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

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

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

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

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.

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

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 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

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

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 pskPKE[KEM, KS, AEAD] from a KEM, a key schedule KS, and an AEAD. To encrypt a message, a KEM ciphertext/key pair (c, K) is generated. 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., $\mathcal{PK}' := \{pk \in \{0,1\}^* \mid \bot \notin \mathsf{Encaps}(pk)\}$. 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 $\mathcal{PK}$.

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$(j \in [n], c)$

```
07 if ∃K : (pk_j, c, K) ∈ E
08   return K
09 K ← Decaps(sk_j, c)
10 return K
```

Oracle CHALL$(j \in [n])$

```
11 (c, K) ←$ Encaps(pk_j)
12 if b = 1
13   K ←$ K
14   E ← E ∪ {(pk_j, c, K)}
15 return (c, K)
```

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

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

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* $\mathcal{K}$*.*
- $\mathsf{AuthEncap}$ *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* $\bot$*).*
- *Deterministic* $\mathsf{AuthDecap}$ *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* $\bot$*).*

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.

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

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

```
01 for i ∈ [n]
02   (sk_i, pk_i) ←$ Gen
03 E, Γ_pk ← ∅
04 b ←$ {0,1}
05 b' ←$ A^{AEncap,ADecap,Chall,RepSK}(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 return (c, K)
```

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

```
09 if ∃K : (pk, pk_j, c, K) ∈ E
10   return K
11 K ← AuthDecap(pk, sk_j, c)
12 return K
```

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

```
13 if j ∈ Γ_pk
14   return ⊥
15 (c, K) ←$ AuthEncap(sk_i, pk_j)
16 if b = 1
17   K ←$ K
18   E ← E ∪ {(pk_i, pk_j, c, K)}
19 return (c, K)
```

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

```
20 (pk_i, sk_i) ← (μ(sk), sk)
21 Γ_pk ← Γ_pk ∪ {i}
```

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
```

2.3 Authenticated Public Key Encryption

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

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

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

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

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

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

Oracle AEnc$(i \in [n], pk \in \mathcal{PK}', m, aad, info)$

```
07 c ←$ 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) ∈ E
10   return ⊥
11 m ← 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| ≠ |m_1| return ⊥
14 c ←$ AuthEnc(sk, pk_j, m_b, aad, info)
15 E ← E ∪ {(μ(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 ∈ [n]
02   (sk_i, pk_i) ←$ Gen
03 E ← ∅
04 (i*, j*, c*, aad*, info*) ←$ A^{AEnc,ADec}(pk_1, ..., pk_n)
05 return [[(pk_{i*}, pk_{j*}, c*, aad*, info*) ∉ E
     ∧ AuthDec(pk_{i*}, sk_{j*}, c*, aad*, info*) ≠ ⊥]]
```

Oracle AEnc$(i \in [n], pk \in \mathcal{PK}', m, aad, info)$

```
06 c ←$ AuthEnc(sk_i, pk, m, aad, info)
07 E ← E ∪ {(pk_i, pk, c, aad, info)}
08 return c
```

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

```
09 m ← AuthDec(pk, sk_j, c, aad, info)
10 return m
```

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*

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

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* $\mathsf{AEAD.Enc}$ *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* $\mathsf{AEAD.Dec}$ *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)-$\mathsf{INT\text{-}CTXT}$ in Listing 6 and (n, q_d)-CCA in Listing 7. Note that an AEAD scheme which is $\mathsf{IND\text{-}CPA}$ and $\mathsf{INT\text{-}CTXT}$ secure is also CCA secure [5]. The advantage of an adversary $\mathcal{A}$ is

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

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

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

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

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	Oracle $\textsc{Enc}(i \in [n], m, aad)$
01 **for** $i \in [n]$	08 $c \leftarrow \mathsf{AEAD.Enc}(k_i, m, aad, nonce_i)$
02 $\quad k_i \xleftarrow{\$} \mathcal{K}'$	09 $\mathcal{E} \leftarrow \mathcal{E} \cup \{i, m, c, aad\}$
03 $\quad nonce_i \xleftarrow{\$} \{0,1\}^{N_{nonce}}$	10 **return** $(c, nonce_i)$
04 $\mathcal{E} \leftarrow \emptyset$	
05 $b \xleftarrow{\$} \{0,1\}$	Oracle $\textsc{Dec}(i \in [n], c, aad)$
06 $b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec}}$	11 **if** $b = 0$
07 **return** $[\![b = b']\!]$	12 $\quad m \leftarrow \mathsf{AEAD.Dec}(k_i, c, aad, nonce_i)$
	13 **else if** $\exists m' : (i, m', c, aad) \in \mathcal{E}$
	14 $\quad m \leftarrow m'$
	15 **else**
	16 $\quad m \leftarrow \perp$
	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	Oracle $\textsc{Enc}(i \in [n], m_0, m_1, aad)$
01 **for** $i \in [n]$	08 $c \leftarrow \mathsf{AEAD.Enc}(k_i, m_b, aad, nonce_i)$
02 $\quad k_i \xleftarrow{\$} \mathcal{K}'$	09 $\mathcal{E} \leftarrow \mathcal{E} \cup \{i, c, aad\}$
03 $\quad nonce_i \xleftarrow{\$} \{0,1\}^{N_{nonce}}$	10 **return** $(c, nonce_i)$
04 $\mathcal{E} \leftarrow \emptyset$	
05 $b \xleftarrow{\$} \{0,1\}$	Oracle $\textsc{Dec}(i \in [n], c, aad)$
06 $b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec}}$	11 **if** $(i, c, aad) \in \mathcal{E}$
07 **return** $[\![b = b']\!]$	12 $\quad$ **return** $\perp$
	13 $m \leftarrow \mathsf{AEAD.Dec}(k_i, c, aad, nonce_i)$
	14 **return** m

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* $\mathsf{Adv}^{(n,q_s)\text{-SUF-CMA}}_{\mathcal{A},\mathsf{SIG}} = \Pr[(n, q_s)\text{-SUF-CMA}(\mathcal{A}) \Rightarrow 1]$.

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* $\perp$*.*

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 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 $(i^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\text{SIGN}}(vk_1, \ldots, vk_n)$	
05 **return** $[\![\mathsf{Vfy}(vk_{i^*}, m^*, \sigma^*) = 1$	
$\quad \wedge (i^*, m^*, \sigma^*) \notin Q]\!]$	

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*

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

The security definition for actively secure NIKE can be found in the full version.

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 $m \in \{0,1\}^*$, $aad \in \{0,1\}^*$, $info \in \{0,1\}^*$,

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

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 $m \in \{0,1\}^*$, $aad \in \{0,1\}^*$, $info \in \{0,1\}^*$,

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

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

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

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 $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \Gamma_{\mathrm{pk}}, \Gamma_{\mathrm{psk}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chall},\textsc{RepPK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
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 $|m_0| \neq |m_1| \vee (j \in \Gamma_{\mathrm{pk}} \wedge (i,j) \in \Gamma_{\mathrm{psk}})$
  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$(j \in [n], pk \in \mathcal{PK}')$

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

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

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

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 $j \in [i]$
    $psk_{ij} \xleftarrow{\$} \mathsf{GenPSK}$
    $psk_{ji} \leftarrow psk_{ij}$
$\mathcal{E}, \Gamma_c, \Gamma_{\mathrm{pk}}, \Gamma_{\mathrm{psk}} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Chall},\textsc{RepPK},\textsc{RepSK},\textsc{RepPSK}}(pk_1, \ldots, pk_n)$
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 $|m_0| \neq |m_1| \vee sk_i = \bot \vee (j \in \Gamma_{\mathrm{pk}} \wedge (i,j) \in \Gamma_{\mathrm{psk}})$
  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$(j \in [n], pk \in \mathcal{PK}')$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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 the full version. □

Theorem 2 (AKEM Insider-CCA + KS_1 PRF + KS_2 PRF + AEAD CCA $\Rightarrow$ 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*

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

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 the full version. □

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* $\mathsf{pskPKE}[\mathsf{KEM}, \mathsf{KS}, \mathsf{AEAD}]$*, 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*

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

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 the full version. □

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* $\mathsf{pskAPKE}[\mathsf{AKEM}, \mathsf{KS}, \mathsf{AEAD}]$*, 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 $(i^*, j^*, (c_1^*, c_2^*), aad^*, info^*) \xleftarrow{\$} \mathcal{A}^{\text{Enc,Dec,RepPK,RepPSK}}(pk_1, \ldots, pk_n)$
08 **return** $[\![(i^* \notin \Gamma_{\mathtt{pk}} \vee (i^*, j^*) \notin \Gamma_{\mathtt{psk}}) \wedge sk_{j^*} \neq \bot$
$\quad\wedge (pk_{i^*}, pk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \notin \mathcal{E}$
$\quad\wedge \mathsf{pskADec}(pk_{i^*}, sk_{j^*}, psk_{i^*j^*}, (c_1^*, c_2^*), aad^*, info^*) \neq \bot]\!]$

$\underline{\text{Oracle Enc}(i \in [n], j \in [n], m, aad, info)}$
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** $j \notin \Gamma_{\mathtt{pk}}$ $\quad/\!/\mathsf{G}_2 - \mathsf{G}_8$
14 $\quad K \xleftarrow{\$} \mathcal{K}$ $\quad/\!/\mathsf{G}_2 - \mathsf{G}_8$
15 $\quad\hat{\mathcal{E}} \leftarrow \hat{\mathcal{E}} \cup \{(pk_i, pk_j, c_1, K)\}$ $\quad/\!/\mathsf{G}_2 - \mathsf{G}_8$
16 $\quad(k, nonce) \leftarrow \mathsf{KS}(K, psk_{ij}, info)$ $\quad/\!/\mathsf{G}_2 - \mathsf{G}_8$
17 $\quad$**if** $\exists k', nonce' : (k', nonce', i, j, K, psk_{ij}, info) \in \Lambda$ $\quad/\!/\mathsf{G}_1 - \mathsf{G}_8$
18 $\quad\quad$**abort** $\quad/\!/\mathsf{G}_3 - \mathsf{G}_8$
19 $\quad\quad(k, nonce) \leftarrow (k', nonce')$ $\quad/\!/\mathsf{G}_1 - \mathsf{G}_8$
20 $\quad(k, nonce) \xleftarrow{\$} \mathcal{K}' \times \{0,1\}^{N_{nonce}}$ $\quad/\!/\mathsf{G}_4 - \mathsf{G}_8$
21 $\quad\Lambda \leftarrow \Lambda \cup \{(k, nonce, i, j, K, psk_{ij}, info)\}$ $\quad/\!/\mathsf{G}_1 - \mathsf{G}_8$
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 $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(k, nonce, m, (c_1, c_2), aad)\}$ $\quad/\!/\mathsf{G}_8$
25 **return** (c_1, c_2)

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

$\underline{\text{Oracle RepPK}(i \in [n], pk)}$
53 $(sk_i, pk_i) \leftarrow (\bot, pk)$
54 $\Gamma_{\mathtt{pk}} \leftarrow \Gamma_{\mathtt{pk}} \cup \{i\}$

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

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

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:

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

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 $\textsc{Dec}_{\text{AEAD}}$.

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 $(i^*, j^*, K^*, psk_{i^*j^*}, \mathit{info}^*)$ 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

Listing 16: Adversary $\mathcal{D}_1$ against INT-CTXT security for AEAD having access to oracles ENC_{AEAD} and DEC_{AEAD}.

```
D_1^{Enc_AEAD, Dec_AEAD}
01 for i ∈ [n]
02   (sk_i, pk_i) ←$ GenSK
03   for j ∈ [i]
04     psk_ij ←$ K_psk
05     psk_ji ← psk_ij
06 E, E', Γ_pk, Γ_psk, Λ ← ∅
07 (i*, j*, (c_1*, c_2*), aad*, info*) ←$ A^{Enc,Dec,RepPK,RepPSK}(pk_1, ..., pk_n)
08 return [[(i* ∉ Γ_pk ∨ (i*, j*) ∉ Γ_psk) ∧ sk_{j*} ≠ ⊥
     ∧ (pk_{i*}, pk_{j*}, psk_{i*j*}, (c_1*, c_2*), aad*, info*) ∉ E
     ∧ pskADec(pk_{i*}, sk_{j*}, psk_{i*j*}, (c_1*, c_2*), aad*, info*) ≠ ⊥]]

Oracle Enc(i ∈ [n], j ∈ [n], m, aad, info)
09 if sk_i = ⊥
10   return ⊥
11 (c_1, K) ←$ AuthEncap(sk_i, pk_j)
12 (k, nonce) ← KS(K, psk_ij, info)
13 if j ∉ Γ_pk
14   K ←$ K
15   Ê ← Ê ∪ {(pk_i, pk_j, c_1, K)}
16   if ∃ℓ' : (ℓ', i, j, K, psk_ij, info) ∈ Λ
17     abort
18   ℓ ← ℓ + 1                                  //new key
19   c_2 ← Enc_AEAD(ℓ, m, aad)                  //enc query
20   Λ ← Λ ∪ {(ℓ, i, j, K, psk_ij, info)}
21 else
22   c_2 ← AEAD.Enc(k, m, aad, nonce)
23 E ← E ∪ {(pk_i, pk_j, psk_ij, (c_1, c_2), aad, info)}
24 E' ← E' ∪ {(k, nonce, m, (c_1, c_2), aad)}
25 return (c_1, c_2)

Oracle Dec(i ∈ [n], j ∈ [n], (c_1, c_2), aad, info)
26 if sk_j = ⊥
27   return ⊥
28 K ← AuthDecap(pk_i, sk_j, c_1)
29 if ∃K' : (pk_i, pk_j, c_1, K') ∈ Ê
30   K ← K'
31   (k, nonce) ← KS(K, psk_ij, info)
32 else if i ∉ Γ_pk ∧ K ≠ ⊥
33   K ←$ K
34   Ê ← Ê ∪ {(pk_i, pk_j, c_1, K)}
35   (k, nonce) ←$ K' × {0,1}^{N_nonce}
36 else
37   (k, nonce) ← KS(K, psk_ij, info)
38 if i ∉ Γ_pk ∨ (i, j) ∉ Γ_psk
39   if ∃ℓ' : (ℓ', i, j, K, psk_ij, info) ∈ Λ
40     m ← Dec_AEAD(ℓ', c_2, aad)               //dec query on old key
41   else if i ∈ Γ_pk
42     ℓ ← ℓ + 1                                //new key
43     m ← Dec_AEAD(ℓ, c_2, aad)                //dec query on new key
44     Λ ← Λ ∪ {(ℓ, i, j, K, psk_ij, info)}
45 else
46   m ← AEAD.Dec(k, c_2, aad, nonce)
47 return m

Oracle RepPK(i ∈ [n], pk)
48 (sk_i, pk_i) ← (⊥, pk)
49 Γ_pk ← Γ_pk ∪ {i}

Oracle RepPSK(i ∈ [n], j ∈ [n], psk)
50 psk_ij ← psk
51 psk_ji ← psk
52 Γ_psk ← Γ_psk ∪ {(i, j), (j, i)}
```

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 prim-

itives, 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.

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

$\underline{\mathsf{Gen}}$	$\underline{\mathsf{Enc}((sk_1, sk_2), (pk_1, pk_2), m, aad, info)}$
01 $(sk_1, pk_1) \xleftarrow{\$} \mathsf{Gen}_1$	06 $(c', K') \xleftarrow{\$} \mathsf{AuthEncap}_2(sk_2, pk_2)$
02 $(sk_2, pk_2) \xleftarrow{\$} \mathsf{Gen}_2$	07 $(c_1, c_2) \xleftarrow{\$} \mathsf{pskAEnc}(sk_1, pk_1, K', m, aad, c' \|\| info)$
03 $sk \leftarrow (sk_1, sk_2)$	08 **return** $((c_1, c_2), c')$
04 $pk \leftarrow (pk_1, pk_2)$	
05 **return** (sk, pk)	$\underline{\mathsf{Dec}((pk_1, pk_2), (sk_1, sk_2), ((c_1, c_2), c'), aad, info)}$
	09 $K' \leftarrow \mathsf{AuthDecap}_2(sk_2, c')$
	10 $m \leftarrow \mathsf{pskADec}(pk_1, sk_1, K', (c_1, c_2), aad, c' \|\| info)$
	11 **return** m

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 *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 the full version. □

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*

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

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 the full version. □

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. 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.

$\underline{\mathsf{Gen}}$
01 $(sk, pk) \xleftarrow{\$} \mathsf{KEM.Gen}$
02 $(sigk, vk) \xleftarrow{\$} \mathsf{SIG.Gen}$
03 **return** $((sk, sigk), (pk, vk))$

$\underline{\mathsf{AuthEncap}((sk_1, sigk_1), (pk_2, vk_2))}$
04 $(c, K') \xleftarrow{\$} \mathsf{Encaps}(pk_2)$
05 $\sigma \xleftarrow{\$} \mathsf{Sign}(sigk_1, c||\mu(sk_1)||pk_2||vk_2)$
06 $K \leftarrow \mathsf{H}(K', \sigma||\mu(sk_1)||\mu'(sigk_1)||pk_2||vk_2)$
07 **return** $((c, \sigma), K)$

$\underline{\mathsf{AuthDecap}((pk_1, vk_1), (sk_2, sigk_2), (c, \sigma))}$
08 **if** $\mathsf{Vfy}(vk_1, c||pk_1||\mu(sk_2)||\mu'(sigk_2), \sigma) \neq 1$
09 $\quad K \leftarrow \perp$
10 **else**
11 $\quad K' \leftarrow \mathsf{Decaps}(sk_2, c)$
12 $\quad K \leftarrow \mathsf{H}(K', \sigma||pk_1||vk_1||\mu(sk_2)||\mu'(sigk_2))$
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 the full version. □

Theorem 8 (SIG SUF-CMA $\Rightarrow$ 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 the full version. □

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. By H_1, we denote function H keyed in the first component and by H_2 function H keyed in the second component.

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 $K \leftarrow \mathsf{H}(K_1, K_2)$
07 **return** (pk^*, K)

$\underline{\mathsf{AuthDecap}(sk_2, pk_1, pk^*)}$
08 $K_1 \leftarrow \mathsf{NIKE.SharedKey}(sk_2, pk_1)$
09 $K_2 \leftarrow \mathsf{NIKE.SharedKey}(sk_2, pk^*)$
10 $K \leftarrow \mathsf{H}(K_1, K_2)$
11 **return** K

Theorem 9 (NIKE Active + H_2 PRF $\Rightarrow$ 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*

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

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 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 the full version. □

Theorem 10 (NIKE Active + H_1 PRF $\Rightarrow$ 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* $(n, q_e + 2q_d, 0, q_e, 2q_e + 2q_d, q_d)$-$\mathsf{Active}$ *adversary against* NIKE *and a* (q_d, q_d)-PRF *adversary* $\mathcal{C}$ *against* H_1 *such that*

$$\mathsf{Adv}^{(n,q_e,q_d)\text{-}\mathsf{Outsider\text{-}Auth}}_{\mathcal{A},\mathsf{AKEM}^{\mathsf{NIKE}}[\mathsf{NIKE},\mathsf{H}]} \leq \mathsf{Adv}^{(n,q_e+2q_d,0,q_e,2q_e+2q_d,q_d)\text{-}\mathsf{Active}}_{\mathcal{B},\mathsf{NIKE}} + \mathsf{Adv}^{(q_d,q_d)\text{-}\mathsf{PRF}}_{\mathcal{C},\mathsf{H}_1}.$$

Proof (Sketch). The structure is similar to the proof of Theorem 9 except that the test query is embedded in the decapsulation oracle instead of the challenge oracle and that it is only embedded for queries with honest public keys. The full proof can be found in the full version. □

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
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
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
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
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)
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. 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
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)

9. Cramer, R., Shoup, V.: SIAM Journal on Computing
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
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
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)
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
14. Gajland, P., de Kock, B., Quaresma, M., Malavolta, G., Schwabe, P.: Swoosh: Practical lattice-based non-interactive key exchange. Cryptology ePrint Archive (2023)
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
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
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
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)
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
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
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
22. Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $\ll$ 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