsf{PSK}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ for future work.

[4] https://irtf.org/cfrg

Table 1: HPKE modes and their security.

HPKE mode	Authenticated?	PSK?	Primitive	Security
$\mathsf{HPKE}_{\mathsf{Base}}$	–	–	PKE	$\mathsf{CCA}_{\mathsf{PKE}}$ (folklore)
$\mathsf{HPKE}_{\mathsf{Auth}}$	√	–	APKE [1]	Insider-CCA & Outsider-Auth [1]
$\mathsf{HPKE}_{\mathsf{PSK}}$	–	√	pskPKE (§3)	CCA & Auth (§4)
$\mathsf{HPKE}_{\mathsf{AuthPSK}}$	√	√	pskAPKE (§3)	Insider-CCA & Outsider-Auth (§4)

Applications of HPKE's PSK Modes. One class of use cases for HPKE's PSKs is a sender to transferring security guarantees from external cryptographic applications to an HPKE ciphertext. More concretely, a sender might want to transfer the post-quantum security guarantees provided by a particular PSK source to a (maybe even only otherwise classically secure) HPKE ciphertext. One such source might be PSKs distributed via an include out-of-band method (e.g., in person) or PSKs agreed upon using a post-quantum secure KEM (as demonstrated in Section 5).

In another example, HPKE's PSKs can be used to transfer the strong authenticity guarantees of an ongoing Messaging Layer Security (MLS) group containing both sender and receiver to 1-on-1 messages between the two. MLS sessions include an HPKE (and signature) public key for each party in the group known to all other group members. A party's signature public key is authenticated via an associated credential binding it to its owner (e.g. an X.509 certificate issued by a CA). Each user also signs their own HPKE public key to assert their ownership to rest of the group. To provide authenticity even over many years, MLS must account for signatures and HPKE keys being corrupted mid session. Rather than assuming the credentials for a corrupt signing key will be revoked, MLS instead gives users the ability to update their keys periodically (or at will). To ensure the old keys are no longer of any use, MLS also equips the group with a sequence of shared symmetric group keys. Whenever a user updates their signature or HPKE key, a new group key is produced in a way that ensures knowing the updating client's old state (including their signature and HPKE private keys) is insufficient to learn the new group key. An MLS group including both parties A and B can now be used to provide strong sender authentication for $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ ciphertexts (e.g. beyond the authenticity provided by static credentials). Say, A wants to send a private message to B. On the one hand A use their HPKE keys from the MLS session to encrypt and authenticate the message, thereby inheriting the authenticity guarantees of the credentials in the MLS session. On the other hand, A can also derive a *psk* off of the current MLS group key (e.g. using MLS's "exporter" functionality) for use with $\mathsf{HPKE}_{\mathsf{AuthPSK}}$. Intuitively, this provides the added guarantee to B that the sender is also *currently* in the MLS session. The same method using $\mathsf{HPKE}_{\mathsf{PSK}}$ in place of $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ gives B the guarantee that the sender is a current member of the group.

1.1 Our Contributions

So far, there has only been a formal analysis of the basic mode $\mathsf{HPKE}_{\mathsf{Base}}$ and the authenticated mode $\mathsf{HPKE}_{\mathsf{Auth}}$. In this work we focus on the HPKE standard in its pre-shared key mode, both in its basic form $\mathsf{HPKE}_{\mathsf{PSK}}$ and in its authenticated mode

$\mathsf{HPKE}_{\mathsf{AuthPSK}}$. Furthermore, we explore possible future post-quantum instantiations of the HPKE standard. To this end we make the following contributions.

Pre-Shared Key (Authenticated) Public Key Encryption. We begin, in Section 3, by introducing pre-shared key public key encryption (pskPKE) and pre-shared key authenticated public key encryption (pskAPKE) schemes, where the syntax of pskPKE matches that of the single-shot basic pre-shared mode $\mathsf{HPKE}_{\mathsf{PSK}}$, and the syntax of pskAPKE matches that of the single-shot authenticated pre-shared mode $\mathsf{HPKE}_{\mathsf{AuthPSK}}$. Compared to their respective non pre-shared modes PKE and APKE, encryption and decryption additionally input *psk*, a uniform symmetric key shared between the sender and the receiver. In terms of security, we define (active, multi-user) security notions capturing both authenticity (Auth) and privacy (CCA) for pskPKE and pskAPKE.

For pskPKE, privacy is essentially standard CCA security for PKE with the difference that the adversary additionally has access to an encryption oracle (which requires the secret pre-shared key to compute the ciphertext) and it is allowed to adaptively corrupt pre-shared (symmetric) keys and long-term (asymmetric) keys. Security holds as long as at least one of the pre-shared and long-term keys used in the challenge ciphertext/forgery has not been corrupted. Authenticity for pskPKE schemes provides the adversary with the same oracles, and the adversary's goal is to non-trivially forge a fresh ciphertext, i.e., one that does not come from the encryption oracle.

Similar to APKE [1], for pskAPKE we consider so called weaker outsider and stronger insider security variants for privacy, and only outsider security for authenticity. Intuitively, outsider notions model settings where the adversary is an outside observer. Conversely, insider notions model settings where the adversary is somehow directly involved; in particular, even selecting some of the long-term asymmetric secrets used to produce target ciphertexts. A bit more formally, we call an honestly generated asymmetric key pair secure if the secret key was not (explicitly) leaked to the adversary and leaked if it was. An asymmetric key pair is called corrupted if it was sampled arbitrarily by the adversary. A scheme is outsider-secure if target ciphertexts are secure when produced using secure key pairs. Meanwhile, insider security holds even if one secure and one corrupted key pair are used. For example, insider privacy (Insider-CCA) for pskAPKE requires that an encapsulated key remains indistinguishable from random despite the encapsulating ciphertext being produced using corrupted sender keys (but secure receiver keys). Note that insider authenticity implies (but is stronger than) Key Compromise Impersonation (KCI) security as KCI security only requires authenticity for leaked (but not corrupt) receiver keys.

We remark that, for simplicity, our modeling assumes the pre-shared key *psk* to be uniformly random from some sufficiently large key-space. Indeed, The HPKE RFC mandates the PSK have at least 32 bytes of entropy to counter Partitioning Oracle Attacks [18] to which HPKE is vulnerable because it is currently only specified for AEAD schemes that are not key-committing. Thus in practice, say, hashing such a PSK to a 32 byte string prior to use with HPKE would, at least in the random oracle model, result in a near uniform distribution.

pskKEM/DEM Construction. In Section 4, we consider the pskKEM/DEM constructions that combine a KEM (AKEM) together with an AEAD (acting as a DEM) to obtain pskPKE (pskAPKE). First, we construct $\mathsf{pskPKE}[\mathsf{KEM}, \mathsf{KS}, \mathsf{AEAD}]$ from a KEM, a key schedule KS, and an AEAD. To encrypt a message, a KEM ciphertext/key pair (c, K) is generated.

Table 2: Security properties needed to prove Outsider-Auth and Insider-CCA security of pskAPKE obtained by the pskAKEM/DEM construction.

	AKEM			AEAD	
	Outsider-Auth	Outsider-CCA	Insider-CCA	INT-CTXT	IND-CPA
$\mathsf{Outsider\text{-}Auth}_{\mathsf{pskAPKE}}$	X	X		X	
$\mathsf{Insider\text{-}CCA}_{\mathsf{pskAPKE}}$			X	X	X

Next, the KEM key K is fed together with the pre-shared key *psk* into KS to obtain the DEM key which in turn is used to AEAD-encrypt the message. At an intuitive level, the DEM key remains uniform as long as one of K or *psk* is uniform, meaning one of the receiver's asymmetric key or the pre-shared key has not been corrupted. We will present two concrete security theorems bootstrapping privacy and authenticity of our pskPKE construction from standard security properties of the underlying KEM, KS, and AEAD. Similarly, we can construct pskAPKE[AKEM, KS, AEAD] by replacing the KEM with an AKEM in the first step of the construction above. We will again present two concrete security theorems bootstrapping privacy and authenticity of our pskAPKE construction from standard security properties of the underlying AKEM, KS, and AEAD. See also Table 2.

An Analysis of the $\mathsf{HPKE}_{\mathsf{PSK}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ Modes. Using the above mentioned transformations, the two modes $\mathsf{HPKE}_{\mathsf{PSK}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ of the HPKE standard can be obtained as pskPKE[DH-KEM, KS, AEAD] and pskAPKE[DH-AKEM, KS, AEAD]. Here DH-KEM is the well known Diffie-Hellman based KEM, DH-AKEM is the Diffie-Hellman based AKEM from [1], key schedule KS is constructed via the functions Extract and Expand both instantiated with HMAC, and AEAD is instantiated using AES-GCM or ChaCha20-Poly1305. Hence, our theorems from Section 4 provide concrete bounds for CCA and Auth security of $\mathsf{HPKE}_{\mathsf{PSK}}$, and Insider-CCA and Outsider-Auth security of $\mathsf{HPKE}_{\mathsf{AuthPSK}}$.

Hybrid APKE. In Section 5, we analyze a natural black-box construction of a hybrid APKE from an AKEM and from $\mathsf{HPKE}_{\mathsf{AuthPSK}}$. Intuitively, the resulting scheme's security depends on either one of two AKEM schemes being secure. (We remark that essentially same construction and analogous proof also construct hybrid PKE from a KEM and $\mathsf{HPKE}_{\mathsf{PSK}}$.) One interesting application of this construction is building a PQ/classic-hybrid APKE which can be done by combining a PQ secure AKEM with a classically secure one (in $\mathsf{HPKE}_{\mathsf{AuthPSK}}$). In comparison with the CPA secure PQ/classic-hybrid variant of HPKE in [2] our construction enjoys CCA security. Moreover, unlike [2], our construction uses both the PQ AKEM and HPKE as black-boxes meaning it can be easily implemented using only the standard interfaces to the two schemes. For these reasons our hybrid construction provides a cheap and easy path towards a practical PQ-secure (A)PKE drop-in replacement for plain HPKE.

Post-Quantum AKEM. As we have seen, AKEM schemes are the fundamental primitive underlying natural APKE and pskAPKE schemes. The HPKE standard in its $\mathsf{HPKE}_{\mathsf{Auth}}$ and $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ modes relies on the Diffie-Hellman based DH-APKE instantiation. Unfortunately, none of them offers security against attackers equipped with a quantum computer.

In Section 6 we propose two generic constructions of AKEM from basic primitives that can be instantiated in a post-quantum secure way.

A well-known approach for constructing a post-quantum AKEM is to combine a post-quantum KEM with a post-quantum signature [10]. Unfortunately, the Encrypt-then-Sign (EtS) approach turns out not to be Insider-CCA secure [10]. The Sign-then-Encrypt (StE) approach is in fact Insider-CCA and Outsider-Auth secure but extending it to Sign-then-KEM in a natural way would add unnecessary overhead through the required detour over the KEM/DEM framework. Our first construction extends EtS approach to the new "Encrypt-then-Sign-then-Hash" (EtStH) approach. It combines a KEM, a digital signature SIG, and a hash function to obtain AKEM. Concretely, AKEM encryption produces a KEM ciphertext/key pair (c, K') and then uses the sender's secret key to sign ciphertext c and the sender's public and verification key. Finally, the DEM key is derived from K', the signature and all the public keys using a hash function. Security in the sense of Insider-CCA and Outsider-Auth is proved under the assumption that the hash function is a PRF. Our scheme can be instantiated, for example, with any post-quantum secure KEM and signature scheme, for example Kyber [8] and Dilithium [11].

Our second AKEM construction relies on a non-interactive key-exchange scheme NIKE. Key encapsulation first computes an ephemeral NIKE key pair and defines the ephemeral public key as the ciphertext. Next, it derives the DEM key from the following two NIKE keys: The first (authentication) key between sender's secret key and the receiver's public key. The second (privacy) key between the ephemeral secret key and the receiver's public key. Note that the knowledge of the receiver's secret key allows to recover both NIKE keys and hence the DEM key. Security in the sense of Insider-CCA and Outsider-Auth is proved assuming the NIKE to be actively secure [13]. Instantiating it with the (actively secure) Diffie-Hellman NIKE [13], we obtain (a variant of) the DH-AKEM from the HPKE standard. But it can also be instantiated with the post-quantum secure NIKE from lattices [14] and from isogenies [12]. We remark that our NIKE approach provides deniability, whereas our more efficient EtStH construction does not.

2 Preliminaries

2.1 Notations

Sets and Algorithms. We write $h \xleftarrow{\$} \mathcal{S}$ to denote that the variable h is uniformly sampled from the finite set $\mathcal{S}$. For an integer n, we define $[n] := \{1, \ldots, n\}$. The notation $[\![b]\!]$, where b is a boolean statement, evaluates to 1 if the statement is true and 0 otherwise.

We use uppercase letters $\mathcal{A}, \mathcal{B}$ to denote algorithms. Unless otherwise stated, algorithms are probabilistic, and we write $(y_1, \ldots) \xleftarrow{\$} \mathcal{A}(x_1, \ldots)$ to denote that $\mathcal{A}$ returns $(y_1, \ldots)$ when run on input $(x_1, \ldots)$. We write $\mathcal{A}^{\mathcal{B}}$ to denote that $\mathcal{A}$ has oracle access to $\mathcal{B}$ during its execution. For a randomised algorithm $\mathcal{A}$, we use the notation $y \in \mathcal{A}(x)$ to denote that y is a possible output of $\mathcal{A}$ on input x.

Security Games. We use standard code-based security games [6]. A *game* G is a probability experiment in which an adversary $\mathcal{A}$ interacts with an implicit challenger that answers oracle queries issued by $\mathcal{A}$. The game $\mathbf{G}$ has one *main procedure* and an arbitrary amount of additional *oracle procedures* which describe how these oracle queries are answered. We denote the (binary) output b of game G between a challenger and an

adversary $\mathcal{A}$ as $\mathsf{G}^{\mathcal{A}} \Rightarrow b$. $\mathcal{A}$ is said to *win* G if $\mathsf{G}^{\mathcal{A}} \Rightarrow 1$, or shortly $\mathsf{G} \Rightarrow 1$. Unless otherwise stated, the randomness in the probability term $\Pr[\mathsf{G}^{\mathcal{A}} \Rightarrow 1]$ is over all the random coins in game G.

We now recall definitions for (authenticated) KEMs (Section 2.2), authenticated PKE schemes (Section 2.3), and digital signatures (Section 2.6). Standard definitions of pseudo-random functions (Section 2.4), Authenticated Encryption with Associated Data (Section 2.5), and non-interactive key exchange (Section 2.7) are postponed to the appendix.

2.2 (Authenticated) Key Encapsulation Mechanisms

We first recall syntax and security of a KEM.

Definition 1 (KEM). *A key encapsulation mechanism* KEM *consists of three algorithms:*

- Gen *outputs a key pair* (sk, pk)*, where* pk *defines a key space* $\mathcal{K}$*.*
- Encaps *takes as input a (receiver) public key* pk*, and outputs an encapsulation* c *and a shared secret* $K \in \mathcal{K}$ *(or* $\perp$*).*
- *Deterministic* Decaps *takes as input a (receiver) secret key* sk *and an encapsulation* c*, and outputs a shared key* $K \in \mathcal{K}$ *(or* $\perp$*).*

We require that for all $(sk, pk) \in \mathsf{Gen}$,

$$\Pr_{(c,K) \xleftarrow{\$} \mathsf{Encaps}(pk)} [\mathsf{Decaps}(sk, c) = K] = 1 \ .$$

To KEM we associate the two sets $\mathcal{SK} := \{sk \mid (sk, pk) \in \mathsf{Gen}\}$ and $\mathcal{PK} := \{pk \mid (sk, pk) \in \mathsf{Gen}\}$. We assume (w.l.o.g.) that there is a function $\mu : \mathcal{SK} \to \mathcal{PK}$ such that for all $(sk, pk) \in \mathsf{Gen}$ it holds $\mu(sk) = pk$. We further define $\mathcal{PK}'$ to be the set of all efficiently recognizable public keys (by Encaps), i.e., $\mathcal{PK}' := \{pk \in \{0,1\}^* \mid \perp \notin \mathsf{Encaps}(pk)\}$. Note that $\mathcal{PK} \subseteq \mathcal{PK}'$ by correctness, but $\mathcal{PK}'$ could potentially contain "benign looking" public keys outside of $\mathcal{PK}$. We will also require a property of the KEM called η-key spreadness:

$$\forall pk \in \mathcal{PK}' : H_\infty(K \mid (c, K) \xleftarrow{\$} \mathsf{Encaps}(pk)) \geq \eta,$$

where H_∞ denotes the min-entropy. This property will assure that an honestly generated key K has sufficient min-entropy, even if it was generated using a pk outside $\mathcal{PK}$.

Privacy in the sense of multi-user chosen-ciphertext security is defined via the game in Listing 1. The advantage of an adversary $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{(n,q_d,q_c)\text{-CCA}}_{\mathcal{A},\mathsf{KEM}} := \left| \Pr[(n, q_d, q_c)\text{-CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right| .$$

Next, we recall syntax and security of an authenticated KEM (AKEM) [1].

Definition 2 (AKEM). *An authenticated key encapsulation mechanism* AKEM *consists of three algorithms:*

- Gen *outputs a key pair* (sk, pk)*, where* pk *defines a key space* $\mathcal{K}$*.*
- AuthEncap *takes as input a (sender) secret key* sk *and a (receiver) public key* pk*, and outputs an encapsulation* c *and a shared secret* $K \in \mathcal{K}$ *(or* $\perp$*).*

Listing 1: Game (n, q_d, q_c)-CCA for KEM. Adversary $\mathcal{A}$ makes at most q_d queries to DECAP, and at most q_c queries to CHALL.

```
(n, q_d, q_c)-CCA
01 for i ∈ [n]
02   (sk_i, pk_i) ←$ Gen
03 E ← ∅
04 b ←$ {0,1}
05 b' ←$ A^{DECAP,CHALL}(pk_1, ..., pk_n)
06 return [[b = b']]

Oracle DECAP(j ∈ [n], c)
07 if ∃K : (pk_j, c, K) ∈ E
08   return K
09 K ← Decaps(sk_j, c)
10 return K

Oracle CHALL(j ∈ [n])
11 (c, K) ←$ Encaps(pk_j)
12 if b = 1
13   K ←$ K
14   E ← E ∪ {(pk_j, c, K)}
15 return (c, K)
```

– *Deterministic* AuthDecap *takes as input a (sender) public key pk, a (receiver) secret key sk, and an encapsulation c, and outputs a shared key $K \in \mathcal{K}$ (or $\perp$).*

We require that for all $(sk_1, pk_1) \in \mathsf{Gen}, (sk_2, pk_2) \in \mathsf{Gen}$,

$$\Pr_{(c,K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_1, pk_2)} [\mathsf{AuthDecap}(pk_1, sk_2, c) = K] = 1 \ .$$

Sets $\mathcal{SK}$, $\mathcal{PK}$, $\mathcal{PK}'$, function μ, and η-key spreadness are defined as in the the KEM case.

Privacy (in the sense of insider and outsider CCA security) is defined via game Listing 2. Oracles AENCAP and ADECAP can be called with arbitrary public keys $pk \in \mathcal{PK}' \supseteq \mathcal{PK}$, i.e., arbitrary strings that pass AKEM's internal verification check. The insider setting is modeled using the REPSK oracle which can be used by the adversary to corrupt a sender's secret key sk. (Here we can assume $sk \in \mathcal{SK}$ since secret keys are usually seeds and can be efficiently verified.)

Note that this security notion is equivalent to the one from [1]. In the outsider case, the adversary cannot corrupt secret keys, i.e., $r_{sk} = 0$. The advantage is defined as

$$\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{sk})\text{-Insider-CCA}}_{\mathcal{A},\mathsf{AKEM}} := \left| \Pr[(n, q_e, q_d, q_c, r_{sk})\text{-Insider-CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right|,$$

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Outsider-CCA}}_{\mathcal{A},\mathsf{AKEM}} := \mathsf{Adv}^{(n,q_e,q_d,q_c,0)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{AKEM}}.$$

Authenticity is defined via the game in Listing 3.

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{AKEM}} := \left| \Pr[(n, q_e, q_d)\text{-Outsider-Auth}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right|.$$

2.3 Authenticated Public Key Encryption

We recall syntax and security of an authenticated PKE (APKE) [1].

Listing 2: Game $(n, q_e, q_d, q_c, r_{sk})$-Insider-CCA for AKEM. Adversary $\mathcal{A}$ makes at most q_e queries to AEncap, at most q_d queries to ADecap, at most q_c queries to Chall, and at most r_{sk} queries to RepSK.

$(n, q_e, q_d, q_c, r_{sk})$-Insider-CCA

$(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
$\mathcal{E}, \Gamma_{\mathtt{pk}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap},\text{Chall},\text{RepSK}}(pk_1, \ldots, pk_n)$
return $[\![b = b']\!]$

Oracle AEncap$(i \in [n], pk \in \mathcal{PK}')$

$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
return (c, K)

Oracle ADecap$(pk \in \mathcal{PK}', j \in [n], c)$

if $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
  return K
$K \leftarrow \mathsf{AuthDecap}(pk, sk_j, c)$
return K

Oracle Chall$(i \in [n], j \in [n])$

if $j \in \Gamma_{\mathtt{pk}}$
  return $\perp$
$(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk_j)$
if $b = 1$
  $K \xleftarrow{\$} \mathcal{K}$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, c, K)\}$
return (c, K)

Oracle RepSK$(i \in [n], sk \in \mathcal{SK})$

$(pk_i, sk_i) \leftarrow (\mu(sk), sk)$
$\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{i\}$

Definition 3 (APKE)**.** *An authenticated public key encryption scheme* APKE *consists of the following three algorithms:*

- Gen *outputs a key pair* (sk, pk).
- AuthEnc *takes as input a (sender) secret key* sk*, a (receiver) public key* pk*, a message* m*, associated data* aad*, a bitstring* $info$*, and outputs a ciphertext* c.
- *Deterministic* AuthDec *takes as input a (receiver) secret key* sk*, a (sender) public key* pk*, a ciphertext* c*, associated data* aad *and a bitstring* $info$*, and outputs a message* m.

We require that for all messages $m \in \{0,1\}^*$, $aad \in \{0,1\}^*$, $info \in \{0,1\}^*$,

$$\Pr_{\substack{(sk_S, pk_S) \xleftarrow{\$} \mathsf{Gen} \\ (sk_R, pk_R) \xleftarrow{\$} \mathsf{Gen}}} \left[\begin{matrix} c \leftarrow \mathsf{AuthEnc}(sk_S, pk_R, m, aad, info), \\ \mathsf{AuthDec}(sk_R, pk_S, c, aad, info) = m \end{matrix}\right] = 1 \ .$$

Sets $\mathcal{SK}$, $\mathcal{PK}$, $\mathcal{PK}'$, function μ, and η-key spreadness are defined as in the the KEM case.

Privacy. We define the game (n, q_e, q_d, q_c)-Insider-CCA in Listing 4, which is the strongest privacy notion for APKE defined in [1].

The advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{APKE}} := \left| \Pr[(n, q_e, q_d, q_c)\text{-Insider-CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right| \ .$$

Listing 3: Game (n, q_e, q_d)-Outsider-Auth for AKEM. Adversary $\mathcal{A}$ makes at most q_e queries to AEncap, and at most q_d queries to ADecap.

(n, q_e, q_d)-Outsider-Auth

```
01 for i ∈ [n]
02   (sk_i, pk_i) ←$ Gen
03 E ← ∅
04 b ←$ {0,1}
05 b' ←$ A^{AEncap,ADecap}(pk_1, ..., pk_n)
06 return [[b = b']]
```

Oracle AEncap$(i \in [n], pk \in \mathcal{PK}')$

```
07 (c, K) ←$ AuthEncap(sk_i, pk)
08 E ← E ∪ {(pk_i, pk, c, K)}
09 return (c, K)
```

Oracle ADecap$(pk \in \mathcal{PK}', j \in [n], c)$

```
10 if ∃K : (pk, pk_j, c, K) ∈ E
11   return K
12 K ← AuthDecap(pk, sk_j, c)
13 if b = 1 ∧ pk ∈ {pk_1, ..., pk_n} ∧ K ≠ ⊥
14   K ←$ K
15   E ← E ∪ {(pk, pk_j, c, K)}
16 return K
```

Authenticity. Furthermore, in Listing 5 we recall the (n, q_e, q_d)-Outsider-Auth game from [1]. The advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{APKE}} := \Pr[(n, q_e, q_d)\text{-Outsider-Auth} \Rightarrow 1].$$

Note that in contrast to the privacy case we use the weaker outsider notion instead of the insider notion for authenticity. This is because [1] show that the $\mathsf{HPKE}_{\mathsf{Auth}}$ construction cannot fulfill insider authenticity for any possible instantiation. Since the same attack can be run against $\mathsf{HPKE}_{\mathsf{PSK}}$, we omit the definition here.

2.4 Pseudorandom Functions

A keyed function F with a finite key space $\mathcal{K}$, input length n, and a finite output range $\mathcal{R}$ is a function $F : \mathcal{K} \times \{0,1\}^* \to \mathcal{R}$.

Definition 4 (Multi-Instance Pseudorandom Function). *The (n_k, q_{PRF})-PRF advantage of an adversary $\mathcal{A}$ against a keyed function F with finite key space $\mathcal{K}$, and finite range $\mathcal{R}$ is defined as*

$$\mathsf{Adv}^{(n_k,q_{\mathsf{PRF}})\text{-PRF}}_{\mathcal{A},F} := \left| \Pr_{K_1,\ldots,K_{n_k} \xleftarrow{\$} \mathcal{K}} [\mathcal{A}^{F(K_1,\cdot),\ldots,F(K_{n_k},\cdot)}] - \Pr[\mathcal{A}^{f_1(\cdot),\ldots,f_{n_k}(\cdot)}] \right| ,$$

where $f_i : \{0,1\}^* \to \mathcal{R}$ for $i \in [n_k]$ are chosen uniformly at random from the set of functions mapping to $\mathcal{R}$ and $\mathcal{A}$ makes at most q_{PRF} queries in total to the oracles $F(K_i, \cdot)$, f_i resp.

Definition 5 (2-Keyed Function). *A 2-keyed function F with finite key spaces $\mathcal{K}_1$ and $\mathcal{K}_2$, and finite range $\mathcal{R}$ is a function*

$$F : \mathcal{K}_1 \times \mathcal{K}_2 \times \{0,1\}^* \to \mathcal{R}.$$

Listing 4: Game (n, q_e, q_d, q_c)-Insider-CCA for APKE in which adversary $\mathcal{A}$ makes at most q_e queries to AEnc, at most q_d queries to ADec and at most q_c queries to Chall.

(n, q_e, q_d, q_c)-Insider-CCA
01 **for** $i \in [n]$
02 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec},\text{Chall}}(pk_1, \ldots, pk_n)$
06 **return** $[\![b = b']\!]$

Oracle AEnc$(i \in [n], pk \in \mathcal{PK}', m, aad, info)$
07 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, aad, info)$
08 **return** c

Oracle ADec$(pk \in \mathcal{PK}', j \in [n], c, aad, info)$
09 **if** $(pk, pk_j, c, aad, info) \in \mathcal{E}$
10 $\quad$ **return** $\perp$
11 $m \leftarrow \mathsf{AuthDec}(pk, sk_j, c, aad, info)$
12 **return** m

Oracle Chall$(sk \in \mathcal{SK}, j \in [n], m_0, m_1, aad, info)$
13 **if** $|m_0| \neq |m_1|$ **return** $\perp$
14 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk, pk_j, m_b, aad, info)$
15 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mu(sk), pk_j, c, aad, info)\}$
16 **return** c

Listing 5: Game (n, q_e, q_d)-Outsider-Auth for APKE in which adversary $\mathcal{A}$ makes at most q_e queries to AEnc and at most q_d queries to ADec.

(n, q_e, q_d)-Outsider-Auth
01 **for** $i \in [n]$
02 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E} \leftarrow \emptyset$
04 $(i^*, j^*, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec}}(pk_1, \ldots, pk_n)$
05 **return** $[\![(pk_{i^*}, pk_{j^*}, c^*, aad^*, info^*) \notin \mathcal{E}$
$\quad \wedge \mathsf{AuthDec}(pk_{i^*}, sk_{j^*}, c^*, aad^*, info^*) \neq \perp]\!]$

Oracle AEnc$(i \in [n], pk \in \mathcal{PK}', m, aad, info)$
06 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, aad, info)$
07 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, aad, info)\}$
08 **return** c

Oracle ADec$(pk \in \mathcal{PK}', j \in [n], c, aad, info)$
09 $m \leftarrow \mathsf{AuthDec}(pk, sk_j, c, aad, info)$
10 **return** m

2.5 Authenticated Encryption with Associated Data

We recall standard syntax and security for AEAD schemes.

Definition 6 (AEAD). *A nonce-based authenticated encryption scheme with associated data and key space $\mathcal{K}'$ consists of the following two algorithms:*

- *Deterministic algorithm* AEAD.Enc *takes as input a key $k \in \mathcal{K}'$, a message m, associated data aad, and a nonce and outputs a ciphertext c.*
- *Deterministic algorithm* AEAD.Dec *takes as input a key $k \in \mathcal{K}'$, a ciphertext c, associated data aad and a nonce $nonce$ and outputs a message m or the failure symbol $\perp$.*

We require that for all $aad \in \{0,1\}^*, m \in \{0,1\}^*, nonce \in \{0,1\}^{N_{nonce}}$

$$\Pr_{k \xleftarrow{\$} \mathcal{K}'}[\mathsf{AEAD.Dec}(k, \mathsf{AEAD.Enc}(k, m, aad, nonce), aad, nonce) = m] = 1 \ ,$$

where N_{nonce} is the length of the nonce in bits.

We define the multi-instance security game (n, q_d)-INT-CTXT in Listing 6 and (n, q_d)-CCA in Listing 7. Note that an AEAD scheme which is IND-CPA and INT-CTXT secure is also CCA secure [5]. The advantage of an adversary $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_d)\text{-INT-CTXT}}_{\mathcal{A},\mathsf{AEAD}} := \left| \Pr[(n, q_d)\text{-INT-CTXT}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right|$$

$$\mathsf{Adv}^{(n,q_d)\text{-CCA}}_{\mathcal{A},\mathsf{AEAD}} := \left| \Pr[(n, q_d)\text{-CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right| .$$

Listing 6: Game (n, q_d)-INT-CTXT for AEAD. Adversary $\mathcal{A}$ makes at most one query per index i to Enc and at most q_d queries in total to Dec.

```
(n, q_d)-INT-CTXT
01 for i ∈ [n]
02   k_i ←$ K'
03   nonce_i ←$ {0,1}^{N_nonce}
04 E ← ∅
05 b ←$ {0,1}
06 b' ←$ A^{Enc,Dec}
07 return [[b = b']]

Oracle Enc(i ∈ [n], m, aad)
08 c ← AEAD.Enc(k_i, m, aad, nonce_i)
09 E ← E ∪ {i, m, c, aad}
10 return (c, nonce_i)

Oracle Dec(i ∈ [n], c, aad)
11 if b = 0
12   m ← AEAD.Dec(k_i, c, aad, nonce_i)
13 else if ∃m' : (i, m', c, aad) ∈ E
14   m ← m'
15 else
16   m ← ⊥
17 return m
```

Listing 7: Game (n, q_d)-CCA for AEAD. Adversary $\mathcal{A}$ makes at most one query per index i to Enc and at most q_d queries in total to Dec.

```
(n, q_d)-CCA
01 for i ∈ [n]
02   k_i ←$ K'
03   nonce_i ←$ {0,1}^{N_nonce}
04 E ← ∅
05 b ←$ {0,1}
06 b' ←$ A^{Enc,Dec}
07 return [[b = b']]

Oracle Enc(i ∈ [n], m_0, m_1, aad)
08 c ← AEAD.Enc(k_i, m_b, aad, nonce_i)
09 E ← E ∪ {i, c, aad}
10 return (c, nonce_i)

Oracle Dec(i ∈ [n], c, aad)
11 if (i, c, aad) ∈ E
12   return ⊥
13 m ← AEAD.Dec(k_i, c, aad, nonce_i)
14 return m
```

2.6 Digital Signatures

Definition 7 (Signature Scheme). *A signature scheme* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *consists of three algorithm:*

- *Key generation* Gen *generates a secret signing key sigk and a verification key vk.*
- *Signing* Sign*: On input a signing key sigk and a message m, outputs a signature* σ*.*
- *Verification* Vfy*: On input a verification key vk, a message m, and a signature* σ*, deterministically outputs a bit b.*

The signature scheme fulfills correctness if for all $(sigk, vk) \xleftarrow{\$} \mathsf{Gen}$ *it holds*

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{Sign}(sk,m)} [\mathsf{Vfy}(vk, m, \sigma) = 1] = 1.$$

To SIG we associate the two sets $\mathcal{SK} := \{sigk \mid (sigk, vk) \in \mathsf{Gen}\}$ and $\mathcal{VK} := \{vk \mid (sigk, vk) \in \mathsf{Gen}\}$. We assume (w.l.o.g.) that there is a function $\mu' : \mathcal{SK} \to \mathcal{VK}$ such that for all $(sigk, vk) \in \mathsf{Gen}$ it holds $\mu'(sigk) = vk$.

Definition 8 (Strong Unforgeability). *Let* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *be a signature scheme. We define multi-user strong unforgeability against a chosen message attack*

*(*SUF-CMA*) via the* (n, q_s)-SUF-CMA *game in Listing 8. The advantage of an adversary* $\mathcal{A}$ *is* $\mathsf{Adv}^{(n,q_s)\text{-SUF-CMA}}_{\mathcal{A},\mathsf{SIG}} = \Pr[(n, q_s)\text{-SUF-CMA}(\mathcal{A}) \Rightarrow 1]$.

Listing 8: Game (n, q_s)-SUF-CMA for SIG. Adversary $\mathcal{A}$ makes at most q_s queries to Sign.

(n, q_s)-SUF-CMA	Oracle $\textsc{Sign}(i \in [n], m)$
01 **for** $i \in [n]$	06 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, m)$
02 $\quad (sigk_i, vk_i) \xleftarrow{\$} \mathsf{Gen}$	07 $Q \leftarrow Q \cup \{(i, m, \sigma)\}$
03 $Q \leftarrow \emptyset$	08 **return** σ
04 $(i^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Sign}}(vk_1, \ldots, vk_n)$	
05 **return** $[\![\mathsf{Vfy}(vk_{i^*}, m^*, \sigma^*) = 1$	
$\quad \wedge (i^*, m^*, \sigma^*) \notin Q]\!]$	

2.7 Non-Interactive Key Exchange

Definition 9 (Non-Interactive Key Exchange). *A* NIKE *scheme consists of a setup, two algorithms* NIKE.KeyGen, NIKE.SharedKey *and a shared key space* $\mathcal{SHK}$*. The algorithms are defined as follows:*

- NIKE.KeyGen *outputs a pair of public and secret key* (sk, pk)*.*
- NIKE.SharedKey *takes a secret key* sk *and a public key* pk *and outputs either a shared key in* $\mathcal{SHK}$ *or the failure symbol* $\perp$*.*

The NIKE *fulfills correctness if for all* $(sk_1, pk_1) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$, $(sk_2, pk_2) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$, *it holds*

$$\mathsf{NIKE.SharedKey}(sk_1, pk_2) = \mathsf{NIKE.SharedKey}(sk_2, pk_1).$$

Sets $\mathcal{SK}$, $\mathcal{PK}$, $\mathcal{PK}'$, and function μ are defined as in the the KEM case. We define the entropy of the public key as

$$P_{\mathsf{NIKE}} := \max_{pk} \Pr[pk = pk' \mid (sk', pk') \xleftarrow{\$} \mathsf{NIKE.KeyGen}].$$

Note that this statistical property can be upper bounded by the passive security of NIKE.

The security definition for actively secure NIKE can be found in Appendix A.1.

3 Pre-Shared Key (Authenticated) Encryption

In this section, we define syntax and security of pre-shared key public key encryption (pskPKE) and pre-shared key authenticated public key encryption (pskAPKE). The former is an extension of common public key encryption with an additional pre-shared symmetric key that has already been shared between the parties. The latter is an analogue extension of authenticated public key encryption (APKE). The pre-shared key (psk) has two functionalities. First, it provides an additional layer of privacy since the security does not have to rely on the asymmetric key only. Second, it also provides authenticity. The

intuition behind security is that even if one of the two keys, either the asymmetric key or the (symmetric) pre-shared key, is corrupted or in any other way insecure, the scheme should still guarantee security.

We start with defining the syntax of pskPKE and pskAPKE in Section 3.1 and then define the security model for privacy (Section 3.2) and authenticity (Section 3.3).

3.1 Syntax

Definition 10 (pskPKE). *A pre-shared key public key encryption scheme* pskPKE *consists of the following four algorithms:*

- GenSK *outputs a key pair* (sk, pk).
- GenPSK *outputs a pre-shared key* psk.
- pskEnc *takes as input a (receiver) public key* pk*, a pre-shared key* psk*, a message* m*, associated data* aad*, a bitstring* $info$*, and outputs a ciphertext* c.
- *Deterministic* pskDec *takes as input a (receiver) secret key* sk*, a pre-shared key* psk*, a ciphertext* c*, associated data* aad *and a bitstring* $info$*, and outputs a message* m.

We require that for all messages $m \in \{0,1\}^*$, $aad \in \{0,1\}^*$, $info \in \{0,1\}^*$,

$$\Pr_{\substack{(sk,pk)\xleftarrow{\$}\mathsf{GenSK}\\ psk\xleftarrow{\$}\mathsf{GenPSK}}}\begin{bmatrix} c \leftarrow \mathsf{pskEnc}(pk, psk, m, aad, info),\\ \mathsf{pskDec}(sk, psk, c, aad, info) = m \end{bmatrix} = 1 \ .$$

Definition 11 (pskAPKE). *A pre-shared key Authenticated public key encryption scheme* pskAPKE *consists of the following four algorithms:*

- GenSK *outputs a key pair* (sk, pk).
- GenPSK *outputs a pre-shared key* psk.
- pskAEnc *takes as input a (sender) secret key* sk*, a (receiver) public key* pk*, a pre-shared key* psk*, a message* m*, associated data* aad*, a bitstring* $info$*, and outputs a ciphertext* c.
- *Deterministic* pskADec *takes as input a (sender) public key* pk*, a (receiver) secret key* sk*, a pre-shared key* psk*, a ciphertext* c*, associated data* aad *and a bitstring* $info$*, and outputs a message* m.

We require that for all messages $m \in \{0,1\}^*$, $aad \in \{0,1\}^*$, $info \in \{0,1\}^*$,

$$\Pr_{\substack{(sk_S,pk_S)\xleftarrow{\$}\mathsf{GenSK}\\ (sk_R,pk_R)\xleftarrow{\$}\mathsf{GenSK}\\ psk\xleftarrow{\$}\mathsf{GenPSK}}}\begin{bmatrix} c \leftarrow \mathsf{pskAEnc}(sk_S, pk_R, psk, m, aad, info),\\ \mathsf{pskADec}(pk_S, sk_R, psk, c, aad, info) = m \end{bmatrix} = 1 \ .$$

Sets $\mathcal{SK}$, $\mathcal{PK}$, $\mathcal{PK}'$, and function μ are defined as in the the KEM case.

Listing 9: Game $(n, q_e, q_d, q_c, r_{pk}, r_{psk})$-CCA for pskPKE. Adversary $\mathcal{A}$ makes at most q_e queries to Enc, at most q_d queries to Dec, at most q_c queries to Chall, at most r_{pk} queries to RepPK, and at most r_{psk} queries to RepPSK.

$(n, q_e, q_d, q_c, r_{pk}, r_{psk})$-CCA

for $i \in [n]$
  $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
  for $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chall},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
return $[\![b = b']\!]$

Oracle Enc$(i \in [n], j \in [n], m, aad, info)$

$c \xleftarrow{\$} \mathsf{pskEnc}(pk_j, psk_{ij}, m, aad, info)$
return c

Oracle Dec$(i \in [n], j \in [n], c, aad, info)$

if $sk_j = \bot \vee (pk_j, psk_{ij}, c, aad, info) \in \mathcal{E}$
  return $\bot$
$m \leftarrow \mathsf{pskDec}(sk_j, psk_{ij}, c, aad, info)$
return m

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$

if $|m_0| \neq |m_1| \vee (j \in \Gamma_{\mathtt{pk}} \wedge (i,j) \in \Gamma_{\mathtt{psk}})$
  return $\bot$
$c \xleftarrow{\$} \mathsf{pskEnc}(pk_j, psk_{ij}, m_b, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, c, aad, info)\}$
return c

Oracle RepPK$(j \in [n], pk \in \mathcal{PK}')$

$(sk_j, pk_j) \leftarrow (\bot, pk)$
$\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle RepPSK$(i \in [n], j \in [n], psk)$

$psk_{ij} \leftarrow psk$
$psk_{ji} \leftarrow psk$
$\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j),(j,i)\}$

Listing 10: Game $(n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})$-Insider-CCA for pskAPKE. Adversary $\mathcal{A}$ makes at most q_e queries to Enc at most q_d queries to Dec, at most q_c queries to Chall, at most r_{pk} queries to RepPK, at most r_{sk} queries to RepSK, and at most r_{psk} queries to RepPSK.

$(n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})$-Insider-CCA

for $i \in [n]$
  $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
  for $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \Gamma_c, \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chall},\textsc{RepPK},\textsc{RepSK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
return $[\![b = b']\!]$

Oracle Enc$(i \in [n], j \in [n], m, aad, info)$

if $sk_i = \bot$
  return $\bot$
$c \xleftarrow{\$} \mathsf{pskAEnc}(sk_i, pk_j, psk_{ij}, m, aad, info)$
return c

Oracle Dec$(i \in [n], j \in [n], c, aad, info)$

if $sk_j = \bot \vee (pk_j, psk_{ij}, c, aad, info) \in \mathcal{E}$
  return $\bot$
$m \leftarrow \mathsf{pskADec}(pk_i, sk_j, psk_{ij}, c, aad, info)$
return m

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$

if $|m_0| \neq |m_1| \vee sk_i = \bot \vee (j \in \Gamma_{\mathtt{pk}} \wedge (i,j) \in \Gamma_{\mathtt{psk}})$
  return $\bot$
$c \xleftarrow{\$} \mathsf{pskAEnc}(sk_i, pk_j, psk_{ij}, m_b, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, psk_{ij}, c, aad, info)\}$
return c

Oracle RepPK$(j \in [n], pk \in \mathcal{PK}')$

$(sk_j, pk_j) \leftarrow (\bot, pk)$
$\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle RepSK$(j \in [n], sk \in \mathcal{SK})$

$(sk_j, pk_j) \leftarrow (sk, \mu(sk))$
$\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle RepPSK$(i \in [n], j \in [\ell], psk)$

$psk_{ij} \leftarrow psk$
$psk_{ji} \leftarrow psk$
$\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j),(j,i)\}$

Listing 11: Game $(n, q_e, q_d, r_{pk}, r_{psk})$-Auth for pskPKE. Adversary $\mathcal{A}$ makes at at most q_e queries to Enc, at most q_d queries to Dec, at most r_{pk} queries to RepPK, and at most r_{psk} queries to RepPSK.

$(n, q_e, q_d, r_{pk}, r_{psk})$-Auth

for $i \in [n]$
  $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
  for $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \Gamma_{\mathsf{psk}} \leftarrow \emptyset$
$(i^*, j^*, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
return $[\![(i^*, j^*) \notin \Gamma_{\mathsf{psk}} \wedge sk_{j^*} \neq \bot$
  $\wedge\, (pk_{j^*}, psk_{i^*j^*}, c^*, aad^*, info^*) \notin \mathcal{E}$
  $\wedge\, \mathsf{pskDec}(sk_{j^*}, psk_{i^*j^*}, c^*, aad^*, info^*) \neq \bot]\!]$

Oracle $\textsc{Enc}(i \in [n], j \in [n], m, aad, info)$

$c \xleftarrow{\$} \mathsf{pskEnc}(pk_j, psk_{ij}, m, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, c, aad, info)\}$
return c

Oracle $\textsc{Dec}(i \in [n], j \in [n], c, aad, info)$

if $sk_j = \bot$
  return $\bot$
$m \leftarrow \mathsf{pskDec}(sk_j, psk_{ij}, c, aad, info)$
return m

Oracle $\textsc{RepPK}(j \in [n], pk \in \mathcal{PK}')$

$(sk_j, pk_j) \leftarrow (\bot, pk)$

Oracle $\textsc{RepPSK}(i \in [n], j \in [n], psk)$

$psk_{ij} \leftarrow psk$
$psk_{ji} \leftarrow psk$
$\Gamma_{\mathsf{psk}} \leftarrow \Gamma_{\mathsf{psk}} \cup \{(i, j), (j, i)\}$

Listing 12: Game $(n, q_e, q_d, r_{pk}, r_{psk})$-Outsider-Auth for pskAPKE. Adversary $\mathcal{A}$ makes at most q_e queries to Enc, at most q_d queries to Dec, at most r_{pk} queries to RepPK, and at most r_{psk} queries to RepPSK.

$(n, q_e, q_d, r_{pk}, r_{psk})$-Outsider-Auth

for $i \in [n]$
  $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
  for $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \Gamma_{\mathsf{pk}}, \Gamma_{\mathsf{psk}} \leftarrow \emptyset$
$(i^*, j^*, c^*, aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
return $[\![(i^* \notin \Gamma_{\mathsf{pk}} \vee (i^*, j^*) \notin \Gamma_{\mathsf{psk}}) \wedge sk_{j^*} \neq \bot$
  $\wedge\, (pk_{i^*}, pk_{j^*}, psk_{i^*j^*}, c^*, aad^*, info^*) \notin \mathcal{E}$
  $\wedge\, \mathsf{pskADec}(pk_{i^*}, sk_{j^*}, psk_{i^*j^*}, c^*, aad^*, info^*) \neq \bot]\!]$

Oracle $\textsc{Enc}(i \in [n], j \in [n], m, aad, info)$

if $sk_i = \bot$
  return $\bot$
$c \xleftarrow{\$} \mathsf{pskAEnc}(sk_i, pk_j, psk_{ij}, m, aad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, psk_{ij}, c, aad, info)\}$
return c

Oracle $\textsc{Dec}(i \in [n], j \in [n], c, aad, info)$

if $sk_j = \bot$
  return $\bot$
$m \leftarrow \mathsf{pskADec}(pk_i, sk_j, psk_{ij}, c, aad, info)$
return m

Oracle $\textsc{RepPK}(i \in [n], pk \in \mathcal{PK}')$

$(sk_i, pk_i) \leftarrow (\bot, pk)$
$\Gamma_{\mathsf{pk}} \leftarrow \Gamma_{\mathsf{pk}} \cup \{i\}$

Oracle $\textsc{RepPSK}(i \in [n], j \in [n], psk)$

$psk_{ij} \leftarrow psk$
$psk_{ji} \leftarrow psk$
$\Gamma_{\mathsf{psk}} \leftarrow \Gamma_{\mathsf{psk}} \cup \{(i, j), (j, i)\}$

3.2 Privacy

Privacy is defined via the games in Listing 9 (pskPKE) and Listing 10 (pskAPKE). The idea is based on the standard CCA definition for PKE with the following modifications.

For the game in Listing 9, the adversary is provided with an encryption oracle Enc since, in contrast to standard PKE, encryption requires the knowledge of the corresponding pre-shared key. We further strengthen the security model by allowing corrupted keys. This is modeled by two oracles, RepPK and RepPSK. Oracle RepPK models the corruption of an asymmetric key pair, where the adversary is allowed to replace the public key of a user with $pk \in \mathcal{PK}'$ (with or without knowing the matching private key). Since the game is not able to decrypt queries to a corrupted receiver's key anymore, we return $\perp$ on such queries. To keep track of corrupted keys, the game maintains a set $\Gamma_{\mathtt{pk}}$ containing all corrupted indices j. Similarly, oracle RepPSK models the corruption of a pre-shared key between two chosen users. Set $\Gamma_{\mathtt{psk}}$ keeps track of sender and receiver pairs for which the corresponding *psk* was corrupted. Provided with the additional oracles, it should still be hard to guess the challenge bit, i.e., which of the two messages given to the challenge oracle was encrypted. To avoid trivial wins, we disallow the challenge oracle to be queried on pairs of users for which both the receiver's key as well as the pre-shared key were corrupted. However, the challenge query is allowed if at most one of them is corrupted. (Note that in particular the adversary is still allowed to issue challenge queries for a corrupted *psk*.) In that sense we model an "insider setting". An insider notion for pskPKE cannot be formulated stronger because it does not make any sense to allow corrupted senders in the sense of a corrupted *sk* since it is not used for encryption.[5] The advantage of adversary $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{pk},r_{psk})\text{-}\mathsf{CCA}}_{\mathcal{A},\mathsf{pskPKE}} := \left|\Pr[(n,q_e,q_d,q_c,r_{pk},r_{psk})\text{-}\mathsf{CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right|.$$

The definition of CCA security for pskAPKE (Listing 10) works similar with the following changes. The main difference compared to pskPKE is that the asymmetric part of the encryption is authenticated which means that the sender's secret key is also involved. To take attacks into account which could make use of corrupted senders, we make the following modification. The game provides another oracle RepSK which allows the adversary to replace an asymmetric secret key directly, which means they are also allowed to query all the other oracles on corrupted sender keys. This is exactly what is called the "Insider" setting in [1] and signcryption [10]. This means in particular that the encryption of two different messages is indistinguishable even if the sender's asymmetric key was adversarially chosen. As in the case of pskPKE, the *psk* can also be corrupted in the sense of an insider attack, i.e., the sender is corrupted. However, due to the symmetric nature of the pre-shared key, this implies a security loss for the receiver as well and we have to exclude trivial wins in the same way as for pskPKE. Therefore, the same security requirement as for pskPKE holds, i.e., in addition to the corruption of a sender's secret key either a receiver's key or the pre-shared key can be corrupted and security still holds. We also define outsider security as the simplified setting, where the adversary does not have access to the RepSK

[5] To prevent confusion we do explicitly call this notion insider secure and only use the term "insider" if it is possible to the (asymmetric) secret key of a sender. See also the security definition of the pskAPKE.

oracle, i.e., $r_{pk} = 0$. The advantage of adversary $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{pk},r_{sk},r_{psk})\text{-Insider-CCA}}_{\mathcal{A},\mathsf{pskAPKE}} := \left|\Pr[(n,q_e,q_d,q_c,r_{pk},r_{sk},r_{psk})\text{-Insider-CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right|,$$

$$\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{pk},r_{psk})\text{-Outsider-CCA}}_{\mathcal{A},\mathsf{pskAPKE}} := \mathsf{Adv}^{(n,q_e,q_d,q_c,r_{pk},0,r_{psk})\text{-Insider-CCA}}_{\mathcal{A},\mathsf{pskAPKE}}.$$

3.3 Authenticity

Authenticity is defined via the games in Listing 11 (pskPKE) and Listing 12 (pskAPKE). For the game in Listing 11, the goal is to forge a fresh ciphertext, i.e., one that was not output by the encryption oracle and that does not decrypt to $\perp$. Further, we do not allow corrupted receiver keys for the challenge, i.e. $sk_{j^*} \neq \perp$, since the decryption would not be possible anymore. As in the privacy case, we model corrupted keys via oracle RepPK which allows the adversary to replace an asymmetric key and oracle RepPSK which allows to replace a pre-shared key. Due to the structure of a pskPKE, authenticity cannot rely on the asymmetric keys since the sender needs no secret material for the encryption. Hence, authenticity can only be achieved via the pre-shared key. To avoid trivial wins, the game excludes forgeries for which the corresponding pre-shared key was corrupted, i.e., replaced via oracle RepPSK. This means, an adversary wins if they can forge a new valid ciphertext for which the pre-shared key was not corrupted given encryption and decryption oracles as well corruption oracles for both the keys. The advantage of adversary $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_e,q_d,r_{pk},r_{psk})\text{-Auth}}_{\mathcal{A},\mathsf{pskPKE}} := \Pr[(n,q_e,q_d,r_{pk},r_{psk})\text{-Auth}(\mathcal{A}) \Rightarrow 1].$$

For the Auth security of a pskAPKE (Listing 12) there is only a slight modification. Encryption pskAEnc also inputs the asymmetric sender's secret key, which is why the authenticity can now also rely on the sender's asymmetric key and not only on the pre-shared key. In the game, this is used for a stronger notion which considers corruptions of the sender's key and corruptions of the pre-shared key. The adversary's forgery is accepted if at most one of the keys was corrupted. This means, it should be hard to forge a fresh ciphertext even if the sender's key or the pre-shared key were corrupted but not both. The advantage of adversary $\mathcal{A}$ is

$$\mathsf{Adv}^{(n,q_e,q_d,r_{pk},r_{psk})\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{pskAPKE}} := \Pr[(n,q_e,q_d,r_{pk},r_{psk})\text{-Outsider-Auth}(\mathcal{A}) \Rightarrow 1].$$

One could also define Insider-Auth security for pskAPKE by allowing the adversary to choose the secret key of the receiver of a forgery (and adjust the non-triviality condition accordingly). Since we do not use Insider-Auth for pskAPKE in the following, we omit the formal definition.

4 HPKE's constructions of a pskPKE and pskAPKE

4.1 Generic Constructions

We construct a pskPKE and a pskAPKE from an (authenticated) key encapsulation mechanism KEM (AKEM), a 2-keyed function KS (where KS_1 denotes KS keyed in the first input

and KS_2 keyed in the second input), and a nonce-based authenticated encryption with additional data scheme AEAD. More concretely, that is pskPKE[KEM, KS, AEAD] where pskPKE.GenSK = KEM.Gen, and the pre-shared key generation pskPKE.GenPSK samples a uniformly random element from the appropriate key space. Encryption and decryption are defined in Listing 13 and Listing 14, respectively.

Listing 13: Encryption and decryption functions of the pre-shared-key PKE scheme pskPKE[KEM, KS, AEAD], built from KEM, KS, and AEAD.

$\mathsf{pskEnc}(pk, psk, m, aad, info)$	$\mathsf{pskDec}(sk, psk, (c_1, c_2), aad, info)$
01 $(c_1, K) \xleftarrow{\$} \mathsf{Encaps}(pk)$	05 $K \leftarrow \mathsf{Decaps}(sk, c_1)$
02 $(k, nonce) \leftarrow \mathsf{KS}(K, psk, info)$	06 $(k, nonce) \leftarrow \mathsf{KS}(K, psk, info)$
03 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$	07 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
04 **return** (c_1, c_2)	08 **return** m

Listing 14: Encryption and decryption function of the pre-shared-key APKE scheme pskAPKE[AKEM, KS, AEAD], built from AKEM, KS, and AEAD.

$\mathsf{pskAEnc}(sk, pk, psk, m, aad, info)$	$\mathsf{pskADec}(pk, sk, psk, (c_1, c_2), aad, info)$
01 $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk, pk)$	05 $K \leftarrow \mathsf{AuthDecap}(pk, sk, c_1)$
02 $(k, nonce) \leftarrow \mathsf{KS}(K, psk, info)$	06 $(k, nonce) \leftarrow \mathsf{KS}(K, psk, info)$
03 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$	07 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
04 **return** (c_1, c_2)	08 **return** m

4.2 Security of pskPKE and pskAPKE

The following Theorems 1–4 state privacy and authenticity of our constructions pskPKE[KEM, KS, AEAD] and pskAPKE[AKEM, KS, AEAD].

Theorem 1 (KEM CCA + KS_1 PRF + KS_2 PRF + AEAD CCA $\Rightarrow$ pskPKE CCA)**.** *For any* $(n, q_e, q_d, q_c, r_{pk}, r_{psk})$-CCA *adversary* $\mathcal{A}$ *against* pskPKE[KEM, KS, AEAD]*, there exists an* (n, q_d, q_c)-CCA *adversary* $\mathcal{B}$ *against* KEM*, a* $(q_c, q_d + q_c)$-PRF *adversary* $\mathcal{C}_1$ *against* KS_1*, a* $(q_e + q_d + q_c, q_e + q_d + q_c)$-PRF *adversary* $\mathcal{C}_2$ *against* KS_2*, and a* $(q_e + q_c, q_d)$-CCA *adversary* $\mathcal{D}$ *against* AEAD *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{pk},r_{psk})\text{-CCA}}_{\mathcal{A},\mathsf{pskPKE[KEM,KS,AEAD]}} \leq & \mathsf{Adv}^{(n,q_d,q_c)\text{-CCA}}_{\mathcal{B},\mathsf{KEM}} + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1} \\ & + \mathsf{Adv}^{(q_e+q_d+q_c,q_e+q_d+q_c)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2} + \mathsf{Adv}^{(q_e+q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}} \\ & + \frac{q_e^2 + q_c^2 + q_e q_c}{2^\eta}.\end{aligned}$$

Proof (Sketch). To prove CCA security for the pskPKE, we use CCA security of the underlying KEM to replace the KEM keys with uniformly random values. Together with a uniformly random *psk*, they can be used as PRF keys. Depending on the challenge query and which keys were corrupted, we can use either the PRF property when keyed on the

first (KS_1) or the second input (KS_2). For the second input we also need the KEM key to have enough entropy to avoid collisions. This yields random symmetric keys and the theorem follows by the CCA of AEAD. The full proof can be found in Appendix B.1. □

Theorem 2 (AKEM Insider-CCA + KS_1 PRF + KS_2 PRF + AEAD CCA ⇒ pskAPKE Insider-CCA)**.** *For any* $(n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})$-Insider-CCA *adversary* $\mathcal{A}$ *against the scheme* pskAPKE[AKEM, KS, AEAD]*, there exists an* $(n, q_e, q_d, q_c, r_{sk})$-Insider-CCA *adversary* $\mathcal{B}$ *against* AKEM*, a* $(q_c, q_d + q_c)$-PRF *adversay* $\mathcal{C}_1$ *against* KS_1*, a* $(q_e + q_d + q_c, q_e + q_d + q_c)$-PRF *adversay* $\mathcal{C}_2$ *against* KS_2*, and a* $(q_e + q_c, q_d)$-CCA *adversary* $\mathcal{D}$ *against* AEAD *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{pk},r_{sk},r_{psk})\text{-Insider-CCA}}_{\mathcal{A},\mathsf{pskAPKE[AKEM,KS,AEAD]}} \leq & \mathsf{Adv}^{(n,q_e,q_d,q_c,r_{sk})\text{-Insider-CCA}}_{\mathcal{B},\mathsf{AKEM}} + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1} \\ & + \mathsf{Adv}^{(q_e+q_d+q_c,q_e+q_d+q_c)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2} \\ & + \mathsf{Adv}^{(q_e+q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}} + \frac{q_e^2 + q_c^2 + q_e q_c}{2^\eta}.\end{aligned}$$

Proof (Sketch). Since we want to achieve Insider-CCA security for the pskAPKE, we have to simulate the RepSK oracle which can be done by reducing to an Insider-CCA secure AKEM. The remaining part of the proof is essentially the same as for Theorem 1. The full proof can be found in Appendix B.2. □

Theorem 3 (KEM CCA + KS_1 PRF + KS_2 PRF + AEAD INT-CTXT ⇒ pskPKE Auth)**.** *For any* $(n, q_e, q_d, r_{pk}, r_{psk})$-Auth *adversary* $\mathcal{A}$ *against the scheme* pskPKE[KEM, KS, AEAD]*, there exists a* (n, q_d, q_e)-CCA *adversary* $\mathcal{B}$ *against* KEM*, a* (q_e, q_e)-PRF *adversary* $\mathcal{C}_1$ *against* KS_1*, a* (q_d, q_d)-PRF *adversary* $\mathcal{C}_2$ *against* KS_2*, and a* $(2q_e + q_d + 1, q_d + 1)$-INT-CTXT *adversary* $\mathcal{D}$ *against* AEAD *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,r_{pk},r_{psk})\text{-Auth}}_{\mathcal{A},\mathsf{pskPKE[KEM,KS,AEAD]}} \leq & \mathsf{Adv}^{(n,q_d,q_e)\text{-CCA}}_{\mathcal{B},\mathsf{KEM}} + \mathsf{Adv}^{(q_e,q_e)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1} + \mathsf{Adv}^{(q_d,q_d)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2} \\ & + \mathsf{Adv}^{(2q_e+q_d+1,q_d+1)\text{-INT-CTXT}}_{\mathcal{D},\mathsf{AEAD}} + \frac{q_e(q_e + q_d - 1)}{|\mathcal{K}|}.\end{aligned}$$

Proof (Sketch). We use the CCA security of KEM to ensure that the key K fed into KS_2 is uniform random such that, with high probability, there are no key collisions. Together with the pre-shared key, at least one of the inputs is uniformly random and the PRF property of KS yields a random output. Then, this output can be used for the AEAD such that decryption queries (with respect to an honest *psk*) can be rejected. The full proof can be found in Appendix B.3. □

Theorem 4 (AKEM Outsider-CCA + AKEM Outsider-Auth + KS_1 PRF + KS_2 PRF + AEAD INT-CTXT ⇒ pskAPKE Outsider-Auth)**.** *For any* $(n, q_e, q_d, r_{pk}, r_{psk})$-Outsider-Auth *adversary* $\mathcal{A}$ *against the scheme* pskAPKE[AKEM, KS, AEAD]*, there exists an* $(n, 0, q_d, q_c)$-CCA *adversary* $\mathcal{B}$ *against* AKEM*, a* $(q_e + q_d, q_e + q_d)$-PRF *adversay* $\mathcal{C}_1$ *against* KS_1*, a* $(q_e + q_d, q_e + q_d)$-PRF *adversay* $\mathcal{C}_2$ *against* KS_2*, and a* $(2q_e + q_d + 1, q_d + 1)$-INT-CTXT *adversary* $\mathcal{D}$

against AEAD *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,r_{pk},r_{psk})\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{pskAPKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} \leq & \mathsf{Adv}^{(n,q_e,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}} \\ & + \mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}_2,\mathsf{AKEM}} \\ & + \mathsf{Adv}^{(q_e+q_d,q_e+q_d)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1} + \mathsf{Adv}^{(q_d,q_d)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2} \\ & + \mathsf{Adv}^{(2q_e+q_d+1,q_d+1)\text{-INT-CTXT}}_{\mathcal{D},\mathsf{AEAD}} + \frac{q_e(q_e+q_d-1)}{|\mathcal{K}|}.\end{aligned}$$

Proof (Sketch). To achieve Outsider-Auth security for pskAPKE, we need to first use Outsider-CCA security and Outsider-Auth security of AKEM to replace the KEM secret in encryption and decryption by random values. Together with the pre-shared keys they can be used as inputs to KS where, depending on the query, either the KEM shared secret or the *psk* act as the PRF key. Next, the PRF output can be used to construct an adversary against INT-CTXT security of AEAD. The full proof can be found in Section 4.4. □

4.3 The Security of HPKE's PSK Modes

The HPKE standard's specification of the $\mathsf{HPKE}_{\mathsf{PSK}}$ mode corresponds to the construction pskPKE[KEM, KS, AEAD] (Listing 13), and the one of the $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ mode to the construction pskAPKE[AKEM, KS, AEAD] (Listing 14) with one exception. The HPKE standard explicitly defines an identifier for each *psk*, the *psk_id*, and the actual key schedule function takes it as an additional parameter alongside the KEM key K, the *psk*, and the *info* bitstring that we consider in our model. For simplicity, we abstract away the *psk_id* and consider it to be encoded as part of *info*, as both are simply hashed into the context of the key derivation. HPKE uses the following specific components:

- KEM is the standard Diffie-Hellman DH-KEM which fulfills CCA security assuming the Gap Diffie-Hellman assumption.
- AKEM is the Diffie-Hellman DH-AKEM from [1] which is proved Insider-CCA and Outsider-Auth-secure assuming the Gap Diffie-Hellman assumption.
- The key schedule KS is constructed via the functions Extract and Expand both instantiated with HMAC [1, Section 6.2]. If we assume HMAC to be a PRF when keyed on either of the inputs, the assumptions for KS hold as well.
- AEAD is instantiated using AES-GCM or ChaCha20-Poly1305, which are shown to fulfill IND-CPA and INT-CTXT security [7] and thus also IND-CCA security.

Thus, applying the composition theorems from the last section, we achieve CCA and Auth security for $\mathsf{HPKE}_{\mathsf{PSK}}$, and Insider-CCA and Outsider-Auth security for $\mathsf{HPKE}_{\mathsf{AuthPSK}}$.

4.4 Proof of Theorem 4

Proof. We describe several games depicted in Listing 15.

Game G_0. This is the $(n, q_e, q_d, r_{pk}, r_{psk})$-Outsider-Auth game for pskAPKE[AKEM, KS, AEAD], thus we have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, r_{pk}, r_{psk})\text{-Outsider-Auth}(\mathcal{A}) \Rightarrow 1].$$

Listing 15: Games $\mathsf{G}_0 - \mathsf{G}_8$ for the proof of Theorem 4.

$\underline{\mathsf{G}_0 - \mathsf{G}_8}$

01 **for** $i \in [n]$
02 $\quad(sk_i, pk_i) \xleftarrow{\$} \mathsf{GenSK}$
03 $\quad$**for** $j \in [i]$
04 $\quad\quad psk_{ij} \xleftarrow{\$} \mathcal{K}_{psk}$
05 $\quad\quad psk_{ji} \leftarrow psk_{ij}$
06 $\mathcal{E}, \mathcal{E}', \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}}, \Lambda \leftarrow \emptyset$
07 $(i^*, j^*, (c_1^*, c_2^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
08 **return** $[\![(i^* \notin \Gamma_{\mathtt{pk}} \vee (i^*, j^*) \notin \Gamma_{\mathtt{psk}}) \wedge sk_{j^*} \neq \bot$
$\quad\wedge (pk_{i^*}, pk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \notin \mathcal{E}$
$\quad\wedge \mathsf{pskADec}(pk_{i^*}, sk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \neq \bot]\!]$

$\underline{\textsc{Enc}(i \in [n], j \in [n], m, aad, info)}$

09 **if** $sk_i = \bot$
10 $\quad$**return** $\bot$
11 $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk_j)$
12 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
13 **if** $j \notin \Gamma_{\mathtt{pk}}$ // $\mathsf{G}_2 - \mathsf{G}_8$
14 $\quad K \xleftarrow{\$} \mathcal{K}$ // $\mathsf{G}_2 - \mathsf{G}_8$
15 $\quad\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(pk_i, pk_j, c_1, K)\}$ // $\mathsf{G}_2 - \mathsf{G}_8$
16 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$ // $\mathsf{G}_2 - \mathsf{G}_8$
17 $\quad$**if** $\exists k', nonce' : (k', nonce', i, j, K, psk_{ij}, info) \in \Lambda$ // $\mathsf{G}_1 - \mathsf{G}_8$
18 $\quad\quad$**abort** // $\mathsf{G}_3 - \mathsf{G}_8$
19 $\quad\quad(k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_1 - \mathsf{G}_8$
20 $\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_4 - \mathsf{G}_8$
21 $\quad\Lambda \leftarrow \Lambda \cup \{(k, nonce, i, j, K, psk_{ij}, info)\}$ // $\mathsf{G}_1 - \mathsf{G}_8$
22 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
23 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
24 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(k, nonce, m, (c_1, c_2), aad)\}$ // G_8
25 **return** (c_1, c_2)

$\underline{\textsc{Oracle Dec}(i \in [n], j \in [n], (c_1, c_2), aad, info)}$

26 **if** $sk_j = \bot$
27 $\quad$**return** $\bot$
28 $K \leftarrow \mathsf{AuthDecap}(pk_i, sk_j, c_1)$
29 **if** $\exists K' : (pk_i, pk_j, c_1, K') \in \hat{\mathcal{E}}$ // $\mathsf{G}_2 - \mathsf{G}_8$
30 $\quad K \leftarrow K'$ // $\mathsf{G}_2 - \mathsf{G}_8$
31 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$ // $\mathsf{G}_6 - \mathsf{G}_8$
32 **else if** $i \notin \Gamma_{\mathtt{pk}} \wedge K \neq \bot$ // $\mathsf{G}_5 - \mathsf{G}_8$
33 $\quad K \xleftarrow{\$} \mathcal{K}$ // $\mathsf{G}_5 - \mathsf{G}_8$
34 $\quad\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(pk_i, pk_j, c_1, K)\}$ // $\mathsf{G}_5 - \mathsf{G}_8$
35 $\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_6 - \mathsf{G}_8$
36 **else** // $\mathsf{G}_6 - \mathsf{G}_8$
37 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$ // $\mathsf{G}_6 - \mathsf{G}_8$
38 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$ // $\mathsf{G}_0 - \mathsf{G}_5$
39 **if** $i \notin \Gamma_{\mathtt{pk}} \vee (i, j) \notin \Gamma_{\mathtt{psk}}$ // $\mathsf{G}_1 - \mathsf{G}_8$
40 $\quad$**if** $\exists k', nonce' : (k', nonce', i, j, K, psk_{ij}, info) \in \Lambda$ // $\mathsf{G}_1 - \mathsf{G}_8$
41 $\quad\quad(k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_1 - \mathsf{G}_8$
42 $\quad$**else if** $i \in \Gamma_{\mathtt{pk}}$ // $\mathsf{G}_7 - \mathsf{G}_8$
43 $\quad\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_7 - \mathsf{G}_8$
44 $\quad m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // $\mathsf{G}_1 - \mathsf{G}_8$
45 $\quad m \leftarrow \bot$ // G_8
46 $\quad$**if** $\exists m' : (k, nonce, m', (c_1, c_2), aad) \in \mathcal{E}'$ // G_8
47 $\quad\quad m \leftarrow m'$ // G_8
48 **else**
49 $\quad m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // $\mathsf{G}_1 - \mathsf{G}_8$
50 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // $\mathsf{G}_0 - \mathsf{G}_1$
51 $\Lambda \leftarrow \Lambda \cup \{(k, nonce, i, j, K, psk_{ij}, info)\}$ // $\mathsf{G}_1 - \mathsf{G}_8$
52 **return** m

$\underline{\textsc{Oracle RepPK}(i \in [n], pk)}$

53 $(sk_i, pk_i) \leftarrow (\bot, pk)$
54 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{i\}$

$\underline{\textsc{Oracle RepPSK}(i \in [n], j \in [n], psk)}$

55 $psk_{ij} \leftarrow psk$
56 $psk_{ji} \leftarrow psk$
57 $\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i, j), (j, i)\}$

Game G_1. We insert a set Λ to log the outputs of KS to use the stored outputs if KS is queried on the same parameters again. This is only done for encryption oracle queries for which the receiver's key was not replaced, i.e. $j \notin \Gamma_{\mathtt{pk}}$ (checked in Line 13), as well as for decryption queries for which not both the sender's key and the *psk* were replaced, i.e. for queries on indices $i \notin \Gamma_{\mathtt{pk}} \vee (i, j) \notin \Gamma_{\mathtt{psk}}$ (checked in Line 39). Since the change is only conceptual, we have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1].$$

Game G_2. If the corresponding receiver key has not been corrupted via RepPK, i.e. $j \notin \Gamma_{\mathtt{pk}}$, we replace the KEM secret in oracle Enc by a uniformly random value (Line 14) and store the output to return consistent queries to oracle Dec. The difference is the advantage of a CCA adversary $\mathcal{B}_1$ against AKEM:

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_e,q_d,q_e)\text{-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}}.$$

Adversary $\mathcal{B}_1$ against CCA security of an AKEM can simulate $\mathsf{G}_1/\mathsf{G}_2$ by issuing a challenge query to the CCA experiment for any Enc query with $j \notin \Gamma_{\mathtt{pk}}$ and a decryption query for any Dec query. Any other query, i.e. Enc queries with $j \in \Gamma_{\mathtt{pk}}$, can be answered by using the AKEM adversary's own encapsulation oracle.

Game G_3. In G_3, the game aborts in the encryption oracle if the corresponding receiver's key was not replaced and there already exists an entry in Λ with the queried parameters (Line 18). The difference is negligible in the size of the key space of KEM. Since K was randomly chosen in the previous game, the probability of having such an entry is at most $\frac{|\Lambda|}{|\mathcal{K}|}$. Note that Λ is filled with another element at most once per Enc/Dec query. This yields the following advantage:

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \frac{q_e(q_e + q_d - 1)}{|\mathcal{K}|}.$$

Game G_4. If the receiver's key was not replaced and we do not abort, we replace the output of KS in the encryption oracle with uniformly random values of the respective domain (Line 19). The game difference is the advantage of a PRF adversary $\mathcal{C}_1$ against KS_1:

$$|\Pr[\mathsf{G}_4 \Rightarrow 1] - \Pr[\mathsf{G}_5 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e,q_e)\text{-}\mathsf{PRF}}_{\mathcal{C}_1,\mathsf{KS}_1}.$$

The changes in Game G_1 ensure consistent outputs of KS, i.e. queries on Enc or Dec with the same parameters lead to the same output of KS (or the game aborts in the for pskEnc). Hence, games G_3 and G_4 can only be distinguished by distinguishing the real output of KS from a uniformly random one. This can be turned into an adversary against PRF security of KS_1, i.e. keyed on the first input. Note that K is chosen uniformly at random due to the changes in Game G_2. There are at most q_e different instances for the PRF and at most the same number of queries.

Game G_5. In Game G_5, the decryption oracle is modified. If there KEM parameter set was not queried before, i.e. the parameters do not occur in $\hat{\mathcal{E}}$, the sender was not corrupted and the shared KEM secret K is not $\perp$ (Line 32), K is replaced by a uniformly random value and the result is stored in $\hat{\mathcal{E}}$. Due to these conditions, the setup matches with oracles of an Auth adversary against AKEM and such an adversary can simulate the games. This results in the following advantage:

$$|\Pr[\mathsf{G}_4 \Rightarrow 1] - \Pr[\mathsf{G}_5 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{B}_2,\mathsf{AKEM}}.$$

Game G_6. The game is modified by choosing uniformly random values instead of the real output of KS in the same case as for the previous game (Line 32). This can be turned into an adversary against PRF security of KS_1, i.e. keyed in the first input. There are at most q_d different instances and at most q_d different queries resulting in

$$|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_6 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_d,q_d)\text{-}\mathsf{PRF}}_{\mathcal{C}_1,\mathsf{KS}_1}.$$

Game G_7. We modify the game by replacing the output of KS by uniformly random values similar to the last game modification but in the following case while querying the decryption oracle: not both the sender's key and the *psk* were corrupted ($i \notin \Gamma_{\mathtt{pk}} \vee (i,j) \notin \Gamma_{\mathtt{psk}}$, Line 39), there is no corresponding element in Λ (Line 42), and the sender's key was corrupted, i.e. $i \in \Gamma_{\mathtt{pk}}$ (Line 42). This can be turned into a PRF adversary $\mathcal{C}_2$ against KS_2, i.e. keyed in the second input:

$$|\Pr[\mathsf{G}_7 \Rightarrow 1] - \Pr[\mathsf{G}_8 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_d,q_d)\text{-}\mathsf{PRF}}_{\mathcal{C}_2,\mathsf{KS}_2}.$$

The two conditions $i \notin \Gamma_{\mathtt{pk}} \vee (i,j) \notin \Gamma_{\mathtt{psk}}$ and $i \in \Gamma_{\mathtt{pk}}$ imply that $(i,j) \notin \Gamma_{\mathtt{psk}}$ which means that the *psk* was not replaced and was therefore chosen uniformly at random in the beginning of the game. Thus, the two games can be simulated by adversary $\mathcal{C}_2$ via their own evaluation oracle. We have at most q_d different indices for the PRF game and at most the same number of queries.

Game G_8. In this game, we replace the actual decryption in an honest decryption oracle query with $\perp$ (Line 45). Distinguishing the game difference can be turned into an INT-CTXT adversary $\mathcal{D}_1$ against AEAD:

$$|\Pr[\mathsf{G}_7 \Rightarrow 1] - \Pr[\mathsf{G}_8 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e+q_d,q_d)\text{-INT-CTXT}}_{\mathcal{D}_1,\mathsf{AEAD}}.$$

Adversary $\mathcal{D}_1$ is formally constructed in Listing 16. Note that k and *nonce* are uniformly random such that the adversary can use their own decryption oracle either on a new index or on a previous index in case the same parameters were queried before and the element is in Λ. Further, encryption queries can also be simulated in each case. If there is an encryption query with a corrupted receiver's key, the adversary can compute the encryption on their own. Otherwise, they can use their own encryption oracle. The abort in cases of a parameter set being queried before (Line 18) prevents the need of querying the encryption oracle twice which is not possible for the INT-CTXT game for an AEAD. There are at most $q_e + q_d$ different keys and adversary $\mathcal{D}_1$ makes at most q_d queries to their decryption oracle $\textsc{Dec}_{\text{AEAD}}$.

Listing 16: Adversary $\mathcal{D}_1$ against INT-CTXT security for AEAD having access to oracles $\textsc{Enc}_{\text{AEAD}}$ and $\textsc{Dec}_{\text{AEAD}}$.

$\mathcal{D}_1^{\textsc{Enc}_{\text{AEAD}},\textsc{Dec}_{\text{AEAD}}}$

01 **for** $i \in [n]$
02 $\quad(sk_i, pk_i) \xleftarrow{\$} \mathsf{GenSK}$
03 $\quad$**for** $j \in [i]$
04 $\quad\quad psk_{ij} \xleftarrow{\$} \mathcal{K}_{psk}$
05 $\quad\quad psk_{ji} \leftarrow psk_{ij}$
06 $\mathcal{E}, \mathcal{E}', \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}}, \Lambda \leftarrow \emptyset$
07 $(i^*, j^*, (c_1^*, c_2^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
08 **return** $[\![(i^* \notin \Gamma_{\mathtt{pk}} \vee (i^*, j^*) \notin \Gamma_{\mathtt{psk}}) \wedge sk_{i^*} \neq \perp$
$\quad\wedge (pk_{i^*}, pk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \notin \mathcal{E}$
$\quad\wedge \mathsf{pskADec}(pk_{i^*}, sk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \neq \perp]\!]$

Oracle $\textsc{Enc}(i \in [n], j \in [n], m, aad, info)$

09 **if** $sk_i = \perp$
10 $\quad$**return** $\perp$
11 $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk_j)$
12 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
13 **if** $j \notin \Gamma_{\mathtt{pk}}$
14 $\quad K \xleftarrow{\$} \mathcal{K}$
15 $\quad\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(pk_i, pk_j, c_1, K)\}$
16 $\quad$**if** $\exists \ell' : (\ell', i, j, K, psk_{ij}, info) \in \Lambda$
17 $\quad\quad$**abort**
18 $\quad\ell \leftarrow \ell + 1$ //new key
19 $\quad c_2 \leftarrow \textsc{Enc}_{\text{AEAD}}(\ell, m, aad)$ //enc query
20 $\quad\Lambda \leftarrow \Lambda \cup \{(\ell, i, j, K, psk_{ij}, info)\}$
21 **else**
22 $\quad c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
23 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
24 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(k, nonce, m, (c_1, c_2), aad)\}$
25 **return** (c_1, c_2)

Oracle $\textsc{Dec}(i \in [n], j \in [n], (c_1, c_2), aad, info)$

26 **if** $sk_j = \perp$
27 $\quad$**return** $\perp$
28 $K \leftarrow \mathsf{AuthDecap}(pk_i, sk_j, c_1)$
29 **if** $\exists K' : (pk_i, pk_j, c_1, K') \in \hat{\mathcal{E}}$
30 $\quad K \leftarrow K'$
31 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
32 **else if** $i \notin \Gamma_{\mathtt{pk}} \wedge K \neq \perp$
33 $\quad K \xleftarrow{\$} \mathcal{K}$
34 $\quad\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(pk_i, pk_j, c_1, K)\}$
35 $\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$
36 **else**
37 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
38 **if** $i \notin \Gamma_{\mathtt{pk}} \vee (i,j) \notin \Gamma_{\mathtt{psk}}$
39 $\quad$**if** $\exists \ell' : (\ell', i, j, K, psk_{ij}, info) \in \Lambda$
40 $\quad\quad m \leftarrow \textsc{Dec}_{\text{AEAD}}(\ell', c_2, aad)$ //dec query on old key
41 $\quad$**else if** $i \in \Gamma_{\mathtt{pk}}$
42 $\quad\quad\ell \leftarrow \ell + 1$ //new key
43 $\quad\quad m \leftarrow \textsc{Dec}_{\text{AEAD}}(\ell, c_2, aad)$ //dec query on new key
44 $\quad\quad\Lambda \leftarrow \Lambda \cup \{(\ell, i, j, K, psk_{ij}, info)\}$
45 **else**
46 $\quad m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
47 **return** m

Oracle $\textsc{RepPK}(i \in [n], pk)$

48 $(sk_i, pk_i) \leftarrow (\perp, pk)$
49 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{i\}$

Oracle $\textsc{RepPSK}(i \in [n], j \in [n], psk)$

50 $psk_{ij} \leftarrow psk$
51 $psk_{ji} \leftarrow psk$
52 $\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j), (j,i)\}$

Reduction to Game G_8. Winning Game G_8 can be reduced to an INT-CTXT adversary $\mathcal{D}_2$ against AEAD:

$$\Pr[\mathsf{G}_8 \Rightarrow 1] \leq \mathsf{Adv}^{(q_e+1,1)\text{-INT-CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}}.$$

The adversary can simulate the decryption oracle since in cases $i \notin \Gamma_{\mathtt{pk}} \vee (i,j) \notin \Gamma_{\mathtt{psk}}$, they can output $\perp$ (or the original encryption if it was produced during the experiment). In cases $(i,j) \in \Gamma_{\mathtt{psk}}$, they can compute the output by their own. For the encryption oracle, the adversary can use their own encryption oracle or compute the output on their own similar to the adversary from the last game hop. The output of the adversary against game G_8 can then be used to issue a decryption query in the INT-CTXT experiment on either a new key or a previous key if the output parameters $i^*, j^*, c_1^*, \mathit{info}^*$ match with that key. Matching parameters can be identified by computing $K^* \leftarrow \mathsf{Decaps}(sk_{j^*}, c_1^*)$ and comparing $(i^*, j^*, K^*, \mathit{psk}_{i^*j^*}, \mathit{info}^*)$ to set Λ similar to Line 16 or Line 39 in Listing 16. If the adversary against G_8 wins, the adversary against the INT-CTXT experiment has a valid ciphertext which does not decrypt to $\perp$ due to the winning condition of G_8. That means they can distinguish between the real or random case since the result of the decryption query must be unequal to $\perp$ in the real case.

Putting everything together, we obtain the stated bound. □

5 Hybrid Post-Quantum APKE

We want to build an HPKE scheme which is secure against classical as well as quantum adversaries. To not rely solely on relatively new post-quantum primitives, a common way is to use combiners which combine well studied classical primitives and post-quantum primitives at the same time. This hybrid approach allows for security against future quantum adversaries but is still secure in a classical setting if current post-quantum primitives are broken. To this end, we use the pre-shared key mode of HPKE to build a combiner from which we can instantiate a hybrid post-quantum construction.

Let $\mathsf{pskAPKE}[\mathsf{AKEM}_1, \mathsf{KS}, \mathsf{AEAD}]$ be a pre-shared key PKE based on an authenticated KEM $\mathsf{AKEM}_1 = (\mathsf{Gen}_1, \mathsf{AuthEncap}_1, \mathsf{AuthDecap}_1)$, a two-keyed function KS, and an authenticated encryption with associated data AEAD as in Listing 14. Further, let $\mathsf{AKEM}_2 = (\mathsf{Gen}_2, \mathsf{AuthEncap}_2, \mathsf{AuthDecap}_2)$ be a second authenticated KEM. From these components, we can construct an APKE using the shared secret of the second KEM as the pre-shared key of the pskAPKE. We remark that the same construction also works for non-authenticated primitives, i.e., we can construct a PKE $\mathsf{PKE}[\mathsf{KEM}_1, \mathsf{KEM}_2, \mathsf{KS}, \mathsf{AEAD}]$ built from $\mathsf{pskPKE}[\mathsf{KEM}_1, \mathsf{KS}, \mathsf{AEAD}]$ and a KEM_2.

The following two theorems state that the APKE is secure (in the sense of Insider-CCA and Outsider-Auth) if at least one of the underlying AKEMs, AKEM_1 or AKEM_2, is secure.

Theorem 5. *Let* AKEM_1 *and* AKEM_2 *be two AKEMs,* KS *a two-keyed function, and* AEAD *an AEAD. If* KS *is a* PRF *in both keys,* AEAD *is* IND-CCA *secure, and* AKEM_1 *or* AKEM_2 *is* CCA *secure, then the construction in Listing 17 is a* CCA *secure* APKE*. In particular, for any* (n, q_e, q_d, q_c)*-*Insider-CCA *adversary* $\mathcal{A}$ *against* $\mathsf{APKE}[\mathsf{AKEM}_1, \mathsf{AKEM}_2, \mathsf{KS}, \mathsf{AEAD}]$ *there exists a* $(n{+}1, q_e, q_d, q_c)$*-*Insider-CCA *adversary* $\mathcal{B}_1$ *against* AKEM_1*, a* (n, q_e, q_d, q_c, q_c)*-*Insider-CCA *adversary* $\mathcal{B}_2$ *against* AKEM_2*, a* $(q_c, q_d + q_c)$*-*PRF *adversary* $\mathcal{C}_1$ *against* KS_1*, a* $(q_c, q_d + q_c)$*-*PRF *adversary* $\mathcal{C}_2$ *against* KS_2*, and a* (q_c, q_d)*-*CCA *adversary* $\mathcal{D}$ *against* AEAD

Listing 17: Authenticated PKE $\mathsf{APKE}[\mathsf{AKEM}_1, \mathsf{AKEM}_2, \mathsf{KS}, \mathsf{AEAD}]$ built from $\mathsf{pskAPKE}[\mathsf{AKEM}_1, \mathsf{KS}, \mathsf{AEAD}]$ and AKEM_2

$\underline{\mathsf{Gen}}$	$\underline{\mathsf{Enc}((sk_1, sk_2), (pk_1, pk_2), m, aad, info)}$
$(sk_1, pk_1) \xleftarrow{\$} \mathsf{Gen}_1$	$(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk_2, pk_2)$
$(sk_2, pk_2) \xleftarrow{\$} \mathsf{Gen}_2$	$(c_1, c_2) \xleftarrow{\$} \mathsf{pskAEnc}(sk_1, pk_1, K', m, aad, c' \|\| info)$
$sk \leftarrow (sk_1, sk_2)$	**return** $((c_1, c_2), c')$
$pk \leftarrow (pk_1, pk_2)$	
return (sk, pk)	$\underline{\mathsf{Dec}((pk_1, pk_2), (sk_1, sk_2), ((c_1, c_2), c'), aad, info)}$
	$K' \leftarrow \mathsf{AuthDecap}_2(sk_2, c')$
	$m \leftarrow \mathsf{pskADec}(pk_1, sk_1, K', (c_1, c_2), aad, c' \|\| info)$
	return m

such that

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq$$

$$\min\{\mathsf{Adv}^{(n+1,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}_1} + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1},$$
$$\mathsf{Adv}^{(n,q_e,q_d,q_c,q_c)\text{-Insider-CCA}}_{\mathcal{B}_2,\mathsf{AKEM}_2} + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2}\} + \mathsf{Adv}^{(q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}$$

Proof (Sketch). The first part of the proof is very similar to Theorem 2 except that the queries to KS_2 can be saved. The second part transforms the KEM secret of AKEM_2 into a uniformly random value using its Insider-CCA security which can then be used as input to KS_2 as a regular *psk*. These outputs are uniformly random values and the remaining transformations are as for the first part. The full proof can be found in Appendix C.1. □

Theorem 6. *Let* AKEM_1 *and* AKEM_2 *be two AKEMs,* KS *a two-keyed function, and* AEAD *an AEAD. If* KS *is a* PRF *in both keys,* AEAD *is* INT-CTXT *and* IND-CPA *secure, and* AKEM_1 *or* AKEM_2 *is* Outsider-Auth *secure, then the construction in Listing 17 is a* Outsider-Auth *secure* APKE*. In particular, for any* (n, q_e, q_d)*-*Outsider-Auth *adversary* $\mathcal{A}$ *against* $\mathsf{APKE}[\mathsf{AKEM}_1, \mathsf{AKEM}_2, \mathsf{KS}, \mathsf{AEAD}]$*, there exists a* $(n+1, 0, q_d, q_c)$*-*Outsider-CCA *adversary* $\mathcal{B}_1$ *against* AKEM_1*, a* $(n+1, q_e, q_d)$*-*Outsider-Auth *adversary* $\mathcal{B}_2$ *against* AKEM_1*, a* $(n, 0, q_d, q_e)$*-*Outsider-CCA *adversary* $\mathcal{B}'_1$ *against* AKEM_2*, a* (n, q_e, q_d)*-*Outsider-Auth *adversary* $\mathcal{B}'_2$ *against* AKEM_2*, a* $(q_e + q_d, q_e + q_d)$*-*PRF *adversary* $\mathcal{C}_1$ *against* KS_1*, a* (q_d, q_d)*-*PRF *adversary* $\mathcal{C}_2$ *against* KS_2*, a* $(2q_e + q_d + 1, q_d + 1)$*-*INT-CTXT *adversary* $\mathcal{D}$ *against* AEAD *such that*

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq$$

$$\begin{aligned}\min\{&\mathsf{Adv}^{(n+1,q_e,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}_1} + \mathsf{Adv}^{(n+1,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}_2,\mathsf{AKEM}_1} \\ &+ \mathsf{Adv}^{(q_e+q_d,q_e+q_d)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1}, \\ &\mathsf{Adv}^{(n,0,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}'_1,\mathsf{AKEM}_2} + \mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}'_2,\mathsf{AKEM}_2} \\ &+ \mathsf{Adv}^{(q_d,q_d)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2}\} \\ &+ \mathsf{Adv}^{(2q_e+q_d+1,q_d+1)\text{-INT-CTXT}}_{\mathcal{D},\mathsf{AEAD}} + \frac{q_e(q_e+q_d-1)}{|\mathcal{K}|}.\end{aligned}$$

Proof (Sketch). The first part of the proof is very similar to Theorem 4 except that the queries to KS_2 can be saved. The second part transforms the KEM secret of AKEM_2 into a uniformly random value using its Insider-CCA and Outsider-Auth security which can then be used as input to KS_2 as a regular *psk* in encryption and decryption oracle. These outputs are uniformly random values and the remaining transformations are as for the first part. The full proof can be found in Appendix C.2. □

Post-Quantum Instantiation. Consequently, one can combine $\mathsf{HPKE}_{\mathsf{AuthPSK}}$ with a post-quantum secure AKEM to obtain an APKE scheme with hybrid security. Analogously, one can combine $\mathsf{HPKE}_{\mathsf{PSK}}$ with a post-quantum secure KEM (such as Kyber) to obtain a PKE with hybrid security. In the next section, we discuss how to construct post-quantum secure AKEM schemes.

6 Post-Quantum AKEM Constructions

6.1 KEM-then-Sign-then-Hash

A well-known approach for constructing a post-quantum AKEM is to combine a post-quantum KEM with a post-quantum signature [10]. This could obviously applied to the classical setting as well but with much worse performance than the NIKE-based construction of HPKE which achieves authentication almost for free.

Our new construction extends the (insecure) Encrypt-then-Sign (EtS) paradigm to Encrypt-then-Sign-then-Hash (EtStH). Let $\mathsf{KEM} = (\mathsf{KEM.Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ be a signature scheme. We construct $\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM}, \mathsf{SIG}, \mathsf{H}]$ as shown in Listing 18 for H a keyed function. We define the output of H to be $\perp$ if one of the inputs is $\perp$. The key generation outputs a public key tuple and a private key tuple. The first component of both tuples is the receiver's public/private key and the second component is the sender's public/private key.

Listing 18: $\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM}, \mathsf{SIG}, \mathsf{H}]$ from a KEM $\mathsf{KEM} = (\mathsf{KEM.Gen}, \mathsf{Encaps}, \mathsf{Decaps})$, a signature scheme $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{Sign}, \mathsf{Vfy})$, and a random oracle H.

```
Gen
01 (sk, pk) ←$ KEM.Gen
02 (sigk, vk) ←$ SIG.Gen
03 return ((sk, sigk), (pk, vk))

AuthEncap((sk_1, sigk_1), (pk_2, vk_2))
04 (c, K') ←$ Encaps(pk_2)
05 σ ←$ Sign(sigk_1, c||μ(sk_1)||pk_2||vk_2)
06 K ← H(K', σ||μ(sk_1)||μ'(sigk_1)||pk_2||vk_2)
07 return ((c, σ), K)

AuthDecap((pk_1, vk_1), (sk_2, sigk_2), (c, σ))
08 if Vfy(vk_1, c||pk_1||μ(sk_2)||μ'(sigk_2), σ) ≠ 1
09    K ← ⊥
10 else
11    K' ← Decaps(sk_2, c)
12    K ← H(K', σ||pk_1||vk_1||μ(sk_2)||μ'(sigk_2))
13 return K
```

Theorem 7 (KEM CCA + H PRF $\Rightarrow$ AKEM Insider-CCA)**.** *If* KEM *is a* CCA *secure key encapsulation mechanism and* H *is a* PRF*, then* $\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM}, \mathsf{SIG}, \mathsf{H}]$ *is an* Insider-CCA

secure AKEM. In particular, for every $(n, q_e, q_d, q_c, r_{sk})$-Insider-CCA *adversary* $\mathcal{A}$ *against* $\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM}, \mathsf{SIG}, \mathsf{H}]$ *there exists a* (n, q_d, q_c)-CCA *adversary* $\mathcal{B}$ *against* KEM *and a* $(q_c, q_d + q_c)$-PRF *adversary* $\mathcal{C}$ *against* H *such that*

$$\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{sk})\text{-Insider-CCA}}_{\mathcal{A},\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM},\mathsf{SIG},\mathsf{H}]} \leq \mathsf{Adv}^{(n,q_d,q_c)\text{-CCA}}_{\mathcal{B},\mathsf{KEM}} + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C},\mathsf{H}}.$$

Proof (Sketch). We use the CCA security of KEM to make the KEM keys random, such that the key to H is uniformly random. Using the PRF property of H gives a uniformly random value for the final key. The full proof can be found in Appendix D.1. □

Theorem 8 (SIG SUF-CMA ⇒ AKEM Outsider-Auth)**.** *If* SIG *is an* SUF-CMA *secure signature scheme, then* $\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM}, \mathsf{SIG}, \mathsf{H}]$ *is an* Outsider-Auth *secure AKEM. In particular, for every* (n, q_e, q_d)-Outsider-Auth *adversary* $\mathcal{A}$ *there exists a* (n, q_e)-SUF-CMA *adversary* $\mathcal{B}$ *against* SIG *such that*

$$\mathsf{Adv}^{\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM},\mathsf{SIG},\mathsf{H}]} \leq \mathsf{Adv}^{(n,q_e)\text{-SUF-CMA}}_{\mathcal{B},\mathsf{SIG}}.$$

Proof (Sketch). Queries to the decapsulation oracle containing invalid signatures cannot be distinguished by an adversary due to the definition of the scheme. Valid queries can be used against the SUF-CMA security of SIG. The full proof can be found in Appendix D.2. □

6.2 AKEM from NIKE

We can build an AKEM from a NIKE. Let $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{NIKE.KeyGen}, \mathsf{NIKE.SharedKey})$ be a NIKE and H a 2-keyed function, then we can construct an AKEM $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ as defined in Listing 19. We define the output of H to be $\perp$ if one of the inputs is $\perp$. By H_1, we denote function H keyed in the first component and by H_2 function H keyed in the second component. In contrast to an earlier version of the paper, hash function H takes both the users' public keys and the ciphertext pk^* as additional input. This change prevents two attacks possible in the previous version. If the sender and receiver keys are not hashed and adversary can query the decapsulation oracle on the output of an encapsulation with switched roles of sender and receiver. If the ciphertext is not hashed, an adversary could find a collision in the remaining inputs to the hash function and exploit this to distinguish the output of the decapsulation oracle.

Theorem 9 (NIKE Active + H_2 PRF ⇒ AKEM Insider-CCA)**.** *Let* NIKE *be a NIKE and* H *a 2-keyed function. If* NIKE *is* Active *secure and* H_2 *a* PRF*, then* $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ *is* Insider-CCA *secure. In particular for any adversary* $\mathcal{A}$ *against* $(n, q_e, q_d, q_c, r_{sk})$-Insider-CCA *security of* $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ *there exists an* $(n + q_c, q_e + 2q_d, 0, q_e + q_d + q_c, q_e + 2q_d, q_c)$-Active *adversary against* NIKE *and a* (q_c, q_c)-PRF *adversary* $\mathcal{C}$ *against* H_2 *such that*

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c,r_{sk})\text{-Insider-CCA}}_{\mathcal{A},\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE},\mathsf{H}]} &\leq \mathsf{Adv}^{(n+q_c,q_e+2q_d,0,q_e+q_d+q_c,q_e+2q_d,q_c)\text{-Active}}_{\mathcal{B},\mathsf{NIKE}} \\ &\quad + \mathsf{Adv}^{(q_c,q_c)\text{-PRF}}_{\mathcal{C},\mathsf{H}_2}.\end{aligned}$$

Proof (Sketch). Assuming an active secure NIKE, the second shared key, K_2 is indistinguishable from random. We show that by constructing an adversary against an Active secure NIKE using an Insider-CCA adversary against $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ by simulating the

Listing 19: $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ from $\mathsf{NIKE} = (\mathsf{NIKE.KeyGen}, \mathsf{NIKE.SharedKey})$ where the setup parameters are known to every user.

$\underline{\mathsf{Gen}}$
01 $(sk, pk) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
02 **return** (sk, pk)

$\underline{\mathsf{AuthEncap}(sk_1, pk_2)}$
03 $(sk^*, pk^*) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
04 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_1, pk_2)$
05 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk^*, pk_2)$
06 $pk_1 \leftarrow \mu(sk_1)$
07 $K \leftarrow \mathsf{H}(K_1, K_2, pk_1||pk_2||pk^*)$
08 **return** (pk^*, K)

$\underline{\mathsf{AuthDecap}(sk_2, pk_1, pk^*)}$
09 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_2, pk_1)$
10 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk_2, pk^*)$
11 $pk_2 \leftarrow \mu(sk_2)$
12 $K \leftarrow \mathsf{H}(K_1, K_2, pk_1||pk_2||pk^*)$
13 **return** K

corruptions from RepSK by registering corrupt users in the NIKE game. Then, every other query can be answered by registering a new key (if the query was made with a chosen public key) or computed by the simulator themselves. The test query of the adversary against NIKE is then directly embedded int the challenge query of the Insider-CCA game. With H_2 being a PRF, we can further show that the resulting key is also uniformly random. The full proof can be found in Appendix D.3. □

Theorem 10 (NIKE Active + H_1 PRF ⇒ AKEM Outsider-Auth)**.** *Let* $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{NIKE.KeyGen}, \mathsf{NIKE.SharedKey})$ *be a NIKE and* H *a 2-keyed function. If* NIKE *is* Active *secure and* H_1 *a* PRF*, then* $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ *is* Outsider-Auth *secure. In particular, for every* (n, q_e, q_d)-Outsider-Auth *adversary against* $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ *there exists an* $(n, q_e + 2q_d, 0, q_e, 2q_e + 2q_d, q_d)$-Active *adversary against* NIKE *and a* (q_d, q_d)-PRF *adversary* $\mathcal{C}$ *against* H_1 *such that*

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE},\mathsf{H}]} \leq \mathsf{Adv}^{(n,q_e+q_d,0,0,q_e+q_d,q_e+q_d)\text{-Active}}_{\mathcal{B},\mathsf{NIKE}} + \mathsf{Adv}^{(q_e+q_d,q_e+q_d)\text{-PRF}}_{\mathcal{C},\mathsf{H}_1} + q_e(q_e+q_d)P_{\mathsf{NIKE}}$$

Proof (Sketch). The structure is similar to the proof of Theorem 9 except that the test query is embedded in both encapsulation and the decapsulation oracle and that it is only embedded for queries with honest public keys. Further we need the entropy of the NIKE public key to avoid attacks using collisions in the hash function. The full proof can be found in Appendix D.4. □

Acknowledgements. The authors thank the anonymous reviewers to point out an error in our NIKE construction and an error in one of our proofs. They also thank Doreen Riepel for very helpful feedback and discussions. Jonas Janneck was supported by the European Union (ERC AdG REWORC - 101054911). Eike Kiltz was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA - 390781972, and by the European Union (ERC AdG REWORC - 101054911).

References

1. Alwen, J., Blanchet, B., Hauck, E., Kiltz, E., Lipp, B., Riepel, D.: Analysing the HPKE standard. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 87–116. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_4 3, 4, 5, 6, 8, 9, 10, 11, 18, 22
2. Anastasova, M., Kampanakis, P., Massimo, J.: PQ-HPKE: post-quantum hybrid public key encryption. IACR Cryptol. ePrint Arch. p. 414 (2022), https://eprint.iacr.org/2022/414 6
3. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-20, Internet Engineering Task Force (Mar 2023), https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/20/, work in Progress 3
4. Barnes, R.L., Bhargavan, K., Lipp, B., Wood, C.A.: Hybrid public key encryption. RFC 9180, RFC Editor (Feb 2022), https://www.rfc-editor.org/rfc/rfc9180.html 3
5. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 531–545. Springer (2000) 12
6. Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331 (2004), https://eprint.iacr.org/2004/331 7
7. Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 247–276. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_10 22
8. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018) 7
9. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing **33**(1), 167–226 (2003) 3
10. Dent, A.W., Zheng, Y. (eds.): Practical Signcryption. Information Security and Cryptography, Springer (2010). https://doi.org/10.1007/978-3-540-89411-7 7, 18, 28
11. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839 7
12. Duman, J., Hartmann, D., Kiltz, E., Kunzweiler, S., Lehmann, J., Riepel, D.: Group action key encapsulation and non-interactive key exchange in the qrom. In: Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II. pp. 36–66. Springer (2023) 7
13. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 254–271. Springer, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_17 7, 32
14. Gajland, P., de Kock, B., Quaresma, M., Malavolta, G., Schwabe, P.: Swoosh: Practical lattice-based non-interactive key exchange. Cryptology ePrint Archive (2023) 7
15. Geoghegan, T., Patton, C., Rescorla, E., Wood, C.A.: Distributed Aggregation Protocol for Privacy Preserving Measurement. Internet-Draft draft-ietf-ppm-dap-04, Internet Engineering Task Force (Mar 2023), https://datatracker.ietf.org/doc/draft-ietf-ppm-dap/04/, work in Progress 3
16. Kinnear, E., McManus, P., Pauly, T., Verma, T., Wood, C.A.: Oblivious DNS over HTTPS. Tech. Rep. 9230 (Jun 2022). https://doi.org/10.17487/RFC9230, https://www.rfc-editor.org/info/rfc9230 3

17. Langley, A., Hamburg, M., Turner, S.: Elliptic curves for security. RFC 7748, RFC Editor (Jan 2016), https://www.rfc-editor.org/rfc/rfc7748.html 3
18. Len, J., Grubbs, P., Ristenpart, T.: Partitioning oracle attacks. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 195–212. USENIX Association (Aug 2021) 5
19. National Institute of Standards and Technology: Digital Signature Standard (DSS). FIPS Publication 186-4 (Jul 2013), https://doi.org/10.6028/nist.fips.186-4 3
20. Paterson, K.G., van der Merwe, T.: Reactive and proactive standardisation of TLS. In: Chen, L., McGrew, D.A., Mitchell, C.J. (eds.) Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10074, pp. 160–186. Springer (2016). https://doi.org/10.1007/978-3-319-49100-4_7, https://doi.org/10.1007/978-3-319-49100-4_7 3
21. Rescorla, E., Oku, K., Sullivan, N., Wood, C.A.: TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-16, Internet Engineering Task Force (Apr 2023), https://datatracker.ietf.org/doc/draft-ietf-tls-esni/16/, work in Progress 3
22. Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) ≪ cost(signature) + cost(encryption). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052234 3

A Omitted Security Definition

A.1 Active Security NIKE

The following definition also known as m-CKS-heavy is taken from [13].

Definition 12 (Active Security). *Let* $\mathsf{NIKE} = (\mathsf{Setup}, \mathsf{NIKE.SharedKey})$ *be a NIKE. We define active security of* NIKE *(*Active*) via game* $(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)$-Active *in Listing 20. The advantage of an adversary* $\mathcal{A}$ *is*

$$\mathsf{Adv}^{(q_H,q_C,q_E,q_{HR},q_{CR},q_T)\text{-Active}}_{\mathcal{A},\mathsf{NIKE}} = \left|\Pr[(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)\text{-}\mathsf{Active}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right|.$$

B Proofs for the pskPKE and pskAPKE constructions

B.1 Proof of Theorem 1

Proof. We construct a sequence of games as shown in Listing 21.

Game G_0. G_0 is the $(n, q_e, q_d, q_c, r_{pk}, r_{psk})$-CCA game for pskPKE[KEM, KS, AEAD] (cf. Section 3.2) executed with an adversary $\mathcal{A}$, i.e.,

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, q_c, r_{pk}, r_{psk})\text{-}\mathsf{CCA}(\mathcal{A}) \Rightarrow 1].$$

Listing 20: Game $(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)$-Active for NIKE. Adversary $\mathcal{A}$ makes at most q_H queries to RegHonest, at most q_C queries to RegCorrupt, at most q_E queries to Extract, at most q_{HR} queries to HonestRev, at most q_{CR} queries to CorruptRev, and at most q_T queries to Test. The total number of users is denoted by $n = q_H + q_C$.

$(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)$-Active
01 $\mathcal{E}, \mathcal{T}, \mathcal{R} \leftarrow \emptyset$
02 $b \xleftarrow{\$} \{0,1\}$
03 $b' \xleftarrow{\$} \mathcal{A}^{\text{RegHonest},\text{RegCorrupt},\text{Extract},\text{HonestRev},\text{CorruptRev},\text{Test}}$
04 **return** $[\![b = b' \wedge \nexists (i,j) \in \mathcal{R} : (i,j,\cdot) \in \mathcal{T}$
$\wedge \nexists i \in \mathcal{E} : (i,\cdot,\cdot) \in \mathcal{T} \wedge \nexists j \in \mathcal{E} : (\cdot,j,\cdot) \in \mathcal{T}]\!]$

Oracle RegHonest$(i \in [n])$
05 **if** $pk_i \neq \bot \wedge sk_i = \bot$
06 **return** $\bot$
07 $(sk_i, pk_i) \xleftarrow{\$}$ NIKE.KeyGen
08 **return** pk_i

Oracle RegCorrupt$(i \in [n], pk)$
09 **if** $sk_i \neq \bot$
10 **return** $\bot$
11 $pk_i \leftarrow pk$
12 $sk_i = \bot$

Oracle Extract$(i \in [n], pk)$
13 $\mathcal{E} \leftarrow \mathcal{E} \cup \{i\}$
14 **return** sk_i

Oracle HonestRev$(i \in [n], j \in [n])$
15 **if** $sk_i = \bot \vee sk_j = \bot$
16 **return** $\bot$
17 $\mathcal{R} \leftarrow \mathcal{R} \cup \{(i,j),(j,i)\}$
18 **return** NIKE.SharedKey(sk_i, pk_j)

Oracle CorruptRev$(i \in [n], j \in [n])$
19 **if** $pk_i = \bot \vee pk_j = \bot$
20 **return** $\bot$
21 $\mathcal{R} \leftarrow \mathcal{R} \cup \{(i,j),(j,i)\}$
22 **if** $sk_i \neq \bot \wedge sk_j = \bot$
23 **return** NIKE.SharedKey(sk_i, pk_j)
24 **if** $sk_i = \bot \wedge sk_j \neq \bot$
25 **return** NIKE.SharedKey(sk_j, pk_i)
26 **return** $\bot$

Oracle Test$(i \in [n], j \in [n])$
27 **if** $i = j \vee sk_i = \bot \vee sk_j = \bot$
28 **return** $\bot$
29 **if** $\exists K : (i,j,K) \in \mathcal{T}$
30 **return** K
31 **if** $b = 0$
32 $K \leftarrow$ NIKE.SharedKey(sk_i, pk_j)
33 **else**
34 $K \xleftarrow{\$} \mathcal{SHK}$
35 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(i,j,K)\}$
36 **return** K

Game G_1. In this game, we make the following changes. If the challenge oracle Chall is queried on a receiver index j which was not replaced (i.e., $j \notin \Gamma_{\text{pk}}$) we replace the KEM key K by a uniformly random value from the key space $\mathcal{K}$. The result is then stored together with the receiver's public key and the ciphertext in set $\mathcal{E}'$. For consistency, we further change the decryption oracle Dec by replacing the KEM key K by the one from set $\mathcal{E}'$ if there already exists such a key for the queried ciphertext c_1 and the receiver's public key pk_j. Distinguishing these changes can be turned into an adversary $\mathcal{B}$ against (n, q_d, q_c)-CCA security of KEM as depicted in Listing 22.

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_d,q_c)\text{-CCA}}_{\mathcal{B},\mathsf{KEM}}.$$

Note that $\mathcal{B}$ can simulate all oracles by their own or with the oracles provided by their own experiment. For the RepPK oracle, the reduction has to store the indices but can abort with $\bot$ for corresponding decryption oracle queries, i.e. decryption queries for which $sk_j = \bot$ due to Line 23 in the game. The decryption oracle can be simulated by the decapsulation oracle provided by the KEM CCA game. If the challenge oracle is queried on non-replaced receiver keys, i.e., $j \notin \Gamma_{\text{pk}}$ (Line 46 in the game), $\mathcal{B}$ can use their own

Listing 21: Games $\mathsf{G}_0 - \mathsf{G}_4$ for the proof of Theorem 1.

Games $\mathsf{G}_0 - \mathsf{G}_4$

01 **for** $i \in [n]$
02 $\quad(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\quad$**for** $j \in [i]$
04 $\quad\quad psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
05 $\quad\quad psk_{ji} \leftarrow psk_{ij}$
06 $\mathcal{E}, \mathcal{E}', \Gamma_{\mathsf{pk}}, \Gamma_{\mathsf{psk}} \leftarrow \emptyset$
07 $b \xleftarrow{\$} \{0,1\}$
08 $b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chall},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
09 **return** $[\![b = b']\!]$

Oracle $\textsc{Enc}(i \in [n], j \in [n], m, aad, info)$

10 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
11 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
12 **if** $(i,j) \notin \Gamma_{\mathsf{psk}}$ // $\mathsf{G}_3 - \mathsf{G}_4$
13 $\quad$**if** $\exists (k', nonce') : (k', nonce', K, psk_{ij}, info) \in \Lambda_{Enc}$ // $\mathsf{G}_3 - \mathsf{G}_4$
14 $\quad\quad$**abort** // $\mathsf{G}_3 - \mathsf{G}_4$
15 $\quad$**else if** $\exists (k', nonce') : (k', nonce', K, psk_{ij}, info) \in \Lambda_{Dec}$ // $\mathsf{G}_3 - \mathsf{G}_4$
16 $\quad\quad (k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_3 - \mathsf{G}_4$
17 $\quad\quad \Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_3 - \mathsf{G}_4$
18 $\quad$**else** // $\mathsf{G}_3 - \mathsf{G}_4$
19 $\quad\quad (k, nonce) \xleftarrow{\$} \mathcal{K} \times \{0,1\}^{N_{nonce}}$ // G_4
20 $\quad\quad \Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_3 - \mathsf{G}_4$
21 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
22 **return** (c_1, c_2)

Oracle $\textsc{Dec}(i \in [n], j \in [n], (c_1, c_2), aad, info)$

23 **if** $sk_j = \bot \vee (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in \mathcal{E}$
24 $\quad$**return** $\bot$
25 $K \leftarrow \mathsf{KEM.Decaps}(sk_j, c_1)$
26 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
27 **if** $\exists K' : (pk_j, c_1, K') \in \mathcal{E}'$ // $\mathsf{G}_1 - \mathsf{G}_4$
28 $\quad(k, nonce) \leftarrow \mathsf{KS}(K', psk_{ij}, info)$ // $\mathsf{G}_1 - \mathsf{G}_4$
29 $\quad$**if** $\exists (k', nonce') : (k', nonce', K', psk_{ij}, info) \in \Lambda_K$ // $\mathsf{G}_2 - \mathsf{G}_4$
30 $\quad\quad (k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_2 - \mathsf{G}_4$
31 $\quad$**else** // $\mathsf{G}_2 - \mathsf{G}_4$
32 $\quad\quad (k, nonce) \xleftarrow{\$} \mathcal{K} \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_2 - \mathsf{G}_4$
33 $\quad\quad \Lambda_K \leftarrow \Lambda_K \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_2 - \mathsf{G}_4$
34 **else if** $(i,j) \notin \Gamma_{\mathsf{psk}}$ // $\mathsf{G}_3 - \mathsf{G}_4$
35 $\quad$**if** $\exists (k', nonce') : (k', nonce', K, psk_{ij}, info) \in \Lambda_{Enc} \cup \Lambda_{Dec}$ // $\mathsf{G}_3 - \mathsf{G}_4$
36 $\quad\quad (k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_3 - \mathsf{G}_4$
37 $\quad$**else** // $\mathsf{G}_3 - \mathsf{G}_4$
38 $\quad\quad (k, nonce) \xleftarrow{\$} \mathcal{K} \times \{0,1\}^{N_{nonce}}$ // G_4
39 $\quad\quad \Lambda_{Dec} \leftarrow \Lambda_{Dec} \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_3 - \mathsf{G}_4$
40 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
41 **return** m

Oracle $\textsc{Chall}(i \in [n], j \in [n], m_0, m_1, aad, info)$

42 **if** $|m_0| \neq |m_1| \vee (j \in \Gamma_{\mathsf{pk}} \wedge (i,j) \in \Gamma_{\mathsf{psk}})$
43 $\quad$**return** $\bot$
44 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
45 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
46 **if** $j \notin \Gamma_{\mathsf{pk}}$ // $\mathsf{G}_1 - \mathsf{G}_4$
47 $\quad K \xleftarrow{\$} \mathcal{K}$ // $\mathsf{G}_1 - \mathsf{G}_4$
48 $\quad\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk_j, c_1, K)\}$ // $\mathsf{G}_1 - \mathsf{G}_4$
49 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$ // $\mathsf{G}_1 - \mathsf{G}_4$
50 $\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_2 - \mathsf{G}_4$
51 $\quad\Lambda_K \leftarrow \Lambda_K \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_2 - \mathsf{G}_4$
52 **else** // $\mathsf{G}_3 - \mathsf{G}_4$
53 $\quad$**if** $\exists (k', nonce') : (k', nonce', K, psk_{ij}, info) \in \Lambda_{Enc}$ // $\mathsf{G}_3 - \mathsf{G}_4$
54 $\quad\quad$**abort** // $\mathsf{G}_3 - \mathsf{G}_4$
55 $\quad$**else if** $\exists (k', nonce') : (k', nonce', K, psk_{ij}, info) \in \Lambda_{Dec}$ // $\mathsf{G}_3 - \mathsf{G}_4$
56 $\quad\quad (k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_3 - \mathsf{G}_4$
57 $\quad\quad \Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_3 - \mathsf{G}_4$
58 $\quad$**else** // $\mathsf{G}_3 - \mathsf{G}_4$
59 $\quad\quad (k, nonce) \xleftarrow{\$} \mathcal{K} \times \{0,1\}^{N_{nonce}}$ // G_4
60 $\quad\quad \Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, psk_{ij}, info)\}$ // $\mathsf{G}_3 - \mathsf{G}_4$
61 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
62 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
63 **return** (c_1, c_2)

Oracle $\textsc{RepPK}(j \in [n], pk)$

64 $(sk_j, pk_j) \leftarrow (\bot, pk)$
65 $\Gamma_{\mathsf{pk}} \leftarrow \Gamma_{\mathsf{pk}} \cup \{j\}$

Oracle $\textsc{RepPSK}(i \in [n], j \in [n], psk)$

66 $psk_{ij} \leftarrow psk$
67 $psk_{ji} \leftarrow psk$
68 $\Gamma_{\mathsf{psk}} \leftarrow \Gamma_{\mathsf{psk}} \cup \{(i,j), (j,i)\}$

challenge. If $\mathcal{B}$ is in the real case of their (n, q_d, q_c)-CCA game (case $b = 0$ in Listing 1), they perfectly simulate G_0. If they are in the random case (case $b = 1$ in Listing 1), they perfectly simulate G_1. Any other encapsulation calls can be computed by the reduction itself.

Game G_2. In Game G_2, we change the challenge oracle by replacing the output of KS by uniformly random values of the respective domain if the receiver's key was not replaced, i.e. if $j \notin \Gamma_{\mathsf{pk}}$ (Line 46). Input and output are also logged using set Λ_K (Line 51) to return consistent decapsulation queries. To this end, the decapsulation oracle is modified such that for KS it returns a previous output stored in Λ_K (Line 30) or a uniformly random as well (Line 32). The difference can be turned into a PRF adversary $\mathcal{C}_1$ against KS_1, i.e. KS keyed in the first component:

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_c, q_d+q_c)\text{-}\mathsf{PRF}}_{\mathcal{C}_1, \mathsf{KS}_1}.$$

To analyze KS_1, we view KS as a keyed function $\mathsf{KS}(K, (psk, info))$ where key K is chosen uniformly at random. Thus, an adversary distinguishing between G_1 and G_2 can be turned

Listing 22: Adversary $\mathcal{B}$ against (n, q_d, q_c)-CCA security for KEM having access to oracles ODecap and OChall using an adversary $\mathcal{A}$ against $\mathsf{G}_0/\mathsf{G}_1$ of the proof of Theorem 1.

$\mathcal{B}^{\text{ODecap},\text{OChall}}(pk_1, \ldots, pk_n)$
01 **for** $i \in [n]$
02 **for** $j \in [i]$
03 $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
04 $psk_{ji} \leftarrow psk_{ij}$
05 $\mathcal{E}, \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}} \leftarrow \emptyset$
06 $b \xleftarrow{\$} \{0,1\}$
07 $b' \xleftarrow{\$} \mathcal{A}^{\text{Enc},\text{Dec},\text{Chall},\text{RepPK},\text{RepPSK}}(pk_1, \ldots, pk_n)$
08 **return** $[\![b = b']\!]$

Oracle Enc$(i \in [n], j \in [n], m, aad, info)$
09 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
10 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
11 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
12 **return** (c_1, c_2)

Oracle Dec$(i \in [n], j \in [n], (c_1, c_2), aad, info)$
13 **if** $j \in \Gamma_{\mathtt{pk}} \vee (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in \mathcal{E}$
14 **return** $\perp$
15 $K \leftarrow \text{ODecap}(j, c_1)$
16 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
17 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
18 **return** m

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$
19 **if** $|m_0| \neq |m_1| \vee (j \in \Gamma_{\mathtt{pk}} \wedge (i,j) \in \Gamma_{\mathtt{psk}})$
20 **return** $\perp$
21 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
22 **if** $j \notin \Gamma_{\mathtt{pk}}$
23 $K \leftarrow \text{OChall}(j)$
24 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
25 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
26 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
27 **return** (c_1, c_2)

Oracle RepPK$(j \in [n], pk)$
28 $pk_j \leftarrow pk$
29 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle RepPSK$(i \in [n], j \in [n], psk)$
30 $psk_{ij} \leftarrow psk$
31 $psk_{ji} \leftarrow psk$
32 $\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j),(j,i)\}$

into an adversary against the PRF security of function KS_1. Adversary $\mathcal{C}_1$ is formally constructed in Listing 23. Note that such an adversary needs at most q_c different PRF keys and at most $q_d + q_c$ queries to the PRF evaluation since they have to simulate KS_1 queries in the decapsulation and challenge oracle.

Game G_3. In G_3, the game aborts if there is a collision in the inputs to KS when called from oracles Enc or Chall in case of a non-replaced pre-shared key, i.e. $(i,j) \notin \Gamma_{\mathtt{psk}}$. The difference can be upper bounded by using the key entropy of KEM. To log the calls to KS, we introduce sets Λ_{Enc} and Λ_{Dec}, where Λ_{Enc} corresponds to calls after encapsulation operations, i.e. made from Enc and Chall, and Λ_{Dec} corresponds to calls after decapsulation operations, i.e. made from Dec. We distinguish these cases because the game only aborts for collisions in Enc or Chall (Line 14 and Line 54). If there is no such collision, i.e. no matching element in Λ_{Enc}, but the same inputs were queried in Dec before, i.e. a matching element in Λ_{Dec}, the stored outputs are used (Line 16 and Line 56). Otherwise, both oracles Enc and Chall store inputs and outputs to KS in Λ_{Enc} and proceed (Line 20 and Line 60). The changes in the decryption oracle are only for consistency. If there was a matching query before, the stored input is used (Line 36) and otherwise it is stored in set Λ_{Dec} (Line 39).

Since KS is deterministic, using the stored values does not change the winning probability which leaves us with the probability of an abort. If the game aborts, the inputs to KS collide which includes K. However, K is the output of a KEM encapsulation, i.e. the probability of a collision in K is $\frac{1}{2^\eta}$ for η being the key entropy of KEM. This leads to a probability of at most $\frac{q_e(q_e+q_c)}{2^\eta}$ for the abort in Enc and at most $\frac{q_c(q_e+q_c)}{2^\eta}$ for the abort

Listing 23: Adversary $\mathcal{C}_1$ against PRF security for KS_1 having access to oracle Eval and using an adversary $\mathcal{A}$ against $\mathsf{G}_1/\mathsf{G}_2$ of the proof of Theorem 1.

$\mathcal{C}_1^{\text{Eval}}$
01 **for** $i \in [n]$
02 $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 **for** $j \in [i]$
04 $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
05 $psk_{ji} \leftarrow psk_{ij}$
06 $\mathcal{E}, \mathcal{E}', \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}} \leftarrow \emptyset$
07 $\ell \leftarrow 0$
08 $b \xleftarrow{\$} \{0,1\}$
09 $b' \xleftarrow{\$} \mathcal{A}^{\text{Enc},\text{Dec},\text{Chall},\text{RepPK},\text{RepPSK}}(pk_1, \ldots, pk_n)$
10 **return** $[\![b = b']\!]$

Oracle Enc$(i \in [n], j \in [n], m, aad, info)$
11 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
12 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
13 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
14 **return** (c_1, c_2)

Oracle Dec$(i \in [n], j \in [n], (c_1, c_2), aad, info)$
15 **if** $sk_j = \bot \vee (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in \mathcal{E}$
16 **return** $\bot$
17 $K \leftarrow \mathsf{KEM.Decaps}(sk_j, c_1)$
18 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
19 **if** $\exists \ell' : (pk_j, c_1, \ell') \in \mathcal{E}'$
20 $(k, nonce) \leftarrow \text{Eval}(\ell, psk_{ij} || info)$ //eval oracle
21 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
22 **return** m

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$
23 **if** $|m_0| \neq |m_1| \vee (j \in \Gamma_{\mathtt{pk}} \wedge (i,j) \in \Gamma_{\mathtt{psk}})$
24 **return** $\bot$
25 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
26 **if** $j \notin \Gamma_{\mathtt{pk}}$
27 $\ell \leftarrow \ell + 1$
28 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk_j, c_1, \ell)\}$
29 $(k, nonce) \leftarrow \text{Eval}(\ell, psk_{ij} || info)$ //eval oracle
30 **else**
31 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
32 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
33 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
34 **return** (c_1, c_2)

Oracle RepPK$(j \in [n], pk)$
35 $(sk_j, pk_j) \leftarrow (\bot, pk)$
36 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle RepPSK$(i \in [n], j \in [n], psk)$
37 $psk_{ij} \leftarrow psk$
38 $psk_{ji} \leftarrow psk$
39 $\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j), (j,i)\}$

in Chall. All together, we obtain

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \frac{q_e^2 + q_c^2 + q_e q_c}{2^\eta}.$$

Game G_4. We modify G_4 by changing the output of KS in every oracle in case of a non-replaced pre-shared key, i.e. $(i,j) \notin \Gamma_{\mathtt{psk}}$, to uniformly random values (Line 19 for Enc, Line 38 for Dec, and Line 59 for Chall). The difference can be turned into a PRF adversary $\mathcal{C}_2$ against KS_2, i.e. KS keyed in the second component:

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e+q_d+q_c, q_e+q_d+q_c)\text{-PRF}}_{\mathcal{C}_2, \mathsf{KS}_2}$$

Note that for the challenge oracle, the assumption $(i,j) \notin \Gamma_{\mathtt{psk}}$ also implicitly holds. Since the change is only applied in the else case (Line 52) the receiver's key was replaced, i.e. $j \in \Gamma_{\mathtt{pk}}$. Additionally it is impossible that both $j \in \Gamma_{\mathtt{pk}}$ and $(i,j) \in \Gamma_{\mathtt{psk}}$ because in this case the challenge oracle would have returned $\bot$ due to the condition in Line 42. Thus, it holds $(i,j) \notin \Gamma_{\mathtt{psk}}$ in the else case which means that $psk_{i,j}$ was not replaced and therefore uniformly sampled during the key generation. We construct adversary $\mathcal{C}_2$ formally in Listing 24. For the pre-shared keys, the number of different keys is determined by the number of users. However, we need at most $q_e + q_d + q_c$ different PRF keys for the reduction and at most the same number of queries to the evaluation oracle.

Reduction to Game G_4. Game G_4 can be reduced to the CCA security of AEAD. We construct an adversary $\mathcal{D}$ against CCA security of AEAD using adversary $\mathcal{A}$ from G_4 as

Listing 24: Adversary $\mathcal{C}_2$ against PRF security for KS_2 having access to oracle Eval and using an adversary $\mathcal{A}$ against $\mathsf{G}_3/\mathsf{G}_4$ of the proof of Theorem 1.

$\underline{\mathcal{C}_2^{\text{Eval}}}$
01 **for** $i \in [n]$
02 $\quad(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\quad$**for** $j \in [i]$
04 $\quad\quad psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
05 $\quad\quad psk_{ji} \leftarrow psk_{ij}$
06 $\mathcal{E}, \mathcal{E}', \Gamma_{\mathsf{pk}}, \Gamma_{\mathsf{psk}} \leftarrow \emptyset$
07 $\ell \leftarrow 0$
08 $b \xleftarrow{\$} \{0,1\}$
09 $b' \xleftarrow{\$} \mathcal{A}^{\text{Enc},\text{Dec},\text{Chall},\text{RepPK},\text{RepPSK}}(pk_1, \ldots, pk_n)$
10 **return** $[\![b = b']\!]$

Oracle Enc$(i \in [n], j \in [n], m, aad, info)$
11 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
12 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
13 **if** $(i,j) \notin \Gamma_{\mathsf{psk}}$
14 $\quad$**if** $\exists(k', nonce', \ell') : (k', nonce', K, i, j, \ell', info) \in \Lambda_{Enc}$
15 $\quad\quad$**abort**
16 $\quad$**else if** $\exists(k', nonce', \ell') : (k', nonce', K, i, j, \ell', info) \in \Lambda_{Dec}$
17 $\quad\quad(k, nonce) \leftarrow (k', nonce')$
18 $\quad\quad\Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, i, j, \ell', info)\}$
19 $\quad$**else**
20 $\quad\quad\ell \leftarrow \ell + 1$
21 $\quad\quad(k, nonce) \leftarrow \text{Eval}(\ell, K||info)$ //eval query
22 $\quad\quad\Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, i, j, \ell, info)\}$
23 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
24 **return** (c_1, c_2)

Oracle Dec$(i \in [n], j \in [n], (c_1, c_2), aad, info)$
25 **if** $sk_j = \perp \vee (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in \mathcal{E}$
26 $\quad$**return** $\perp$
27 $K \leftarrow \mathsf{KEM.Decaps}(sk_j, c_1)$
28 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
29 **if** $\exists K' : (pk_j, c_1, K') \in \mathcal{E}'$
30 $\quad(k, nonce) \leftarrow \mathsf{KS}(K', psk_{ij}, info)$
31 $\quad$**if** $\exists(k', nonce') : (k', nonce', K', psk_{ij}, info) \in \Lambda_K$
32 $\quad\quad(k, nonce) \leftarrow (k', nonce')$
33 $\quad$**else**
34 $\quad\quad(k, nonce) \xleftarrow{\$} \mathcal{K} \times \{0,1\}^{N_{nonce}}$
35 $\quad\quad\Lambda_K \leftarrow \Lambda_K \cup \{(k, nonce, K, psk_{ij}, info)\}$
36 **else if** $(i,j) \notin \Gamma_{\mathsf{psk}}$
37 $\quad$**if** $\exists(k', nonce', \ell') : (k', nonce', K, i, j, \ell', info) \in \Lambda_{Enc} \cup \Lambda_{Dec}$
38 $\quad\quad(k, nonce) \leftarrow (k', nonce')$
39 $\quad$**else**
40 $\quad\quad\ell \leftarrow \ell + 1$
41 $\quad\quad(k, nonce) \leftarrow \text{Eval}(\ell, K||info)$ //eval query
42 $\quad\quad\Lambda_{Dec} \leftarrow \Lambda_{Dec} \cup \{(k, nonce, K, i, j, \ell, info)\}$
43 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
44 **return** m

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$
45 **if** $|m_0| \neq |m_1| \vee (j \in \Gamma_{\mathsf{pk}} \wedge (i,j) \in \Gamma_{\mathsf{psk}})$
46 $\quad$**return** $\perp$
47 $(c_1, K) \xleftarrow{\$} \mathsf{KEM.Encaps}(pk_j)$
48 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
49 **if** $j \notin \Gamma_{\mathsf{pk}}$
50 $\quad K \xleftarrow{\$} \mathcal{K}$
51 $\quad\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk_j, c_1, K)\}$
52 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
53 $\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$
54 $\quad\Lambda_K \leftarrow \Lambda_K \cup \{(k, nonce, K, psk_{ij}, info)\}$
55 **else**
56 $\quad$**if** $\exists(k', nonce', \ell') : (k', nonce', K, i, j, \ell', info) \in \Lambda_{Enc}$
57 $\quad\quad$**abort**
58 $\quad$**else if** $\exists(k', nonce', \ell') : (k', nonce', K, i, j, \ell', info) \in \Lambda_{Dec}$
59 $\quad\quad(k, nonce) \leftarrow (k', nonce')$
60 $\quad\quad\Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, i, j, \ell', info)\}$
61 $\quad$**else**
62 $\quad\quad\ell \leftarrow \ell + 1$
63 $\quad\quad(k, nonce) \leftarrow \text{Eval}(\ell, K||info)$ //eval query
64 $\quad\quad\Lambda_{Enc} \leftarrow \Lambda_{Enc} \cup \{(k, nonce, K, i, j, \ell, info)\}$
65 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
66 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
67 **return** (c_1, c_2)

Oracle RepPK$(j \in [n], pk)$
68 $(sk_j, pk_j) \leftarrow (\perp, pk)$
69 $\Gamma_{\mathsf{pk}} \leftarrow \Gamma_{\mathsf{pk}} \cup \{j\}$

Oracle RepPSK$(i \in [n], j \in [n], psk)$
70 $psk_{ij} \leftarrow psk$
71 $psk_{ji} \leftarrow psk$
72 $\Gamma_{\mathsf{psk}} \leftarrow \Gamma_{\mathsf{psk}} \cup \{(i,j), (j,i)\}$

described in Listing 25. Adversary $\mathcal{D}$ simulates the oracles using their own encryption oracle Enc_{AEAD} and decryption oracle Dec_{AEAD}. The previous game modifications ensure that the keys and nonces for the AEAD encryptions are uniformly random and that there is no collision between oracle Enc and Chall which are both calling the encryption function. This matches the conditions of the CCA game for AEAD. The difficulty is that the simulator has to recognize if a query to the decryption oracle corresponds to the same key and nonce as for a previous call without knowing the actual key and nonce. This can be solved by keeping track of the queried parameter since KS deterministically relates their inputs and outputs (set $\mathcal{E}'$ and sets Λ_{Enc} and Λ_{Dec} from the previous games).

It remains to show that the simulation of the decryption oracle is sound. This is the case if the CCA decryption oracle is never queried on challenge parameters, i.e. on tuples (ℓ, c_2, aad) corresponding to a CCA encryption query (see Line 11 in the CCA experiment for AEAD). Assume that it is queried on such a tuple, i.e. (ℓ, c_2, aad) was part of a challenge query. However, since the inputs to KS must also be equal (condition in Line 11 and one-to-one relation from instance ℓ to challenge query), values $K, psk_{i,j}$, and *info* also match the challenge query. If c_1 also equals the one from the challenge query, all relevant parameters are the same and the parameters can be found in set $\mathcal{E}$. However, in this case the decryption oracle aborts after the first check in Line 25, returns $\perp$ and the CCA oracle is not queried. The remaining case is a query for a c_1 which is not equal to the one stored in $\mathcal{E}$. This case can also not occur due to the condition in Line 25 which guarantees that the public key was not replaced ($sk_j \neq \perp$) because otherwise the oracle returns $\perp$ again. If the public key was not replaced, there exists an entry in set $\mathcal{E}'$ connecting K and c_1 which means that we have the same c_1 whenever we have the same K as for a previous challenge query. In total, the decryption oracle of the CCA game never returns $\perp$ and the simulation is sound which completes the proof:

$$\Pr[\mathsf{G}_4 \Rightarrow 1] \leq \mathsf{Adv}^{(q_e+q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}.$$

Putting everything together, we obtain the stated bound. □

B.2 Proof of Theorem 2

Proof. We describe several games depicted in Listing 26.

Game G_0. Let G_0 be the $(n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})$-Insider-CCA game for the construction pskAPKE[AKEM, KS, AEAD], i.e.,

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})\text{-Insider-CCA}(\mathcal{A}) \Rightarrow 1].$$

Game G_1. In this game, we make the following changes. If the challenge oracle is queried on a receiver index j which was not replaced, i.e. $j \notin \Gamma_{\mathtt{pk}}$, we replace the KEM key K by a uniformly random value from the key space $\mathcal{K}$. The result is then stored together with the receiver's public key and the ciphertext in set $\mathcal{E}'$. We further change the decryption oracle by replacing the KEM key K by the one from set $\mathcal{E}'$ if there already exists such a key for the queried ciphertext c_1 and the receiver's public key pk_j. Distinguishing these changes can be turned into an adversary $\mathcal{B}$ against (n, q_d, q_{c1})-CCA security of KEM as depicted in Listing 27.

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_e,q_d,q_c,r_{sk})\text{-Insider-CCA}}_{\mathcal{B},\mathsf{AKEM}}.$$

Note that $\mathcal{B}$ can simulate all oracles by their own or with the oracles provided by their own experiment. For RepPK and RepSK, the simulator can store the values and answer other oracle queries accordingly. Note that we introduce another set $\mathcal{R}$ to store the indices for which the public key was replaced to differentiate public key replacements from secret key replacements since we handle them differently. [6] The encryption and decryption oracle can

[6] This is implicitly done in the security notion, e.g. decryption queries are still possible for replaced receiver's secret keys but not for replaced receiver's public keys since the experiment is not able to decrypt anymore.

Listing 25: Adversary $\mathcal{D}$ against $(q_e + q_c, q_d)$-CCA security for AEAD having access to oracles ENC_{AEAD} and DEC_{AEAD}. Adversary $\mathcal{A}$ against G_4 makes at most q_d queries to DEC and q_c queries to CHALL.

```
D^{Enc_AEAD, Dec_AEAD}
01 for i ∈ [n]
02   (sk_i, pk_i) ←$ Gen
03   for j ∈ [i]
04     psk_ij ←$ GenPSK
05     psk_ji ← psk_ij
06 E, E', Γ_pk, Γ_psk ← ∅
07 ℓ ← 0
08 b' ←$ A^{Enc,Dec,Chall,RepPK,RepPSK}(pk_1, ..., pk_n)
09 return b'

Oracle Enc(i ∈ [n], j ∈ [n], m, aad, info)
10 (c_1, K) ←$ KEM.Encaps(pk_j)
11 (k, nonce) ← KS(K, psk_ij, info)
12 if (i, j) ∉ Γ_psk
13   if ∃ℓ' : (ℓ', K, i, j, info) ∈ Λ_Enc
14     abort
15   else if ∃ℓ' : (ℓ', K, i, j, info) ∈ Λ_Dec
16     c_2 ← Enc_AEAD(ℓ', m, m, aad)
17     Λ_Enc ← Λ_Enc ∪ {(ℓ', K, i, j, info)}
18   else
19     ℓ ← ℓ + 1
20     c_2 ← Enc_AEAD(ℓ, m, m, aad)
21     Λ_Enc ← Λ_Enc ∪ {(ℓ, K, i, j, info)}
22 else
23   c_2 ← AEAD.Enc(k, m, aad, nonce)
24 return (c_1, c_2)

Oracle Dec(i ∈ [n], j ∈ [n], (c_1, c_2), aad, info)
25 if sk_j = ⊥ ∨ (pk_j, psk_ij, (c_1, c_2), aad, info) ∈ E
26   return ⊥
27 K ← KEM.Decaps(sk_j, c_1)
28 (k, nonce) ← KS(K, psk_ij, info)
29 if ∃ℓ' : (pk_j, c_1, ℓ') ∈ E'
30   m ← Dec_AEAD(ℓ', c_2, aad)
31 else if (i, j) ∉ Γ_psk
32   if ∃ℓ' : (ℓ', K, i, j, info) ∈ Λ_Enc ∪ Λ_Dec
33     m ← Dec_AEAD(ℓ', c_2, aad)
34   else
35     ℓ ← ℓ + 1
36     m ← Dec_AEAD(ℓ, c_2, aad)
37     Λ_Dec ← Λ_Dec ∪ {(ℓ, K, i, j, info)}
38 else
39   m ← AEAD.Dec(k, c_2, aad, nonce)
40 return m

Oracle Chall(i ∈ [n], j ∈ [n], m_0, m_1, aad, info)
41 if |m_0| ≠ |m_1| ∨ (j ∈ Γ_pk ∧ (i, j) ∈ Γ_psk)
42   return ⊥
43 (c_1, K) ←$ KEM.Encaps(pk_j)
44 (k, nonce) ← KS(K, psk_ij, info)
45 if j ∉ Γ_pk
46   ℓ ← ℓ + 1
47   E' ← E' ∪ {(pk_j, c_1, ℓ)}
48   c_2 ← Enc_AEAD(ℓ, m_0, m_1, aad)
49 else
50   if ∃ℓ' : (ℓ', K, i, j, info) ∈ Λ_Enc
51     abort
52   else if ∃ℓ' : (ℓ', K, i, j, info) ∈ Λ_Dec
53     c_2 ← Enc_AEAD(ℓ', m_0, m_1, aad)
54     Λ_Enc ← Λ_Enc ∪ {(ℓ', K, i, j, info)}
55   else
56     ℓ ← ℓ + 1
57     c_2 ← Enc_AEAD(ℓ, m_0, m_1, aad)
58     Λ_Enc ← Λ_Enc ∪ {(ℓ, K, i, j, info)}
59 E ← E ∪ {(pk_j, psk_ij, (c_1, c_2), aad, info)}
60 return (c_1, c_2)

Oracle RepPK(j ∈ [n], pk)
61 (sk_j, pk_j) ← (⊥, pk)
62 Γ_pk ← Γ_pk ∪ {j}

Oracle RepPSK(i ∈ [n], j ∈ [n], psk)
63 psk_ij ← psk
64 psk_ji ← psk
65 Γ_psk ← Γ_psk ∪ {(i, j), (j, i)}
```

be simulated by the encapsulation and decapsulation oracle provided by the AKEM Insider-CCA game. If the challenge oracle is queried on non-replaced receiver keys, i.e. $j \notin \Gamma_{\mathsf{pk}}$ (Line 24), $\mathcal{B}$ can use their own challenge. If $\mathcal{B}$ is in the real case of their (n, q_d, q_c)-CCA game, they perfectly simulate G_0. If they are in the random case, they perfectly simulate G_1.

Game G_2. Game G_1 is exactly Game G_1 in the proof of Theorem 1. Hence, we can apply the same game modifications as in the previous proof to obtain the same final game, called

Listing 26: Games $\mathsf{G}_0 - \mathsf{G}_1$ for the proof of Theorem 2.

$\underline{\mathsf{G}_0 - \mathsf{G}_1}$

for $i \in [n]$
  $(sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
  for $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \mathcal{E}', \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chall},\textsc{RepPK},\textsc{RepSK},\textsc{RepPSK}}$
return $[\![b = b']\!]$

Oracle $\textsc{Enc}(i \in [n], j \in [n], m, aad, info)$

if $sk_i = \bot$
  return $\bot$
$(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk_j)$
$(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
$c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
return (c_1, c_2)

Oracle $\textsc{Dec}(i \in [n], j \in [n], (c_1, c_2), aad, info)$

if $sk_j = \bot \vee (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in \mathcal{E}$
  return $\bot$
if $\exists K : (pk_i, pk_j, c, K) \in \mathcal{E}'$ $\quad /\!\!/ \mathsf{G}_1$
  return K $\quad /\!\!/ \mathsf{G}_1$
$K \leftarrow \mathsf{AuthDecap}(pk_i, sk_j, c_1)$
$(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
$m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
return m

Oracle $\textsc{Chall}(i \in [n], j \in [n], m_0, m_1, aad, info)$

if $|m_0| \neq |m_1| \vee sk_i = \bot \vee (j \in \Gamma_{\mathtt{pk}} \wedge (i,j) \in \Gamma_{\mathtt{psk}})$
  return $\bot$
$(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk_j)$
if $j \notin \Gamma_{\mathtt{pk}}$ $\quad /\!\!/ \mathsf{G}_1$
  $K \xleftarrow{\$} \mathcal{K}$ $\quad /\!\!/ \mathsf{G}_1$
  $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk_i, pk_j, c, K)\}$ $\quad /\!\!/ \mathsf{G}_1$
$(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
$c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
return (c_1, c_2)

Oracle $\textsc{RepPK}(j \in [n], pk)$

$(sk_j, pk_j) \leftarrow (pk, \bot)$
$\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle $\textsc{RepSK}(j \in [n], sk)$

$(sk_j, pk_j) \leftarrow (sk, \mu(sk))$
$\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$

Oracle $\textsc{RepPSK}(i \in [n], j \in [n], psk)$

$psk_{ij} \leftarrow psk$
$psk_{ji} \leftarrow psk$
$\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j), (j,i)\}$

Listing 27: Adversary $\mathcal{B}$ against $(n, q_e, q_d, q_c, r_{pk}, r_{sk})$-Insider-CCA security for AKEM with access to oracles OAEncap, OADecap, OChall, and ORepSK. Adversary $\mathcal{A}$ against $\mathsf{G}_0/\mathsf{G}_1$ makes at most q_e queries to Enc, at most q_d queries to Dec, at most q_c queries to Chall, at most r_{pk} queries to RepPK, at most r_{sk} queries to RepSK, and at most r_{psk} queries to RepPSK.

$\mathcal{B}^{\text{OAEncap},\text{OADecap},\text{OChall},\text{ORepSK}}(pk_1, \ldots, pk_n)$
01 **for** $i \in [n]$
02 **for** $j \in [i]$
03 $psk_{ij} \xleftarrow{\$} \mathcal{K}_{psk}$
04 $psk_{ji} \leftarrow psk_{ij}$
05 $\mathcal{E}, \Gamma_{\mathtt{pk}}, \Gamma_{\mathtt{psk}}, \mathcal{R} \leftarrow \emptyset$
06 $b \xleftarrow{\$} \{0,1\}$
07 $b' \xleftarrow{\$} \mathcal{A}^{\text{Enc},\text{Dec},\text{Chall},\text{RepPK},\text{RepSK},\text{RepPSK}}(pk_1, \ldots, pk_n)$
08 **return** $[\![b = b']\!]$

Oracle Enc$(i \in [n], j \in [n], m, aad, info)$
09 **if** $i \in \mathcal{R}$
10 **return** $\perp$
11 $(c_1, K) \xleftarrow{\$} \text{OAEncap}(i, pk_j)$
12 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
13 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
14 **return** (c_1, c_2)

Oracle Dec$(i \in [n], j \in [n], (c_1, c_2), aad, info)$
15 **if** $j \in \mathcal{R} \vee (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in \mathcal{E}$
16 **return** $\perp$
17 $K \leftarrow \text{OADecap}(pk_i, j, c_1)$
18 **if** $K = \perp$
19 **return** $\perp$
20 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
21 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
22 **return** m

Oracle Chall$(i \in [n], j \in [n], m_0, m_1, aad, info)$
23 **if** $|m_0| \neq |m_1| \vee i \in \mathcal{R} \vee (j \in \Gamma_{\mathtt{pk}} \wedge (i,j) \in \Gamma_{\mathtt{psk}})$
24 **return** $\perp$
25 $(c_1, K) \xleftarrow{\$} \text{OAEncap}(i, pk_j)$
26 **if** $j \notin \Gamma_{\mathtt{pk}}$
27 $(c_1, K) \xleftarrow{\$} \text{OChall}(i, j)$
28 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
29 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
30 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
31 **return** (c_1, c_2)

Oracle RepPK$(j \in [n], pk)$
32 $pk_j \leftarrow pk$
33 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$
34 $\mathcal{R} \leftarrow \mathcal{R} \cup \{j\}$

Oracle RepSK$(j \in [n], sk)$
35 $(sk_j, pk_j) \leftarrow (sk, \mu(sk))$
36 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{j\}$
37 ORepSK(j, sk)

Oracle RepPSK$(i \in [n], j \in [n], psk)$
38 $psk_{ij} \leftarrow psk$
39 $psk_{ji} \leftarrow psk$
40 $\Gamma_{\mathtt{psk}} \leftarrow \Gamma_{\mathtt{psk}} \cup \{(i,j), (j,i)\}$

G_2 This results in the same bound

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_c, q_d+q_c)\text{-PRF}}_{\mathcal{C}_1,\mathsf{KS}_1} + \mathsf{Adv}^{(q_e+q_d+q_c, q_e+q_d+q_c)\text{-PRF}}_{\mathcal{C}_2,\mathsf{KS}_2} + \frac{q_e^2 + q_c^2 + q_e q_c}{2^\eta},$$

with a final winning probability of

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \mathsf{Adv}^{(q_e+q_c, q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}.$$

Putting everything together results in the stated bound. □

B.3 Proof of Theorem 3

Proof. We describe a series of games in Listing 28.

Listing 28: Games $\mathsf{G}_0 - \mathsf{G}_7$ for the proof of Theorem 3.

$\underline{\mathsf{G}_0 - \mathsf{G}_7}$
01 **for** $i \in [n]$
02 $(sk_i, pk_i) \xleftarrow{\$} \mathsf{GenSK}$
03 **for** $j \in [i]$
04 $psk_{ij} \xleftarrow{\$} \mathcal{K}_{psk}$
05 $psk_{ji} \leftarrow psk_{ij}$
06 $\mathcal{E}, \Gamma_{\mathsf{psk}}, \mathcal{E}', \hat{\mathcal{E}}, \Lambda \leftarrow \emptyset$
07 $(i^*, j^*, (c_1^*, c_2^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
08 **return** $[\![(i^*, j^*) \notin \Gamma_{\mathsf{psk}} \wedge sk_{i^*} \neq \bot$
$\wedge (pk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \notin \mathcal{E}$
$\wedge \mathsf{pskDec}(sk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \neq \bot]\!]$

$\underline{\text{Oracle } \textsc{Enc}(i \in [n], j \in [n], m, aad, info)}$
09 $(c_1, K) \xleftarrow{\$} \mathsf{Encaps}(pk_j)$
10 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
11 **if** $sk_j \neq \bot$ // $\mathsf{G}_1 - \mathsf{G}_7$
12 $K \xleftarrow{\$} \mathcal{K}$ // $\mathsf{G}_2 - \mathsf{G}_7$
13 $\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(pk_j, c_1, K)\}$ // $\mathsf{G}_2 - \mathsf{G}_7$
14 **if** $\exists k', nonce' : (k', nonce', i, j, K, psk_{ij}, info) \in \Lambda$ // $\mathsf{G}_1 - \mathsf{G}_7$
15 **abort** // $\mathsf{G}_3 - \mathsf{G}_7$
16 $(k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_1 - \mathsf{G}_7$
17 $(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_4 - \mathsf{G}_7$
18 $\Lambda \leftarrow \Lambda \cup \{(k, nonce, i, j, K, psk_{ij}, info)\}$ // $\mathsf{G}_1 - \mathsf{G}_7$
19 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
20 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, (c_1, c_2), aad, info)\}$
21 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(k, nonce, m, (c_1, c_2), aad)\}$ // $\mathsf{G}_6 - \mathsf{G}_7$
22 **return** (c_1, c_2)

$\underline{\text{Oracle } \textsc{Dec}(i \in [n], j \in [n], (c_1, c_2), aad, info)}$
23 **if** $sk_j = \bot$
24 **return** $\bot$
25 $K \leftarrow \mathsf{Decaps}(sk_j, c_1)$
26 **if** $\exists K' : (pk_j, c_1, K') \in \hat{\mathcal{E}}$ // $\mathsf{G}_2 - \mathsf{G}_7$
27 $K \leftarrow K'$ // $\mathsf{G}_2 - \mathsf{G}_7$
28 $(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$
29 **if** $(i, j) \notin \Gamma_{\mathsf{psk}}$ // $\mathsf{G}_1 - \mathsf{G}_7$
30 **if** $\exists k', nonce' : (k', nonce', i, j, K, psk_{ij}, info) \in \Lambda$ // $\mathsf{G}_1 - \mathsf{G}_7$
31 $(k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_1 - \mathsf{G}_7$
32 **else** // $\mathsf{G}_1 - \mathsf{G}_7$
33 $(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_5 - \mathsf{G}_7$
34 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // $\mathsf{G}_1 - \mathsf{G}_7$
35 $m \leftarrow \bot$ // G_7
36 **if** $\exists m' : (k, nonce, m, (c_1, c_2), aad) \in \mathcal{E}'$ // $\mathsf{G}_6 - \mathsf{G}_7$
37 $m \leftarrow m'$ // $\mathsf{G}_6 - \mathsf{G}_7$
38 **else** // $\mathsf{G}_1 - \mathsf{G}_7$
39 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // $\mathsf{G}_1 - \mathsf{G}_7$
40 $\Lambda \leftarrow \Lambda \cup \{(k, nonce, i, j, K, psk_{ij}, info)\}$ // $\mathsf{G}_1 - \mathsf{G}_7$
41 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // G_0
42 **return** m

$\underline{\text{Oracle } \textsc{RepPK}(j \in [n], pk)}$
43 $(sk_j, pk_j) \leftarrow (\bot, pk)$

$\underline{\text{Oracle } \textsc{RepPSK}(i \in [n], j \in [n], psk)}$
44 $psk_{ij} \leftarrow psk$
45 $psk_{ji} \leftarrow psk$
46 $\Gamma_{\mathsf{psk}} \leftarrow \Gamma_{\mathsf{psk}} \cup \{(i, j), (j, i)\}$

Game G_0. This is the $(n, q_e, q_d, r_{pk}, r_{psk})$-Auth game for $\mathsf{pskPKE}[\mathsf{KEM}, \mathsf{KS}, \mathsf{AEAD}]$, thus we have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, r_{pk}, r_{psk})\text{-Auth}(\mathcal{A}) \Rightarrow 1].$$

Game G_1. We introduce a set Λ to log the outputs of KS produced in Enc and Dec. More specifically, Λ contains elements $k, nonce, i, j, K, psk, info$ of the form that $k, nonce$ is the output of a KS call, $K, psk, info$, the input, and i, j sender and receiver index for

that query. If KS was already called with the same parameters, we can take the stored value. For the encryption oracle, this is only done for queries for which the corresponding receiver has not been corrupted, i.e. for queries with $sk_j \neq \bot$ (Line 11). For the decryption oracle, we use the stored value only if the corresponding *psk* has not been replaced, i.e. for queries on indices $(i, j) \notin \Gamma_{\mathtt{psk}}$ (Line 29). Since the change is only of formal nature, we have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1].$$

Game G_2. If the receiver's key was not replaced, we replace the KEM secret K by a uniformly random value in the encryption oracle Enc The result is stored to answer the decryption oracle correctly. The change corresponds to an (n, q_d, q_c)-CCA adversary against KEM. For simulating the RepPK oracle, we refer to the proof of Theorem 1 which presents a solution for the same case. We obtain

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_d,q_e)\text{-}\mathsf{CCA}}_{\mathcal{B},\mathsf{KEM}}.$$

Game G_3. We abort in the encryption oracle if the corresponding receiver's key was not replaced and there already exists an entry in Λ with the queried parameters. Since K was randomly chosen in the previous game, the probability of having such an entry is at most $\frac{|\Lambda|}{|\mathcal{K}|}$. Note that Λ is filled with an additional element at most once per Enc/Dec query. This yields the following advantage:

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \frac{q_e(q_e + q_d - 1)}{|\mathcal{K}|}.$$

Game G_4. If the receiver's key was not replaced and we do not abort, we replace the output of KS in the encryption oracle with uniformly random values of the respective domain. The changes in Game G_1 ensure consistent outputs of KS, i.e. queries on Enc or Dec with the same parameters lead to the same output of KS. Hence, games G_3 and G_4 can only be distinguished by distinguishing the real output of KS from a uniformly random one. This can be turned into an adversary against PRF security of KS_1, i.e. keyed on the first input. Note that K is chosen uniformly at random due to the changes in Game G_2. There are at most q_e different instances for the PRF and at most the same number of queries resulting in

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e,q_e)\text{-}\mathsf{PRF}}_{\mathcal{C}_1,\mathsf{KS}_1}.$$

Game G_5. Now we apply changes to the decryption oracle analogously to the changes from the previous game. If the corresponding *psk* has not been revealed, we replace the output of KS in the decryption oracle with uniformly random values of the respective domain. Again, we have consistent outputs of KS, i.e. queries on Enc and Dec with the same parameters lead to the same output of KS. Games G_4 and G_5 can only be distinguished by distinguishing the real output of KS from a uniformly random one. This can be turned into an adversary against PRF security of KS_2, i.e. keyed on the second input. Note that *psk* was not replaced and is therefore uniformly random which matches the PRF game. There are at most q_d different instances for the PRF and at most the same number of queries resulting in

$$|\Pr[\mathsf{G}_4 \Rightarrow 1] - \Pr[\mathsf{G}_5 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_d,q_d)\text{-}\mathsf{PRF}}_{\mathcal{C}_2,\mathsf{KS}_2}.$$

Game G_6. We now store the results of encryption queries to replace the actual decryption in the case of an honest oracle query, i.e. $(i,j) \notin \Gamma_{\mathtt{psk}}$, by the stored results. This is only a formal change since the correctness of the AEAD implies same win probabilities for this game change:

$$\Pr[\mathsf{G}_5 \Rightarrow 1] = \Pr[\mathsf{G}_6 \Rightarrow 1].$$

Game G_7. In this game, we replace the actual decryption in an honest decryption oracle query with $\perp$. Distinguishing the game difference can be turned into an adversary against INT-CTXT security of the underlying AEAD. We depict the adversary in Listing 29. There are $q_e + q_d$ instances and adversary $\mathcal{D}_1$ makes at most q_d queries to their decryption oracle $\textsc{Dec}_{\text{AEAD}}$. Thus, we obtain

$$|\Pr[\mathsf{G}_6 \Rightarrow 1] - \Pr[\mathsf{G}_7 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e+q_d,q_d)\text{-INT-CTXT}}_{\mathcal{D}_1,\mathsf{AEAD}}.$$

Reduction to Game G_7. We can reduce INT-CTXT security to Game G_7. An adversary against INT-CTXT can be used to simulate G_7. The adversary can simulate the decryption oracle since in cases $(i,j) \notin \Gamma_{\mathtt{psk}}$, they can output $\perp$ (or the original encryption if it was produced during the experiment). In cases $(i,j) \in \Gamma_{\mathtt{psk}}$, they can compute the output by their own. For the encryption oracle, the adversary can compute the output on their own in cases $sk_j = \perp$ and use their own encryption oracle in cases $sk_j \neq \perp$ since key k and *nonce* are uniformly random similarly to the adversary in Listing 29.

The output of the adversary against game G_7 can then be used to issue a decryption query in the INT-CTXT experiment on either a new instance or a previous instance if the output parameters $i^*, j^*, c_1^*, info^*$ match with that instance. Matching parameters can be identified by computing $K^* \leftarrow \mathsf{Decaps}(sk_{j^*}, c_1^*)$ and comparing $(i^*, j^*, K^*, psk_{i^*j^*}, info^*)$ to set Λ similar to Line 14 or Line 31 in Listing 29. If the adversary against G_7 wins, the adversary against the INT-CTXT experiment has a valid ciphertext which does not decrypt to $\perp$ due to the winning condition of G_7. Further, since the ciphertext must not decrypt to $\perp$, it also holds $sk_j^* \neq \perp$ which means that the parameters correspond to an actual instance of the simulator. That means the adversary can distinguish between the real or random case since the result of the decryption query must be unequal to $\perp$ in the real case. This results in

$$\Pr[\mathsf{G}_7 \Rightarrow 1] \leq \mathsf{Adv}^{(q_e+1,1)\text{-INT-CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}}.$$

Putting everything together we obtain the stated bound.

□

C Proofs for the Hybrid Post-Quantum APKE

C.1 Proof of Theorem 5

Proof. In order to prove Theorem 5, we will prove the following two inequalities.

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} &\leq \mathsf{Adv}^{(n+1,q_e,q_d,q_c,q_c)\text{-Insider-CCA}}_{\mathcal{B},\mathsf{AKEM}_1} \\ &\quad + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_1} + \mathsf{Adv}^{(q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}\end{aligned} \tag{1}$$

Listing 29: Adversary $\mathcal{D}_1$ against $(q_e + q_d, q_d)$-INT-CTXT security for AEAD having access to oracles ENC_{AEAD} and DEC_{AEAD} from the proof of Theorem 3. Adversary $\mathcal{A}$ against $\mathsf{G}_5/\mathsf{G}_6$ makes at most q_e queries to ENC, at most q_d queries to DEC, at most r_{pk} queries to REPPK, and at most r_{psk} queries to REPPSK.

$\mathcal{D}_1^{\text{ENC}_{\text{AEAD}},\text{DEC}_{\text{AEAD}}}$

```
for i ∈ [n]
  (sk_i, pk_i) ←$ GenSK
  for j ∈ [i]
    psk_ij ←$ K_psk
    psk_ji ← psk_ij
E, Γ_psk, Ê, Λ ← ∅
ℓ ← 0
(i*, j*, (c1*, c2*), aad*, info*) ←$ A^{Enc,Dec,RepPK,RepPSK}(pk_1, ..., pk_n)
return ⟦(i*, j*) ∉ Γ_psk ∧ sk_j* ≠ ⊥
  ∧ (pk_j*, psk_i*j*, (c1*, c2*), aad*, info*) ∉ E
  ∧ pskDec(sk_j*, psk_i*j*, (c1*, c2*), aad*, info*) ≠ ⊥⟧
```

Oracle $\text{ENC}(i \in [n], j \in [n], m, aad, info)$

```
(c1, K) ←$ Encaps(pk_j)
if sk_j ≠ ⊥
  K ←$ K
  Ê ← Ê ∪ {(pk_j, c1, K)}
  if ∃ℓ : (ℓ, i, j, K, psk_ij, info) ∈ Λ
    abort
  else
    ℓ ← ℓ + 1
    c2 ← Enc_AEAD(ℓ, m, aad)
else
  (k, nonce) ← KS(K, psk_ij, info)
  c2 ← AEAD.Enc(k, m, aad, nonce)
Λ ← Λ ∪ {(ℓ, i, j, K, psk_ij, info)}
E ← E ∪ {(pk_j, psk_ij, (c1, c2), aad, info)}
return (c1, c2)
```

Oracle $\text{DEC}(i \in [n], j \in [n], (c_1, c_2), aad, info)$

```
if sk_j = ⊥
  return ⊥
K ← Decaps(sk_j, c1)
if ∃K' : (pk_j, c1, K') ∈ Ê
  K ← K'
if (i, j) ∉ Γ_psk
  if ∃ℓ : (ℓ, i, j, K, psk_ij, info) ∈ Λ
    m ← Dec_AEAD(ℓ, c2, aad)
  else
    ℓ ← ℓ + 1
    m ← Dec_AEAD(ℓ, c2, aad)
else
  (k, nonce) ← KS(K, psk_ij, info)
  m ← AEAD.Dec(k, c2, aad, nonce)
Λ ← Λ ∪ {(ℓ, i, j, K, psk_ij, info)}
return m
```

Oracle $\text{REPPK}(j \in [n], pk)$

```
(sk_j, pk_j) ← (⊥, pk)
```

Oracle $\text{REPPSK}(i \in [n], j \in [n], psk)$

```
psk_ij ← psk
psk_ji ← psk
Γ_psk ← Γ_psk ∪ {(i, j), (j, i)}
```

and

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq \mathsf{Adv}^{(n,q_e,q_d,q_c,q_c)\text{-Insider-CCA}}_{\mathcal{B},\mathsf{AKEM}_2} + \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_2} + \mathsf{Adv}^{(q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}. \tag{2}$$

Inequality (1): First, we show

$$\mathsf{Adv}^{(n,q_e,q_d,q_c)\text{-Insider-CCA}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq \mathsf{Adv}^{(n+1,q_e,q_d,q_c,q_e+q_d,q_c,q_e+q_d+q_c)\text{-Insider-CCA}}_{\tilde{\mathcal{A}},\mathsf{pskAPKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]}. \tag{3}$$

Then, we can apply a similar proof as for Theorem 2 with the following change. Due to the special structure of adversary $\tilde{\mathcal{A}}$, we can skip the game which introduces the PRF security loss from querying KS in the second input (the *psk* part). This is because none of the conditions which are used for the PRF advantage are met. In detail, for the challenge oracle, it always holds $j \notin \Gamma_{\mathsf{pk}}$ since the challenge oracle may only be queried for an honest receiver. The same holds for the loss based on the entropy of the KEM key which eliminates collisions as a preparation for the *psk* part of the PRF and the number of different keys in the CCA experiment for AEAD. Due to the structure of the adversary in Listing 30 it

never holds $(i,j) \notin \Gamma_{\mathtt{psk}}$. This results in

$$\begin{aligned}\mathsf{Adv}^{(n+1,q_e,q_d,q_c,q_e+q_d,q_c,q_e+q_d+q_c)\text{-Insider-CCA}}_{\tilde{\mathcal{A}},\mathsf{pskAPKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} &\leq \mathsf{Adv}^{(n+1,q_e,q_d,q_c,q_c)\text{-Insider-CCA}}_{\mathcal{B},\mathsf{AKEM}_1} \\ &+ \mathsf{Adv}^{(q_c,q_d+q_c)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_1} \\ &+ \mathsf{Adv}^{(q_c,q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}.\end{aligned}$$

To prove (3), we use an (n,q_e,q_d,q_c,q_c)-Insider-CCA adversary against APKE to construct an adversary $\tilde{\mathcal{A}}$ against $(n+1,q_e,q_d,q_c,q_e+q_d,q_c,q_e+q_d+q_c)$-Insider-CCA security against pskAPKE as shown in Listing 30. One can see that $\tilde{\mathcal{A}}$ uses one query to RepPK for each query to Enc and pskDec, i.e. $r_{pk} = q_e + q_d$, and one query to RepSK for each Chall query, i.e. $r_{sk} = q_c$. Additionally, we need to replace the "psk" for each query resulting in $r_{psk} = q_e + q_d + q_c$. This fulfills the adversarial query bounds from inequality (3).

Listing 30: $(n+1,q_e,q_d,q_c,q_e+q_d,q_c,q_e+q_d+q_c)$-Insider-CCA adversary $\tilde{\mathcal{A}}$ against $\mathsf{pskAPKE}[\mathsf{AKEM}_1,\mathsf{KS},\mathsf{AEAD}]$ using an (n,q_e,q_d,q_c)-Insider-CCA adversary $\mathcal{A}$ against $\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]$ for Inequality (3) of the proof of Theorem 5.

$\tilde{\mathcal{A}}^{\text{OEnc,ODec,OChall,ORepPK,ORepSK,ORepPSK}}(pk^1_1,\ldots,pk^1_{n+1})$

01 **for** $i \in [n]$
02 $\quad(sk^2_i, pk^2_i) \xleftarrow{\$} \mathsf{AKEM}_2.\mathsf{Gen}$
03 $\quad pk_i \leftarrow (pk^1_i, pk^2_i)$
04 $b' \leftarrow \mathcal{A}^{\text{AEnc,ADec,Chall}}(pk_1,\ldots,pk_n)$
05 **return** b'

Oracle $\text{AEnc}(i \in [n], (pk^1, pk^2), m, aad, info)$

06 **if** $\exists j' \in [n] : pk^1 = pk^1_{j'}$
07 $\quad j \leftarrow j'$
08 **else**
09 $\quad \text{ORepPK}(n+1, pk^1)$
10 $\quad j \leftarrow n+1$
11 $(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk^2_i, pk^2)$
12 $\text{ORepPSK}(i, j, K')$
13 $c \leftarrow \text{OEnc}(i, j, m, aad, info)$
14 **return** (c, c')

Oracle $\text{ADec}((pk^1, pk^2), j \in [n], (c, c'), aad, info)$

15 **if** $\exists i' \in [n] : pk^1 = pk^1_{i'}$
16 $\quad i \leftarrow i'$
17 **else**
18 $\quad \text{ORepPK}(n+1, pk^1)$
19 $\quad i \leftarrow n+1$
20 $K' \leftarrow \mathsf{AuthDecap}_2(pk^2, sk^2_j, c')$
21 $\text{ORepPSK}(i, j, K')$
22 $m \leftarrow \text{ODec}(i, j, c, aad, info)$
23 **return** m

Oracle $\text{Chall}((sk^1, sk^2), j \in [n], m_0, m_1, aad, info)$

24 $\text{ORepSK}(n+1, sk^1)$
25 $(c', K')\mathsf{AuthEncap}_2(sk^2, pk^2_j)$
26 $\text{ORepPSK}(n+1, j, K')$
27 $c \leftarrow \text{OChall}(n+1, j, m_0, m_1, aad, info)$
28 **return** (c, c')

Inequality (2): Proving the second inequality (2), we start with the game (n,q_e,q_d,q_c)-Insider-CCA for $\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]$ denoted by G_0 in Listing 31.

Game G_1. In Game G_1, we replace the AKEM secret of AKEM_2 in the challenge oracle by a uniformly random value. This can be directly turned into an adversary against AKEM_2's Insider-CCA security:

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_e,q_d,q_c,q_c)\text{-Insider-CCA}}_{\mathcal{B},\mathsf{AKEM}_2}.$$

Game G_2. In G_2, the challenge oracle is changed such that the output of KS is replaced by uniformly random values. Additionally, the decryption oracle is modified in the same way but only if there is a corresponding element in $\mathcal{E}'$. Since K', which is the second key to KS,

Listing 31: Games $\mathsf{G}_0 - \mathsf{G}_2$ for the proof of Inequality (2) of Theorem 5.

$\underline{\mathsf{G}_0 - \mathsf{G}_2}$
01 **for** $i \in [n]$
02 $\quad (sk_i^1, pk_i^1) \xleftarrow{\$} \mathsf{AKEM}_1.\mathsf{Gen}$
03 $\quad (sk_i^2, pk_i^2) \xleftarrow{\$} \mathsf{AKEM}_2.\mathsf{Gen}$
04 $\quad pk_i \leftarrow (pk_i^1, pk_i^2)$
05 $\mathcal{E}, \mathcal{E}', \Lambda \leftarrow \emptyset$
06 $b \xleftarrow{\$} \{0,1\}$
07 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEnc,ADec,Chall}}(pk_1, \ldots, pk_n)$
08 **return** $[\![b = b']\!]$

Oracle AEnc$(i \in [n], (pk^1, pk^2), m, aad, info)$
09 $(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk_i^2, pk^2)$
10 $c \leftarrow \mathsf{pskAEnc}(sk_i^1, pk^1, K', m, aad, c'||info)$
11 **return** (c, c')

Oracle ADec$((pk^1, pk^2), j \in [n], ((c_1, c_2), c'), aad, info)$
12 **if** $((pk^1, pk^2), (pk_j^1, pk_j^2), (c, c'), aad, info) \in \mathcal{E}$
13 $\quad$ **return** $\perp$
14 $K \leftarrow \mathsf{AuthDecap}_1(pk^1, sk_j^1, c_1)$
15 $K' \leftarrow \mathsf{AuthDecap}_2(pk^2, sk_j^2, c')$
16 $(k, nonce) \leftarrow \mathsf{KS}(K, K', c'||info)$
17 **if** $\exists K : (pk^2, pk_j^2, c', K) \in \mathcal{E}'$ $\quad /\!/ \mathsf{G}_1 - \mathsf{G}_2$
18 $\quad K' \leftarrow K$ $\quad /\!/ \mathsf{G}_1 - \mathsf{G}_2$
19 $\quad (k, nonce) \leftarrow \mathsf{KS}(K, K', c'||info)$ $\quad /\!/ \mathsf{G}_1 - \mathsf{G}_2$
20 $\quad (k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ $\quad /\!/ \mathsf{G}_2$
21 $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
22 **return** m

Oracle Chall$((sk^1, sk^2), j \in [n], m_0, m_1, aad, info)$
23 **if** $|m_0| \neq |m_1|$
24 $\quad$ **return** $\perp$
25 $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}_1(sk^1, pk_i^1)$
26 $(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk^2, pk_j^2)$
27 $K' \xleftarrow{\$} \mathcal{K}_2$ $\quad /\!/ \mathsf{G}_1 - \mathsf{G}_2$
28 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(\mu(sk^2), pk_j^2, c', K')\}$ $\quad /\!/ \mathsf{G}_1 - \mathsf{G}_2$
29 $(k, nonce) \leftarrow \mathsf{KS}(K, K', c'||info)$
30 $(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ $\quad /\!/ \mathsf{G}_2$
31 $c_2 \leftarrow \mathsf{AEAD.Enc}(k, m_b, aad, nonce)$
32 $c \leftarrow (c_1, c_2)$
33 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((\mu(sk^1), \mu(sk^2)), (pk_j^1, pk_j^2), (c, c'), aad, info)\}$
34 **return** (c, c')

is uniformly random due to the changes in the last game, this can be turned into an PRF adversary against KS_2, i.e. KS keyed on the second input. Due to changes in the previous game, we keep consistent outputs with the decryption oracle since we reuse output values which were sampled before. There are at most q_c different keys as well as q_c evaluation queries resulting in

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_c, q_d+q_c)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_2}.$$

Reduction to Game G_2. Game G_2 can be reduced to the CCA security of AEAD. We construct an adversary $\mathcal{D}$ against CCA security of AEAD using adversary $\mathcal{A}$ from G_2. This similar to the the reduction to Game G_4 in the proof of Theorem 1 and the adversary constructed in Listing 25. Note that there is a difference in the number of AEAD keys since we do not have a static *psk* and adversary $\mathcal{D}$ can simulate encryption queries without any query to their own oracles. This results in

$$\Pr[\mathsf{G}_2 \Rightarrow 1] \leq \mathsf{Adv}^{(q_c, q_d)\text{-CCA}}_{\mathcal{D},\mathsf{AEAD}}.$$

Putting everything together, we obtain (2) which concludes the proof. □

C.2 Proof of Theorem 6

Proof. In order to prove Theorem 6, we prove the following two equations.

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq\ & \mathsf{Adv}^{(n+1,q_e,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}_1} \\ &+ \mathsf{Adv}^{(n+1,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}_2,\mathsf{AKEM}_1} \\ &+ \mathsf{Adv}^{(q_e+q_d,q_e+q_d)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_1} \\ &+ \mathsf{Adv}^{(2q_e+q_d+1,q_d+1)\text{-INT-CTXT}}_{\mathcal{D},\mathsf{AEAD}} \\ &+ \frac{q_e(q_e+q_d-1)}{|\mathcal{K}|}\end{aligned} \tag{4}$$

and

$$\begin{aligned}\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq\ & \mathsf{Adv}^{(n,0,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}_2} \\ &+ \mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}_2,\mathsf{AKEM}_2} \\ &+ \mathsf{Adv}^{(q_d,q_d)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_2} \\ &+ \mathsf{Adv}^{(2q_e+q_d+1,q_d+1)\text{-INT-CTXT}}_{\mathcal{D},\mathsf{AEAD}} \\ &+ \frac{q_e(q_e+q_d-1)}{|\mathcal{K}|}.\end{aligned} \tag{5}$$

Inequality (4): We use a similar technique as in the proof of Theorem 5. First, we show

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{APKE}[\mathsf{AKEM}_1,\mathsf{AKEM}_2,\mathsf{KS},\mathsf{AEAD}]} \leq \mathsf{Adv}^{(n+1,q_e,q_d,q_e+q_d,q_e+q_d)\text{-Outsider-Auth}}_{\tilde{\mathcal{A}},\mathsf{pskAPKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]}. \tag{6}$$

Then, we can apply a similar proof as for Theorem 4 with the following change. Due to the special structure of adversary $\tilde{\mathcal{A}}$, we can skip the changes in Game G_6 which introduces the PRF security loss from querying KS in the second input (the *psk* part). This is because none of the conditions which are used for the PRF advantage are met. In detail, for both the encryption and decryption oracle the conditions are $i \notin \Gamma_{\mathtt{pk}} \vee (i,j) \notin \Gamma_{\mathtt{psk}}$ and $i \in \Gamma_{\mathtt{pk}}$ which leads to the condition $(i,j) \notin \Gamma_{\mathtt{psk}}$. However, $(i,j) \notin \Gamma_{\mathtt{psk}}$ never holds since $\tilde{\mathcal{A}}$ queries the RepPSK oracle before querying the encryption or decryption oracle.

This results in

$$\begin{aligned}\mathsf{Adv}^{(n+1,q_e,q_d,q_e+q_d,q_e+q_d)\text{-Outsider-Auth}}_{\tilde{\mathcal{A}},\mathsf{pskAPKE}[\mathsf{AKEM},\mathsf{KS},\mathsf{AEAD}]} \leq\ & \mathsf{Adv}^{(n+1,q_e,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}_1} \\ &+ \mathsf{Adv}^{(n+1,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}_2,\mathsf{AKEM}_1} \\ &+ \mathsf{Adv}^{(q_e+q_d,q_e+q_d)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_1} \\ &+ \mathsf{Adv}^{(2q_e+q_d+1,q_d+1)\text{-INT-CTXT}}_{\mathcal{D},\mathsf{AEAD}} \\ &+ \frac{q_e(q_e+q_d-1)}{|\mathcal{K}|}.\end{aligned}$$

To prove (6), we use an (n,q_e,q_d)-Outsider-Auth adversary against APKE to construct an adversary $\tilde{\mathcal{A}}$ against $(n+1,q_e,q_d,q_e+q_d,q_e+q_d)$-Outsider-Auth security of pskAPKE as

Listing 32: $(n+1, q_e, q_d, q_e+q_d, 0, q_e+q_d)$-Outsider-Auth adversary $\tilde{\mathcal{A}}$ against $\mathsf{pskAPKE}[\mathsf{AKEM}_1, \mathsf{KS}, \mathsf{AEAD}]$ using an (n, q_e, q_d)-Outsider-Auth adversary $\mathcal{A}$ against $\mathsf{APKE}[\mathsf{AKEM}_1, \mathsf{AKEM}_2, \mathsf{KS}, \mathsf{AEAD}]$ for Inequality (6) of the proof of Theorem 6.

$\tilde{\mathcal{A}}^{\text{OEnc},\text{ODec},\text{OChall},\text{ORepPK},\text{ORepSK},\text{ORepPSK}}(pk^1_1, \ldots, pk^1_{n+1})$

for $i \in [n]$
  $(sk^2_i, pk^2_i) \xleftarrow{\$} \mathsf{AKEM}_2.\mathsf{Gen}$
  $pk_i \leftarrow (pk^1_i, pk^2_i)$
$(i^*, j^*, (c^*, c'^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec}}(pk_1, \ldots, pk_n)$
return $(i^*, j^*, c^*, aad^*, info^*)$

Oracle $\text{AEnc}(i \in [n], (pk^1, pk^2), m, aad, info)$

if $\exists j' \in [\ell] : pk^1 = pk^1_{j'}$
  $j \leftarrow j'$
else
  $\text{ORepPK}(n+1, pk^1)$
  $j \leftarrow n+1$
$(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk^2_i, pk^2)$
$\text{ORepPSK}(i, j, K')$
$c \leftarrow \text{OEnc}(i, j, m, aad, info)$
return (c, c')

Oracle $\text{ADec}((pk^1, pk^2), j \in [n], (c, c'), aad, info)$

if $\exists i' \in [\ell] : pk^1 = pk^1_{i'}$
  $i \leftarrow i'$
else
  $\text{ORepPK}(n+1, pk^1)$
  $i \leftarrow n+1$
$K' \leftarrow \mathsf{AuthDecap}_2(pk^2, sk^2_j, c')$
$\text{ORepPSK}(i, j, K')$
$m \leftarrow \text{ODec}(i, j, c, aad, info)$
return m

shown in Listing 32. One can see that $\tilde{\mathcal{A}}$ uses one query to RepPK for each query to Enc and Dec, i.e. $r_{pk} = q_e + q_d$. Additionally, we need to replace the "psk" for each query resulting in $r_{psk} = q_e + q_d$. This fulfills the adversarial query bounds from inequality (6).

Inequality (5): Proving the second inequality (5), we start with the (n, q_e, q_d)-Outsider-Auth game for $\mathsf{APKE}[\mathsf{AKEM}_1, \mathsf{AKEM}_2, \mathsf{KS}, \mathsf{AEAD}]$ denoted by G_0 in Listing 33.

Listing 33: Games $\mathsf{G}_0 - \mathsf{G}_6$ for the proof of Inequality (5) of Theorem 6.

$\mathsf{G}_0 - \mathsf{G}_6$

for $i \in [n]$
  $(sk^1_i, pk^1_i) \xleftarrow{\$} \mathsf{AKEM}_1.\mathsf{Gen}$
  $(sk^2_i, pk^2_i) \xleftarrow{\$} \mathsf{AKEM}_2.\mathsf{Gen}$
  $pk_i \leftarrow (pk^1_i, pk^2_i)$
$\mathcal{E}, \mathcal{E}', \Lambda \leftarrow \emptyset$
$(i^*, j^*, (c^*, c'^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{AEnc},\text{ADec}}(pk_1, \ldots, pk_n)$
return
  $[\![((pk^1_{i^*}, pk^2_{i^*}), (pk^1_{j^*}, pk^2_{j^*}), (c^*, c'^*), aad^*, info^*) \notin \mathcal{E}$
  $\wedge\, (sk^1_{j^*}, sk^2_{j^*}) \neq \perp$
  $\wedge\, \text{ADec}((pk^1_{i^*}, pk^2_{i^*}), (sk^1_{j^*}, sk^2_{j^*}), (c^*, c'^*), aad^*, info^*) \neq \perp]\!]$

Oracle $\text{AEnc}(i \in [n], (pk^1, pk^2), m, aad, info)$

$(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk^2_i, pk^2)$
$K' \xleftarrow{\$} \mathcal{K}_2$ // $\mathsf{G}_1 - \mathsf{G}_6$
$\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk^2_i, pk^2, c', K')\}$ // $\mathsf{G}_1 - \mathsf{G}_6$
$(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}_1(sk^1_i, pk^1)$
$(k, nonce) \leftarrow \mathsf{KS}(K, K', c'||info)$
if $\exists (k', nonce') : (k', nonce', (pk^1_i, pk^2_i), (pk^1, pk^2), K, K', c'||info) \in \Lambda$ // $\mathsf{G}_3 - \mathsf{G}_6$
  abort // $\mathsf{G}_5 - \mathsf{G}_6$
  $(k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_3 - \mathsf{G}_6$
$c_2 \leftarrow \mathsf{AEAD.Enc}(k, m, aad, nonce)$
$c \leftarrow (c_1, c_2)$
$\Lambda \leftarrow \Lambda \cup \{(k, nonce, (pk^1_i, pk^2_i), (pk^1, pk^2), K, K', c'||info)\}$ // $\mathsf{G}_3 - \mathsf{G}_6$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk^1_i, pk^2_i), (pk^1, pk^2), (c, c'), aad, info)\}$
return (c, c')

Oracle $\text{ADec}((pk^1, pk^2), j \in [n], ((c_1, c_2), c'), aad, info)$

$b \leftarrow 0$ // $\mathsf{G}_2 - \mathsf{G}_6$
$K' \leftarrow \mathsf{AuthDecap}(pk^2, sk^2_j, c')$
$K \leftarrow \mathsf{AuthDecap}_1(pk^1, sk^1_j, c_1)$
if $\exists \hat{K} : (pk^2, pk^2_j, c', \hat{K}) \in \mathcal{E}'$ // $\mathsf{G}_1 - \mathsf{G}_6$
  $K' \leftarrow \hat{K}$ // $\mathsf{G}_1 - \mathsf{G}_6$
else if $pk^2 \in \{pk^2_1, \ldots, pk^2_l\} \wedge K' \neq \perp$ // $\mathsf{G}_2 - \mathsf{G}_6$
  $K' \xleftarrow{\$} \mathcal{K}_2$ // $\mathsf{G}_2 - \mathsf{G}_6$
  $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk^2, pk^2_j, c', K')\}$ // $\mathsf{G}_2 - \mathsf{G}_6$
  $b \leftarrow 1$ // $\mathsf{G}_2 - \mathsf{G}_6$
$(k, nonce) \leftarrow \mathsf{KS}(K, K', c'||info)$
if $b = 1$ // $\mathsf{G}_4 - \mathsf{G}_6$
  $(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ // $\mathsf{G}_4 - \mathsf{G}_6$
if $\exists (k', nonce') : (k', nonce', (pk^1, pk^2), (pk^1_j, pk^2_j), K, K', c'||info) \in \Lambda$ // $\mathsf{G}_3 - \mathsf{G}_6$
  $(k, nonce) \leftarrow (k', nonce')$ // $\mathsf{G}_3 - \mathsf{G}_6$
  $m \leftarrow \perp$ // G_6
else // G_6
  $m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // G_6
$m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$ // $\mathsf{G}_0 - \mathsf{G}_4$
if $b = 1$ // G_6
  $m \leftarrow \perp$ // G_6
$\Lambda \leftarrow \Lambda \cup \{(k, nonce, (pk^1, pk^2), (pk^1_j, pk^2_j), K, K', c'||info)\}$ // $\mathsf{G}_3 - \mathsf{G}_6$
return m

Game G_1. We replace the AKEM secret of AKEM_2 by a uniformly random value in the encryption oracle and store the result together with the corresponding parameters in set

$\mathcal{E}'$ so that the decapsulation gives consistent results (Line 25). The advantage can be turned into an adversary against the Outsider-CCA security of AKEM_2 issuing a challenge query for each query to Enc and a decapsulation query for each query to pskDec. Note that there are no encapsulation queries necessary. This results in

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,0,q_d,q_e)\text{-Outsider-CCA}}_{\mathcal{B}_1,\mathsf{AKEM}_2}.$$

Game G_2. Now we replace the AKEM secret of AKEM_2 by a uniformly random value in the decryption oracle if the parameter set was not queried before, i.e. the parameters do not occur in $\mathcal{E}'$ which stores results from previous queries to AEnc and ADec (Line 24), the receiver is honest, i.e. $pk^2 \in \{pk_1^2, \ldots, pk_l^2\}$, and the shared KEM secret is not $\perp$. We additionally add a bit b (Line 29) which indicates if the before mentioned case occurs which will be used in further game modifications. The setup now matches with oracles of an Outsider-Auth adversary against AKEM_2 and such an adversary can simulate the games. This results in the following advantage:

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_e,q_d)\text{-Outsider-Auth}}_{\mathcal{B}_2,\mathsf{AKEM}_2}.$$

Game G_3. We insert a set Λ to log the outputs of KS. If KS was already called with the same parameters, we take the stored value. Since the change is only conceptual due to a deterministic KS, we have

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[\mathsf{G}_3 \Rightarrow 1].$$

Game G_4. In case $b = 1$ (the case we replaced the KEM secret in Game G_2), we replace the output of KS with uniformly random values of the respective domain. The changes in the last game ensure consistent outputs of KS, i.e. queries on AEnc and ADec with the same parameters lead to the same output of KS. Hence, game G_3 and G_4 can only be distinguished by distinguishing the real output of KS from a uniformly random one. This can be turned into an adversary against PRF security of KS_2, i.e. keyed in the second input, since key K' is uniformly chosen in case $b = 1$ due to the changes in Game G_2. There are at most q_d different keys and at most q_d different queries resulting in

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_d,q_d)\text{-PRF}}_{\mathcal{C},\mathsf{KS}_2}.$$

Game G_5. We abort in the encryption oracle if there already exists an entry in Λ with the queried parameters. Since K' was randomly chosen in G_1, the probability of having such an entry is at most $\frac{|\Lambda|}{|\mathcal{K}|}$. Note that Λ is filled with another element at most once per AEnc/ADec query. This yields the following advantage:

$$|\Pr[\mathsf{G}_4 \Rightarrow 1] - \Pr[\mathsf{G}_5 \Rightarrow 1]| \leq \frac{q_e(q_e + q_d - 1)}{|\mathcal{K}|}.$$

Game G_6. In this game, we replace the actual decryption in an honest decryption oracle query, i.e. case $b = 1$, with $\perp$. The same holds if there is already an element in Λ which has the same parameters. Distinguishing the game difference can be turned into an adversary against INT-CTXT security of the underlying AEAD, the construction can be found in Listing 34. There are $q_e + q_d$ different keys and adversary $\mathcal{D}_1$ makes at most q_d queries to their decryption oracle $\mathrm{DEC}_{\mathrm{AEAD}}$. Thus, we obtain

$$|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_6 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e+q_d,q_d)\text{-INT-CTXT}}_{\mathcal{D}_1,\mathsf{AEAD}}.$$

Reduction to Game G_6. We can reduce INT-CTXT security to Game G_6 by constructing an adversary simulating G_6 very similar to the one in Listing 34. The adversary can simulate the decryption oracle since in case $b = 1$ (Line 31 in Listing 33) and when there is a corresponding element in Λ (Line 40 in Listing 33), they can output $\perp$ (or the original encryption if it was produced during the experiment). Otherwise, they can compute the output by their own. For the encryption oracle, the adversary can use their own encryption oracle similar to the adversary from the last game hop. The output of the adversary against game G_6 can then be used to issue a decryption query in the INT-CTXT experiment on either a new instance or a previous instance if parameters $(pk^1_{i^*}, pk^2_{i^*}), (pk^1_{j^*}, pk^2_{j^*}), K^*, K'^*, c'^*||info^*$ match with that instance.. Matching parameters can be identified by computing $K^* \leftarrow \mathsf{AuthDecap}_1(sk_{j^*}, c_1)$, $K'^* \leftarrow \mathsf{AuthDecap}_2(sk_{j^*}, c'^*)$ and comparing $((pk^1_{i^*}, pk^2_{i^*}), (pk^1_{j^*}, pk^2_{j^*}), K^*, K'^*, c'^*||info^*)$ to set Λ similar to Line 12 in Listing 34. If the adversary against G_6 wins, the adversary against the INT-CTXT experiment has a valid ciphertext which does not decrypt to $\perp$ due to the winning condition of G_6. That means they can distinguish between the real or random case since the result of the decryption query must be unequal to $\perp$ in the real case. This results in

$$\Pr[\mathsf{G}_6 \Rightarrow 1] \leq \mathsf{Adv}^{(q_e+1,1)\text{-INT-CTXT}}_{\mathcal{D}_2,\mathsf{AEAD}}.$$

Putting everything together, we obtain the stated bound.

Listing 34: $(q_e + q_d, q_d)$-INT-CTXT adversary $\mathcal{D}_1$ against AEAD using an adversary $\mathcal{A}$ against Game G_6 from the proof of Inequality (5) of Theorem 6.

$\mathcal{D}_1^{\text{ENC}_{\text{AEAD}},\text{DEC}_{\text{AEAD}}}$

01 **for** $i \in [n]$
02 $\quad(sk^1_i, pk^1_i) \xleftarrow{\$} \mathsf{AKEM}_1.\mathsf{Gen}$
03 $\quad(sk^2_i, pk^2_i) \xleftarrow{\$} \mathsf{AKEM}_2.\mathsf{Gen}$
04 $\quad pk_i \leftarrow (pk^1_i, pk^2_i)$
05 $\mathcal{E}, \mathcal{E}', \Lambda \leftarrow \emptyset$
06 $(i^*, j^*, (c^*, c'^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{AENC},\text{ADEC}}(pk_1, \ldots, pk_n)$
07 **return**
$\quad[\![((pk^1_{i^*}, pk^2_{i^*}), (pk^1_{j^*}, pk^2_{j^*}), (c^*, c'^*), aad^*, info^*) \notin \mathcal{E}$
$\quad\wedge (sk^1_{j^*}, sk^2_{j^*}) \neq \perp$
$\quad\wedge \text{ADEC}((pk^1_{i^*}, pk^2_{i^*}), (sk^1_{j^*}, sk^2_{j^*}), (c^*, c'^*), aad^*, info^*) \neq \perp]\!]$

Oracle $\text{AENC}(i \in [n], (pk^1, pk^2), m, aad, info)$

08 $(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk^2_i, pk^2)$
09 $K' \xleftarrow{\$} \mathcal{K}_2$
10 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk^2_i, pk^2, c', K')\}$
11 $(c_1, K) \xleftarrow{\$} \mathsf{AuthEncap}_1(sk^1_i, pk^1)$
12 **if** $\exists \ell^* : (\ell^*, (pk^1_i, pk^2_i), (pk^1, pk^2), K, K', c'||info) \in \Lambda$
13 $\quad$**abort**
14 **else**
15 $\quad\ell \leftarrow \ell + 1$
16 $\quad c_2 \leftarrow \text{ENC}_{\text{AEAD}}(\ell, m, aad, nonce)$
17 $c \leftarrow (c_1, c_2)$
18 $\Lambda \leftarrow \Lambda \cup \{(k, nonce, (pk^1_i, pk^2_i), (pk^1, pk^2), K, K', c'||info)\}$
19 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk^1_i, pk^2_i), (pk^1, pk^2), (c, c'), aad, info)\}$
20 **return** (c, c')

Oracle $\text{ADEC}((pk^1, pk^2), j \in [n], ((c_1, c_2), c'), aad, info)$

21 $b \leftarrow 0$
22 $K' \leftarrow \mathsf{AuthDecap}(pk^2, sk^2_j, c')$
23 $K \leftarrow \mathsf{AuthDecap}_1(pk^1, sk^1_j, c_1)$
24 **if** $\exists \hat{K} : (pk^2, pk^2_j, c', \hat{K}) \in \mathcal{E}'$
25 $\quad K' \leftarrow \hat{K}$
26 **else if** $pk^2 \in \{pk^2_1, \ldots, pk^2_l\} \wedge K' \neq \perp$
27 $\quad K' \xleftarrow{\$} \mathcal{K}_2$
28 $\quad\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk^2, pk^2_j, c', K')\}$
29 $\quad b \leftarrow 1$
30 **if** $\exists \ell^* : (\ell^*, (pk^1, pk^2), (pk^1_j, pk^2_j), K, K', c'||info) \in \Lambda$
31 $\quad m \leftarrow \text{DEC}_{\text{AEAD}}(\ell^*, c_2, aad, nonce)$
32 **else if** $b = 1$
33 $\quad\ell \leftarrow \ell + 1$
34 $\quad m \leftarrow \text{DEC}_{\text{AEAD}}(\ell, c_2, aad, nonce)$
35 $\quad\Lambda \leftarrow \Lambda \cup \{\ell, (pk^1, pk^2), (pk^1_j, pk^2_j), K, K', c'||info)\}$
36 **else**
37 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, K', c'||info)$
38 $\quad m \leftarrow \mathsf{AEAD.Dec}(k, c_2, aad, nonce)$
39 **return** m

□

D Proof for the AKEM Constructions

D.1 Proof of Theorem 7

Proof. We start with the Insider-CCA game for $\mathsf{AKEM}^{\mathsf{EtStH}}[\mathsf{KEM}, \mathsf{SIG}, \mathsf{H}]$ as Game G_0 in Listing 35:

$$\Pr[(n, q_e, q_d, q_c, r_{sk})\text{-Insider-CCA}(\mathcal{A}) \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1].$$

Listing 35: Games $\mathsf{G}_0 - \mathsf{G}_3$ for the proof of Theorem 7.

$\underline{\mathsf{G}_0 - \mathsf{G}_3}$
01 **for** $i \in [n]$
02 $\quad((sk_i, sigk_i), (pk_i, vk_i)) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E}, \mathcal{E}' \Gamma_{\mathrm{pk}} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap,ADecap,Chall,RepSK}}((pk_1, vk_1), \ldots, (pk_n, vk_n))$
06 **return** $[\![b = b']\!]$

$\underline{\text{Oracle AEncap}(i \in [n], (pk, vk))}$
07 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk)$
08 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk||vk)$
09 $K \leftarrow \mathsf{H}(K', \sigma||pk_i||vk_i||pk||vk)$
10 **return** $((c, \sigma), K)$

$\underline{\text{Oracle ADecap}((pk, vk), j \in [n], (c, \sigma))}$
11 **if** $\exists K : ((pk, vk), (pk_j, vk_j), (c, \sigma), K) \in \mathcal{E}$
12 $\quad$**return** K
13 **if** $\mathsf{Vfy}(vk, c||pk||pk_j||vk_j, \sigma) \neq 1$
14 $\quad$**return** $\perp$
15 $K' \leftarrow \mathsf{Decaps}(sk_j, c)$
16 **if** $\exists \hat{K} : (pk_j, c, \hat{K}) \in \mathcal{E}'$ $\quad /\!/ \mathsf{G}_2 - \mathsf{G}_3$
17 $\quad K' \leftarrow \hat{K}$ $\quad /\!/ \mathsf{G}_2 - \mathsf{G}_3$
18 $\quad K \xleftarrow{\$} \mathcal{K}$ $\quad /\!/ \mathsf{G}_3$
19 **else** $\quad /\!/ \mathsf{G}_3$
20 $\quad K \leftarrow \mathsf{H}(K', \sigma||pk||vk||pk_j||vk_j)$ $\quad /\!/ \mathsf{G}_3$
21 $K \leftarrow \mathsf{H}(K', \sigma||pk||vk||pk_j||vk_j)$ $\quad /\!/ \mathsf{G}_0 - \mathsf{G}_2$
22 **return** K

$\underline{\text{Oracle Chall}(i \in [n], j \in [n])}$
23 **if** $j \in \Gamma_{\mathrm{pk}}$
24 $\quad$**return** $\perp$
25 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk_j)$
26 $K' \xleftarrow{\$} \mathcal{K}'$ $\quad /\!/ \mathsf{G}_2 - \mathsf{G}_3$
27 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk_j, c, K')\}$ $\quad /\!/ \mathsf{G}_2 - \mathsf{G}_3$
28 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk_j||vk_j)$
29 $K \leftarrow \mathsf{H}(K', \sigma||pk_i||vk_i||pk_j||vk_j)$
30 $K \xleftarrow{\$} \mathcal{K}$ $\quad /\!/ \mathsf{G}_3$
31 **if** $b = 1$
32 $\quad K \xleftarrow{\$} \mathcal{K}$
33 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{((pk_i, vk_i), (pk_j, vk_j), (c, \sigma), K)\}$ $\quad /\!/ \mathsf{G}_0$
34 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk_i, vk_i), (pk_j, vk_j), (c, \sigma), K)\}$ $\quad /\!/ \mathsf{G}_1 - \mathsf{G}_3$
35 **return** $((c, \sigma), K)$

$\underline{\text{Oracle RepSK}(i \in [n], (sk, sigk))}$
36 $(sk_i, sigk_i) \leftarrow (sk, sigk)$
37 $(pk_i, vk_i) \leftarrow (\mu(sk), \mu'(sigk))$
38 $\Gamma_{\mathrm{pk}} \leftarrow \Gamma_{\mathrm{pk}} \cup \{i\}$

Game G_1. In Game G_1, we change the challenge oracle such that query results are not only stored in set $\mathcal{E}$ in the case of $b = 1$ but also for $b = 0$. Due to the correctness of the decapsulation algorithm, this change does not effect the outputs of any query and the winning probabilities do not change. These changes are only conceptual resulting in

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1].$$

Game G_2. In Game G_2, the KEM secret in the challenge oracle is replaced by uniformly random value from the KEM keyspace $\mathcal{K}'$ and the sampled values are stored. The decapsulation oracle is also modified to provide consistent outputs. The advantage of this change corresponds to a CCA adversary against KEM depicted in Listing 36. Note that a KEM adversary can simulate the complete game including the RepSK oracle and its

effects since the signing keys are sampled by the simulator itself, the ADECAP can be computed by the simulator in case the secret key was replaced and the challenge oracle always returns $\perp$ in case REPSK was called on the receiver's key. In particular, we have

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_d,q_c)\text{-}\mathsf{CCA}}_{\mathcal{B},\mathsf{KEM}}.$$

Listing 36: Adversary $\mathcal{B}$ against CCA security of KEM having access to oracles ODECAP and OCHALL using adversary $\mathcal{A}$ from Games $\mathsf{G}_1/\mathsf{G}_2$ of the proof of Theorem 7.

$\mathcal{B}^{\text{ODECAP},\text{OCHALL}}(pk_1, \dots pk_n)$
01 **for** $i \in [n]$
02 $(sigk_i, vk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E}, \Gamma_{\mathsf{pk}} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AENCAP},\text{ADECAP},\text{CHALL},\text{REPSK}}((pk_1, vk_1), \dots, (pk_n, vk_n))$
06 **return** $[\![b = b']\!]$

Oracle AENCAP$(i \in [n], (pk, vk))$
07 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk)$
08 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk||vk)$
09 $K \leftarrow \mathsf{H}(K', \sigma||pk_i||vk_i||pk||vk)$
10 **return** $((c, \sigma), K)$

Oracle ADECAP$((pk, vk), j \in [n], (c, \sigma))$
11 **if** $\exists K : ((pk, vk), (pk_j, vk_j), (c, \sigma), K) \in \mathcal{E}$
12 **return** K
13 **if** $\mathsf{Vfy}(vk, c||pk||pk_j||vk_j, \sigma) \neq 1$
14 **return** $\perp$
15 **if** $j \in \Gamma_{\mathsf{pk}}$ //check for corruptions
16 $K' \leftarrow \mathsf{Decaps}(sk_j, c)$
17 **else**
18 $K' \leftarrow \text{ODECAP}(j, c)$ //use decapsulation oracle
19 $K \leftarrow \mathsf{H}(K', \sigma||pk||vk||pk_j||vk_j)$
20 **return** K

Oracle CHALL$(i \in [n], j \in [n])$
21 **if** $j \in \Gamma_{\mathsf{pk}}$
22 **return** $\perp$
23 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk_j)$
24 $K' \leftarrow \text{OCHALL}(j)$ //embed challenge
25 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk_j||vk_j)$
26 $K \leftarrow \mathsf{H}(K', \sigma||pk_i||vk_i||pk_j||vk_j)$
27 **if** $b = 1$
28 $K \xleftarrow{\$} \mathcal{K}$
29 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk_i, vk_i), (pk_j, vk_j), (c, \sigma), K)\}$
30 **return** $((c, \sigma), K)$

Oracle REPSK$(i \in [n], (sk, sigk))$
31 $(sk_i, sigk_i) \leftarrow (sk, sigk)$
32 $(pk_i, vk_i) \leftarrow (\mu(sk), \mu'(sigk))$
33 $\Gamma_{\mathsf{pk}} \leftarrow \Gamma_{\mathsf{pk}} \cup \{i\}$

Game G_3. In G_3, the game is modified such that the challenge oracle always returns a uniformly random key K (Line 30), i.e. the output of H is uniformly random. Additionally, the decapsulation oracle is changed in a similar way by replacing the output of H by uniformly random value in case there is an element in set $\mathcal{E}'$ (Line 16). The difference can be turned into an PRF adversary $\mathcal{C}$ against H such that:

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_c,q_d+q_c)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{H}}.$$

To proof the claim, we construct an adversary in Listing 37. Adversary $\mathcal{C}$ can keep track of the key they queried by using index ℓ which is increased for every challenge query corresponding to a new key of the PRF game. This key can be reused if there is a decapsulation query which can be checked by modifying set $\mathcal{E}'$ to not contain the actual key K' but index ℓ of the key (Line 26 and Line 16). Adversary $\mathcal{C}$ needs at most q_c many different keys and at most one evaluation query for each decapsulation as well as challenge

query: $q_d + q_c$. Note that in the decapsulation oracle of G_3, a new key K is chosen no matter if the same value was queried before in the challenge oracle, i.e. same key K' and same input $\sigma||pk||vk||pk_j||vk_j$. However, if this was the case, this would lead to a return in Line 12 of the decapsulation oracle.

Game G_3 is now independent of the challenge bit b since a uniformly random value is output in both cases:

$$\Pr[\mathsf{G}_3 \Rightarrow 1] = \frac{1}{2}.$$

The stated bound can be obtained by combining the game differences.

Listing 37: Adversary $\mathcal{C}$ against PRF security of H having access to oracle Eval using adversary $\mathcal{A}$ from Games $\mathsf{G}_2/\mathsf{G}_3$ of the proof of Theorem 7.

$\underline{\mathcal{C}^{\text{Eval}}}$
01 **for** $i \in [n]$
02 $\quad((sk_i, sigk_i), (pk_i, vk_i)) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E}, \mathcal{E}', \Gamma_{\mathsf{pk}} \leftarrow \emptyset$
04 $\ell \leftarrow 0$
05 $b \xleftarrow{\$} \{0,1\}$
06 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap,ADecap,Chall,RepSK}}((pk_1, vk_1), \ldots, (pk_n, vk_n))$
07 **return** $[\![b = b']\!]$

Oracle AEncap$(i \in [n], (pk, vk))$
08 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk)$
09 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk||vk)$
10 $K \leftarrow \mathsf{H}(K', \sigma||pk_i)$
11 **return** $((c, \sigma), K)$

Oracle ADecap$((pk, vk), j \in [n], (c, \sigma))$
12 **if** $\exists K : ((pk, vk), (pk_j, vk_j), (c, \sigma), K) \in \mathcal{E}$
13 $\quad$**return** K
14 **if** $\mathsf{Vfy}(vk, c||pk||pk_j||vk_j, \sigma) \neq 1$
15 $\quad$**return** $\perp$
16 **if** $\exists \hat{\ell} : (pk_j, c, \hat{\ell}) \in \mathcal{E}'$
17 $\quad K \leftarrow \text{Eval}(\hat{\ell}, \sigma||pk_i||vk_i||pk_j||vk_j)$ //eval query
18 **else**
19 $\quad K' \leftarrow \mathsf{Decaps}(sk_j, c)$
20 $\quad K \leftarrow \mathsf{H}(K', \sigma||pk_i||vk_i||pk_j||vk_j)$
21 **return** K

Oracle Chall$(i \in [n], j \in [n])$
22 **if** $j \in \Gamma_{\mathsf{pk}}$
23 $\quad$**return** $\perp$
24 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk_j)$
25 $\ell \leftarrow \ell + 1$ //new key
26 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(pk_j, c, \ell)\}$
27 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk_j||vk_j)$
28 $K \leftarrow \text{Eval}(\ell, \sigma||pk_i||vk_i||pk_j||vk_j)$ //eval query
29 **if** $b = 1$
30 $\quad K \xleftarrow{\$} \mathcal{K}$
31 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk_i, vk_i), (pk_j, vk_j), (c, \sigma), K)\}$
32 **return** $((c, \sigma), K)$

Oracle RepSK$(i \in [n], (sk, sigk))$
33 $(sk_i, sigk_i) \leftarrow (sk, sigk)$
34 $(pk_i, vk_i) \leftarrow (\mu(sk), \mu'(sigk))$
35 $\Gamma_{\mathsf{pk}} \leftarrow \Gamma_{\mathsf{pk}} \cup \{i\}$

□

D.2 Proof of Theorem 8

Proof. We start with the Outsider-Auth game for AKEM as Game G_0 in Listing 38:

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d)\text{-Outsider-Auth} \Rightarrow 1].$$

Listing 38: Games $\mathsf{G}_0 - \mathsf{G}_1$ for the proof of Theorem 8.

$\underline{\mathsf{G}_0 - \mathsf{G}_1}$
01 **for** $i \in [n]$
02 $\quad((sk_i, sigk_i), (pk_i, vk_i)) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap}}((pk_1, vk_1), \ldots, (pk_n, vk_n))$
06 **return** $[\![b = b']\!]$

Oracle AEncap$(i \in [n], (pk, vk))$
07 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk)$
08 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_i, c||pk_i||pk||vk)$
09 $K \leftarrow \mathsf{H}(K', \sigma, pk_i)$
10 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk_i, vk_i), (pk, vk), (c, \sigma), K)\}$
11 **return** $((c, \sigma), K)$

Oracle ADecap$((pk, vk), j \in [n], (c, \sigma))$
12 **if** $\exists K : ((pk, vk), (pk_j, vk_j), (c, \sigma), K) \in \mathcal{E}$
13 $\quad$ **then return** K
14 **if** $\mathsf{Vfy}(vk, c||pk||pk_j||vk_j, \sigma) \neq 1$
15 $\quad K \leftarrow \bot$
16 **else**
17 $\quad K' \leftarrow \mathsf{Decaps}(sk_j, c)$
18 $\quad K \leftarrow \mathsf{H}(K', \sigma, pk)$
19 $\quad$ **if** $(pk, vk) \in \{(pk_1, vk_1), \ldots, (pk_n, vk_n)\}$ $/\!/\mathsf{G}_1$
20 $\quad\quad$ **abort** $/\!/\mathsf{G}_1$
21 **if** $b = 1 \wedge (pk, vk) \in \{(pk_1, vk_1), \ldots, (pk_n, vk_n)\} \wedge K \neq \bot$
22 $\quad K \xleftarrow{\$} \mathcal{K}$
23 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{((pk, vk), (pk_j, vk_j), (c, \sigma), K)\}$
24 **return** K

Game G_1. In G_1, the decapsulation oracle is modified such that the game aborts in case of a valid signature and honest sender keys. The difference can be turned into an SUF-CMA adversary $\mathcal{B}$ against SIG:

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n,q_e)\text{-SUF-CMA}}_{\mathcal{B},\mathsf{SIG}}$$

The adversary is depicted in Listing 39. Analyzing the winning condition of adversary $\mathcal{B}$, we can observe that signature σ is valid for the output message due to the check in Line 14. Moreover, the returned tuple $(i, c||pk||pk_j||vk_j, \sigma)$ cannot be in the set of queried signature Q (Game Listing 8): if it was in the set, there was a query to OSign on inputs $i, c||pk||pk_j||vk_j$ returning σ. This means that there is also an entry $((pk_i, vk_i), (pk, vk), (c, \sigma), \cdot)$ in $\mathcal{E}$ due to Line 10 leading to a return in Line 13. Hence, if adversary $\mathcal{B}$ returns in Line 20, they win the SUF-CMA game. Thus, distinguishing the games can be turned into an adversary against SUF-CMA for n users issuing at most q_e signature queries.

Game G_1 is now independent of challenge bit b since in the case of honest sender keys the game aborts and otherwise the condition in Line 21 is not true

$$\Pr[\mathsf{G}_1 \Rightarrow 1] = \frac{1}{2}.$$

Putting everything together yields the stated bound.

□

D.3 Proof of Theorem 9

Proof. We start with the IND-CCA experiment against AKEMs for the construction $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ which we call Game G_0.

In Game G_1, we add the output of a challenge query to set $\mathcal{E}$ in any case to provide consistent outputs later. This change does not influence the winning probability. Further, we replace the output of the second shared key invocation in the challenge oracle by a uniformly random output of the respective domain. We show that the difference of

Listing 39: Adversary $\mathcal{B}$ against SUF-CMA with access to oracle OSign using adversary $\mathcal{A}$ against $\mathsf{G}_0/\mathsf{G}_1$ from the proof of Theorem 8.

$\mathcal{B}^{\text{OSign}}(vk_1, \ldots, vk_n)$
01 **for** $i \in [n]$
02 $\quad(sk_i, pk_i) \xleftarrow{\$} \mathsf{KEM.Gen}$
03 $\mathcal{E}, \mathcal{E}' \leftarrow \emptyset$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap}}((pk_1, vk_1), \ldots, (pk_n, vk_n))$
06 **return** $\bot$

Oracle AEncap$(i \in [n], (pk, vk))$
07 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk)$
08 $\sigma \xleftarrow{\$} \text{OSign}(i, c||pk_i||pk||vk)$ //signing oracle
09 $K \leftarrow \mathsf{H}(K', \sigma, pk_i)$
10 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk_i, vk_i), (pk, vk), (c, \sigma), K)\}$
11 **return** $((c, \sigma), K)$

Oracle ADecap$((pk, vk), j \in [n], (c, \sigma))$
12 **if** $\exists K : ((pk, vk), (pk_j, vk_j), (c, \sigma), K) \in \mathcal{E}$
13 $\quad$**return** K
14 **if** $\mathsf{Vfy}(vk, c||pk||pk_j||vk_j, \sigma) \neq 1$
15 $\quad K \leftarrow \bot$
16 **else**
17 $\quad K' \leftarrow \mathsf{Decaps}(sk_j, c)$
18 $\quad K \leftarrow \mathsf{H}(K', \sigma, pk)$
19 $\quad$**if** $\exists i : (pk, vk) = (pk_i, vk_i)$
20 $\quad\quad$**return** $(i, c||pk||pk_j||vk_j, \sigma)$ //return forgery
21 **if** $b = 1 \wedge (pk, vk) \in \{(pk_1, vk_1), \ldots, (pk_n, vk_n)\} \wedge K \neq \bot$
22 $\quad K \xleftarrow{\$} \mathcal{K}$
23 $\mathcal{E} \leftarrow \mathcal{E} \cup \{((pk, vk), (pk_j, vk_j), (c, \sigma), K)\}$
24 **return** K

the second step is indistinguishable assuming active security of NIKE. This is done by constructing an Active adversary against NIKE simulating game G_0 (in the adversary's own real case) or G_1 (in the adversary's own random game). We depict the construction of adversary $\mathcal{B}$ against Active security of NIKE using adversary $\mathcal{A}$ from game $\mathsf{G}_0/\mathsf{G}_1$ against Insider-CCA security of $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ in Listing 41. This leads to

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n+q_c, q_e+2q_d, 0, q_e+q_d+q_c, q_e+2q_d, q_c)\text{-Active}}_{\mathcal{B},\mathsf{NIKE}}.$$

In Game G_2, we replace the output of H in the challenge oracle by a uniformly random output of the respective domain. This change can be upper bounded by the advantage of a PRF adversary against H_2 since the second input, K_2, is uniformly random. We obtain

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_c, q_c)\text{-PRF}}_{\mathcal{C},\mathsf{H}_2}.$$

The resulting game has winning probability $\frac{1}{2}$ since b is hidden from the adversary. Combining the game differences yields the stated bound. □

D.4 Proof of Theorem 10

Proof. The proof uses a similar strategy as the proof for Theorem 9. We start with the Outsider-Auth game for $\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE}, \mathsf{H}]$ in Listing 42 as G_0:

$$\left|\Pr[\mathsf{G}_0 \Rightarrow 1] - \frac{1}{2}\right| = \mathsf{Adv}^{(n, q_e, q_d)\text{-Outsider-Auth}}_{\mathcal{A},\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE},\mathsf{H}]}.$$

Game G_1. In Game G_1, we modify the encapsulation oracle by replacing the first NIKE shared key K_1 by a uniformly random value from the key space (Line 21) if the receiver key is honest (Line 20) and save the result together with the public keys in set Λ. In case there already exists a matching element in Λ, we use that one instead (Line 13). The same modifications are applied to the decapsulation oracle with the difference that the replacement with a uniformly random value is only done if the sender's key pk is honest

Listing 40: Games $\mathsf{G}_0 - \mathsf{G}_2$ for the proof of Theorem 9

$\underline{\mathsf{G}_0 - \mathsf{G}_2}$
01 **for** $i \in [n]$
02 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathcal{E}, \Gamma_{\mathsf{pk}} \leftarrow \emptyset$
04 $b \xleftarrow{\$} \{0,1\}$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap,ADecap,Chall,RepSK}}(pk_1, \ldots, pk_n)$
06 **return** $[\![b = b']\!]$

Oracle AEncap$(i \in [n], pk)$
07 $(sk^*, pk^*) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
08 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_i, pk)$
09 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk^*, pk)$
10 $K \leftarrow \mathsf{H}(K_1, K_2, pk_i || pk || pk^*)$
11 **return** (pk^*, K)

Oracle ADecap$(pk, j \in [n], c)$
12 **if** $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
13 $\quad$ **return** K
14 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, pk)$
15 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, c)$
16 $K \leftarrow \mathsf{H}(K_1, K_2, pk || pk_j || c)$
17 **return** K

Oracle Chall$(i \in [n], j \in [n])$
18 **if** $j \in \Gamma_{\mathsf{pk}}$
19 $\quad$ **return** $\perp$
20 $(sk^*, pk^*) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
21 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_i, pk_j)$
22 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk^*, pk_j)$
23 $K_2 \xleftarrow{\$} \mathcal{K}_1$ $\quad /\!\!/ \mathsf{G}_1 - \mathsf{G}_2$
24 $c \leftarrow pk^*$
25 $K \leftarrow \mathsf{H}(K_1, K_2, pk_i || pk_j || pk^*)$
26 $K \xleftarrow{\$} \mathcal{K}$ $\quad /\!\!/ \mathsf{G}_2$
27 **if** $b = 1$
28 $\quad K \xleftarrow{\$} \mathcal{K}$
29 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, c, K)\}$
30 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk_j, c, K)\}$ $\quad /\!\!/ \mathsf{G}_1 - \mathsf{G}_2$
31 **return** (c, K)

Oracle RepSK$(i \in [n], sk)$
32 $(sk_i, pk_i) \leftarrow (sk, \mu(sk))$
33 $\Gamma_{\mathsf{pk}} \leftarrow \Gamma_{\mathsf{pk}} \cup \{i\}$

and the original AKEM key K is not $\perp$ (Line 41). The game difference is at most the advantage of an adversary $\mathcal{B}$ against Active security of NIKE:

$$|\Pr[\mathsf{G}_0 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{(n, q_e+q_d, 0, 0, q_e+q_d, q_e+q_d)\text{-Active}}_{\mathcal{B}, \mathsf{NIKE}}.$$

Adversary $\mathcal{B}$ is formally constructed in Listing 43. In the real case of the Active game of adversary $\mathcal{B}$, they perfectly simulate G_0 since every call to the NIKE oracles returns the actual shared key K_1. In the random case, Test returns a uniformly random output when queried on i, j. The test oracle Test is only called for honest receiver (sender) keys in the encapsulation (decapsulation) oracle thus simulating Game G_1 for adversary $\mathcal{A}$. Note that Test returns the same value when queried multiple times on the same indices i, j which corresponds to the check for elements in set Λ in Game G_1.

Game G_2. In Game G_2, a set $\mathcal{H}$ is introduced which stores the output and all the inputs of queries to H. Additionally, flag BAD is set to **true** and the game aborts if H is queried on the same inputs twice, i.e. there already exists a matching element in $\mathcal{H}$, in the case of a matching element in Λ. The game difference is

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq q_e(q_e + q_d) P_{\mathsf{NIKE}}.$$

The **abort** case in the decapsulation never occurs. If there was a query on the same inputs to H in oracle ADecap, this includes the last input $pk || pk_j, || c$ and therefore there must also be a matching element in $\mathcal{E}$, the oracle would have returned in Line 29 and the

Listing 41: Adversary $\mathcal{B}$ against Active security of NIKE using an adversary $\mathcal{A}$ against $\mathsf{G}_0/\mathsf{G}_1$ for the proof of Theorem 9.

```
B^{RegHonest,RegCorrupt,Extract,HonestRev,CorruptRev,Test}
01 for i ∈ [n]
02   sk_i ← ⊥
03   pk_i ←$ RegHonest(i)
04 ℓ ← n
05 E, Γ_pk ← ∅
06 b ←$ A^{AEncap,ADecap,Chall,RepSK}(pk_1, ..., pk_n)
07 return b

AEncap(i ∈ [n], pk)
08 if sk_i = ⊥                                  //honest sender
09   if ∃j : pk_j = pk                          //honest receiver
10     K_1 ← HonestRev(i, j)
11   else                                       //corrupted receiver
12     ℓ ← ℓ + 1
13     RegCorrupt(ℓ, pk)
14     K_1 ← CorruptRev(i, ℓ)
15 else                                         //corrupted sender
16   K_1 ← NIKE.SharedKey(sk_i, pk)
17 (sk*, pk*) ←$ NIKE.KeyGen
18 K_2 ← NIKE.SharedKey(sk*, pk)
19 K ← H(K_1, K_2, pk_i||pk||pk*)
20 return (pk*, K)

RepSK(i ∈ [n], sk)
21 (sk_i, pk_i) ← (sk, μ(sk))
22 Γ_pk ← Γ_pk ∪ {i}

ADecap(pk, j ∈ [n], c)
23 if ∃K : (pk, pk_j, c, K) ∈ E
24   return K
25 if sk_j = ⊥                                  //honest receiver
26   if ∃i : pk_i = pk                          //honest sender
27     K_1 ← HonestRev(i, j)
28   else                                       //corrupted sender
29     ℓ ← ℓ + 1
30     RegCorrupt(ℓ, pk)
31     K_1 ← CorruptRev(ℓ, j)
32   ℓ ← ℓ + 1
33   RegCorrupt(ℓ, c)
34   K_2 ← CorruptRev(ℓ, j)
35 else                                         //corrupted receiver
36   K_1 ← NIKE.SharedKey(sk_j, pk)
37   K_2 ← NIKE.SharedKey(sk_j, c)
38 K ← H(K_1, K_2, pk||pk_j||c)
39 return K

Chall(i ∈ [n], j ∈ [n])
40 if j ∈ Γ_pk
41   return ⊥
42 ℓ ← ℓ + 1
43 sk* ← ⊥
44 pk* ←$ RegHonest(ℓ)
45 if sk_i = ⊥                                  //honest sender
46   K_1 ← HonestRev(i, j)
47 else                                         //corrupted sender
48   K_1 ← NIKE.SharedKey(sk_i, pk_j)
49 K_2 ← Test(ℓ, j)
50 K ← H(K_1, K_2, pk_i||pk_j||pk*)
51 E ← E ∪ {(pk_i, pk_j, pk*, K)}
52 return (pk*, K)
```

Listing 42: Games $\mathsf{G}_0 - \mathsf{G}_3$ for the proof of Theorem 10.

$\underline{\mathsf{G}_0 - \mathsf{G}_3}$
01 **for** $i \in [n]$
02 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\mathsf{BAD} \leftarrow \mathbf{false}$
04 $\mathcal{E}, \Lambda, \mathcal{H} \leftarrow \emptyset$
05 $b \xleftarrow{\$} \{0,1\}$
06 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap,ADecap}}(pk_1, \ldots, pk_n)$
07 **return** $[\![b = b']\!]$

$\underline{\text{Oracle AEncap}(i \in [n], pk \in \mathcal{PK}')}$
08 $(sk^*, pk^*) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
09 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_i, pk)$
10 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk^*, pk)$
11 $K \leftarrow \mathsf{H}(K_1, K_2, pk_i || pk || pk^*)$
12 **if** $\exists K_1' : (K_1', \{pk_i, pk\}) \in \Lambda$ // $\mathsf{G}_1 - \mathsf{G}_3$
13 $\quad K_1 \leftarrow K_1'$ // $\mathsf{G}_1 - \mathsf{G}_3$
14 $\quad K \leftarrow \mathsf{H}(K_1, K_2, pk_i || pk || pk^*)$ // $\mathsf{G}_1 - \mathsf{G}_3$
15 $\quad$ **if** $\exists h : (h, K_1, K_2, pk_i || pk || pk^*) \in \mathcal{H}$ // $\mathsf{G}_2 - \mathsf{G}_3$
16 $\quad\quad \mathsf{BAD} \leftarrow \mathbf{true}; \mathbf{abort}$ // $\mathsf{G}_2 - \mathsf{G}_3$
17 $\quad$ **else** // $\mathsf{G}_2 - \mathsf{G}_3$
18 $\quad\quad K \xleftarrow{\$} \mathcal{K}$ // G_3
19 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, K_1, K_2, pk_i || pk || pk^*)\}$ // $\mathsf{G}_2 - \mathsf{G}_3$
20 **else if** $pk \in \{pk_1, \ldots, pk_n\}$ // $\mathsf{G}_1 - \mathsf{G}_3$
21 $\quad K_1 \xleftarrow{\$} \mathcal{K}_{\mathsf{NIKE}}$ // $\mathsf{G}_1 - \mathsf{G}_3$
22 $\quad \Lambda \leftarrow \Lambda \cup \{(K_1, \{pk_i, pk\})\}$ // $\mathsf{G}_1 - \mathsf{G}_3$
23 $\quad K \leftarrow \mathsf{H}(K_1, K_2, pk_i || pk || pk^*)$ // $\mathsf{G}_1 - \mathsf{G}_3$
24 $\quad K \xleftarrow{\$} \mathcal{K}$ // G_3
25 $\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, K_1, K_2, pk_i || pk || pk^*)\}$ // $\mathsf{G}_2 - \mathsf{G}_3$
26 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, pk^*, K)\}$
27 **return** (pk^*, K)

$\underline{\text{Oracle ADecap}(pk \in \mathcal{PK}', j \in [n], c)}$
28 **if** $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
29 $\quad$ **return** K
30 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, pk)$
31 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, c)$
32 $K \leftarrow \mathsf{H}(K_1, K_2, pk || pk_j || c)$
33 **if** $\exists K_1' : (K_1', \{pk, pk_j\}) \in \Lambda$ // $\mathsf{G}_1 - \mathsf{G}_3$
34 $\quad K_1 \leftarrow K_1'$ // $\mathsf{G}_1 - \mathsf{G}_3$
35 $\quad K \leftarrow \mathsf{H}(K_1, K_2, pk || pk_j || c)$ // $\mathsf{G}_1 - \mathsf{G}_3$
36 $\quad$ **if** $\exists h : (h, K_1, K_2, pk || pk_j || c) \in \mathcal{H}$ // $\mathsf{G}_2 - \mathsf{G}_3$
37 $\quad\quad \mathsf{BAD} \leftarrow \mathbf{true}; \mathbf{abort}$ // $\mathsf{G}_2 - \mathsf{G}_3$
38 $\quad$ **else** // $\mathsf{G}_2 - \mathsf{G}_3$
39 $\quad\quad K \xleftarrow{\$} \mathcal{K}$ // G_3
40 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, K_1, K_2, pk || pk_j || c)\}$ // $\mathsf{G}_2 - \mathsf{G}_3$
41 **else if** $pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \bot$ // $\mathsf{G}_1 - \mathsf{G}_3$
42 $\quad K_1 \xleftarrow{\$} \mathcal{K}_{\mathsf{NIKE}}$ // $\mathsf{G}_1 - \mathsf{G}_3$
43 $\quad \Lambda \leftarrow \Lambda \cup \{(K_1, \{pk, pk_j\})\}$ // $\mathsf{G}_1 - \mathsf{G}_3$
44 $\quad K \leftarrow \mathsf{H}(K_1, K_2, pk || pk_j || c)$ // $\mathsf{G}_1 - \mathsf{G}_3$
45 $\quad K \xleftarrow{\$} \mathcal{K}$ // G_3
46 $\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, K_1, K_2, pk || pk_j || c)\}$ // $\mathsf{G}_2 - \mathsf{G}_3$
47 **if** $b = 1 \wedge pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \bot$
48 $\quad K \xleftarrow{\$} \mathcal{K}$
49 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$
50 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$ // $\mathsf{G}_2 - \mathsf{G}_3$
51 **return** K

Listing 43: Adversary $\mathcal{B}$ against Active security of NIKE using an adversary $\mathcal{A}$ against $\mathsf{G}_0/\mathsf{G}_1$ for the proof of Theorem 10.

$\underline{\mathcal{B}^{\text{RegHonest,RegCorrupt,Extract,HonestRev,CorruptRev,Test}}}$
01 **for** $i \in [n]$
02 $\quad pk_i \xleftarrow{\$} \text{RegHonest}(i)$
03 $\ell \leftarrow n$
04 $\mathcal{E} \leftarrow \emptyset$
05 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap,ADecap}}(pk_1, \ldots, pk_n)$
06 **return** b'

$\underline{\text{Oracle AEncap}(i \in [n], pk \in \mathcal{PK}')}$
07 $(sk^*, pk^*) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
08 **if** $\exists j : pk_j = pk$ //honest receiver
09 $\quad K_1 \leftarrow \text{Test}(i, j)$
10 **else** //corrupted receiver
11 $\quad \ell \leftarrow \ell + 1$
12 $\quad \text{RegCorrupt}(\ell, pk)$
13 $\quad K_1 \leftarrow \text{CorruptRev}(i, \ell)$
14 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk^*, pk)$
15 $K \leftarrow \mathsf{H}(K_1, K_2, pk_i || pk || pk^*)$
16 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, pk^*, K)\}$
17 **return** (pk^*, K)

$\underline{\text{Oracle ADecap}(pk \in \mathcal{PK}', j \in [n], c)}$
18 **if** $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
19 $\quad$ **return** K
20 $K \leftarrow \mathsf{AuthDecap}(pk, sk_j, c)$
21 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, pk)$
22 **if** $\exists i : pk_i = pk$ //honest sender
23 $\quad K_1 \leftarrow \text{Test}(i, j)$
24 **else** //corrupted sender
25 $\quad \ell \leftarrow \ell + 1$
26 $\quad \text{RegCorrupt}(\ell, pk)$
27 $\quad K_1 \leftarrow \text{CorruptRev}(j, \ell)$
28 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, c)$
29 $K \leftarrow \mathsf{H}(K_1, K_2, pk || pk_j || c)$
30 **if** $b = 1 \wedge pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \bot$
31 $\quad K \xleftarrow{\$} \mathcal{K}$
32 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$
33 **return** K

abort is not reached. For the **abort** in the encapsulation oracle to trigger, there must be an element in $\mathcal{H}$ for which especially pk^* matches. Since pk^* is randomly generated by $\mathsf{NIKE.KeyGen}$, there must be a collision in two public keys. There are at most $q_e + q_d$ elements in set $\mathcal{H}$. Hence, for a single query to AEncap the probability can be upper bounded by the entropy of the NIKE public key and in total we obtain

$$\Pr[\mathsf{BAD} = \mathbf{true}] \leq (q_e + q_d) P_{\mathsf{NIKE}}.$$

Game G_3. In G_3, the output of H is replaced by a uniformly random value and the result is stored in $\mathcal{H}$. The difference can be turned into an adversary $\mathcal{C}$ against PRF security of H_1, i.e. H keyed on the first component:

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \mathsf{Adv}^{(q_e+q_d, q_e+q_d)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{H}_1}.$$

We depict adversary $\mathcal{C}$ in Listing 44. Note that adversary $\mathcal{C}$ needs potentially a new PRF key for each call to AEncap and ADecap and the same amount of queries to the PRF oracle Eval.

Listing 44: Adversary $\mathcal{C}$ against PRF security of H_1 using an adversary $\mathcal{A}$ against $\mathsf{G}_2/\mathsf{G}_3$ for the proof of Theorem 10.

$\underline{\mathcal{C}^{\text{Eval}}}$
01 **for** $i \in [n]$
02 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{Gen}$
03 $\ell \leftarrow 0$
04 $\mathcal{E}, \Lambda \leftarrow \emptyset$
05 $b \xleftarrow{\$} \{0,1\}$
06 $b' \xleftarrow{\$} \mathcal{A}^{\text{AEncap},\text{ADecap}}(pk_1, \ldots, pk_n)$
07 **return** $[\![b = b']\!]$

Oracle AEncap$(i \in [n], pk \in \mathcal{PK}')$
08 $(sk^*, pk^*) \xleftarrow{\$} \mathsf{NIKE.KeyGen}$
09 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_i, pk)$
10 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk^*, pk)$
11 **if** $\exists \ell' : (\ell', \{pk_i, pk\}) \in \Lambda$
12 $\quad$ **if** $\exists h : (h, \ell', K_2, pk_i||pk||pk^*) \in \mathcal{H}$
13 $\quad\quad$ **abort**
14 $\quad$ **else**
15 $\quad\quad K \leftarrow \text{Eval}(\ell', K_2||pk_i||pk||pk^*)$ //eval query
16 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, \ell', K_2, pk_i||pk||pk^*)\}$
17 **else if** $pk \in \{pk_1, \ldots, pk_n\}$
18 $\quad \ell \leftarrow \ell + 1$ //new key
19 $\quad \Lambda \leftarrow \Lambda \cup \{\ell, \{pk_i, pk\}\}$
20 $\quad K \leftarrow \text{Eval}(\ell, K_2||pk_i||pk||pk^*)$ //eval query
21 $\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, \ell, K_2, pk_i||pk||pk^*)\}$
22 **else**
23 $\quad K \leftarrow \mathsf{H}(K_1, K_2, pk_i||pk||c)$
24 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, pk^*, K)\}$
25 **return** (pk^*, K)

Oracle ADecap$(pk \in \mathcal{PK}', j \in [n], c)$
26 **if** $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
27 $\quad$ **return** K
28 $K \leftarrow \mathsf{AuthDecap}(pk, sk_j, c)$
29 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, pk)$
30 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk_j, c)$
31 **if** $\exists \ell' : (\ell', \{pk, pk_j\}) \in \Lambda$
32 $\quad$ **if** $\exists h : (h, \ell', K_2, pk||pk_j||c) \in \mathcal{H}$
33 $\quad\quad$ **abort**
34 $\quad$ **else**
35 $\quad\quad K \leftarrow \text{Eval}(\ell', K_2||pk_i||pk||pk^*)$ //eval query
36 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, \ell', K_2, pk||pk_j||c)\}$
37 **else if** $pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \perp$
38 $\quad \ell \leftarrow \ell + 1$ //new key
39 $\quad \Lambda \leftarrow \Lambda \cup \{(\ell, \{pk, pk_j\})\}$
40 $\quad K \leftarrow \text{Eval}(\ell, K_2, pk||pk_j||c)$ //eval query
41 $\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{(K, \ell, K_2, pk||pk_j||c)\}$
42 **else**
43 $\quad K \leftarrow \mathsf{H}(K_1, K_2, pk||pk_j||c)$
44 **if** $b = 1 \wedge pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \perp$
45 $\quad K \xleftarrow{\$} \mathcal{K}$
46 $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$
47 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\}$
48 **return** K

In the resulting game, challenge bit b is completely hidden and the winning probability is $\frac{1}{2}$ from which the stated bound follows. □