K-Waay: Fast and Deniable Post-Quantum X3DH without Ring Signatures*

Daniel Collins[1], Loïs Huguenin-Dumittan[†1], Ngoc Khanh Nguyen[‡2], Nicolas Rolin[§3], and Serge Vaudenay[1]

[1]EPFL, Switzerland, `firstname.lastname@epfl.ch`
[2]King's College London, United Kingdom, `ngoc_khanh.nguyen@kcl.ac.uk`
[3]Spuerkeess, Luxembourg, `nicrolin@hotmail.fr`

January 27, 2024

Abstract

The Signal protocol and its X3DH key exchange core are regularly used by billions of people in applications like WhatsApp but are unfortunately not quantum-secure. Thus, designing an efficient and post-quantum secure X3DH alternative is paramount. Notably, X3DH supports *asynchronicity*, as parties can immediately derive keys after uploading them to a central server, and *deniability*, allowing parties to plausibly deny having completed key exchange. To satisfy these constraints, existing post-quantum X3DH proposals use ring signatures (or equivalently a form of designated-verifier signatures) to provide authentication without compromising deniability as regular signatures would. Existing ring signature schemes, however, have some drawbacks. Notably, they are not generally proven secure in the quantum random oracle model (QROM) and so the quantum security of parameters that are proposed is unclear and likely weaker than claimed. In addition, they are generally slower than standard primitives like KEMs.

In this work, we propose an efficient, deniable and post-quantum X3DH-like protocol that we call K-Waay, that does *not* rely on ring signatures. At its core, K-Waay uses a split-KEM, a primitive introduced by Brendel et al. [SAC 2020], to provide Diffie-Hellman-like implicit authentication and secrecy guarantees. Along the way, we revisit the formalism of Brendel et al. and identify that additional security properties are required to prove a split-KEM-based protocol secure. We instantiate split-KEM by building a protocol based on the Frodo key exchange protocol relying on the *plain* LWE assumption: our proofs might be of independent interest as we show it satisfies our novel unforgeability and deniability security notions. Finally, we complement our theoretical results by thoroughly benchmarking both K-Waay and existing X3DH protocols. Our results show even when using plain LWE and a conservative choice of parameters that K-Waay is significantly faster than previous work.

*This is the full version of an article published at USENIX Security 2024. [Col+24]

†The author is supported by a grant (project no. 192364) of the Swiss National Science Foundation (SNSF).

‡All of this work was completed while the author was working at EPFL.

§Part of this work was completed while the author was studying at EPFL.

Contents

1 Introduction

Researchers for several years now have sought to build cryptographic primitives and protocols that are resistant to efficient quantum attacks [Sho94]. This is highly evidenced with the NIST Post-Quantum Cryptography competition for standardising quantum-safe key encapsulation mechanisms (KEM) and signatures, organised by the United States National Institute of Standards and Technology (NIST). Recently, four schemes were selected by NIST for standardisation, out of which three rely on algebraic lattices. Indeed, with the US National Security Agency releasing their new CNSA 2.0 Suite [US], which says that CRYSTALS-Kyber [Bos+18] and CRYSTALS-Dilithium [Duc+18] should be the main cryptographic force for communication security beginning from 2030, lattices are a natural candidate for building more advanced cryptographic primitives, such as secure messaging.

The widely used Signal protocol for secure messaging as currently deployed is not quantum-safe since it is based on Diffie-Hellman key exchange [DH76]. The protocol, used in applications like Signal and WhatsApp, comprises two components, namely 1) the X3DH key exchange [MP16] which is used to bootstrap sessions of 2) the Double Ratchet messaging protocol [PM16]. The Double Ratchet has been investigated in a line of recent works [ACD19; Bie+22; Can+22] that each neatly abstract the protocol into primitives like so-called *continuous key agreement*. Fortunately, these primitives have post-quantum (PQ) instantiations that leaves the core structure and resulting security guarantees of the Double Ratchet in place.

In standard X3DH, parties use a mixture of ephemeral (one-time), semi-static (many-time but temporary) and long-term keys. First, parties upload their keying material to a central server or public key infrastructure in a so-called prekey bundle. A party can then derive a session key by downloading their partner's bundle and performing three (or four) Diffie-Hellman key exchanges with a mixture of ephemeral and long-term (resp. plus semi-static) keys, ensuring at least confidentiality if the ephemeral *or* long-term key of each party is corrupted.

Observe that X3DH does not use signatures after signed prekeys are uploaded: at that point, the DH exchanges provide (implicit) authentication guarantees. Consequently, the protocol provides a level of *deniability* [DNS04; RGK06; UG15] as was formalized by Vatandas et al. [Vat+20]: informally, a participant can deny having performed key exchange with its counterpart. This is an important privacy guarantee that prevents (at least on a cryptographic level) a conversation transcript from incriminating an unsuspecting party, particularly in situations like whistleblowing and protesting.

In 2023, Signal announced and rolled out their initial *hybrid* post-quantum key exchange solution called PQXDH [KS23]. Like in X3DH, several Diffie-Hellman key exchanges are performed at once, but in PQXDH, parties upload prekey bundles that also contain a Kyber-1024 public key that the initiator additionally encapsulates to the responder with. Moreover, prekey bundles are still signed with the same signature scheme as regular X3DH based on Curve25519 [Ber06]. Although PQXDH provides post-quantum confidentiality [Bha+23], which is an important first step towards post-quantum security as it prevents "store-now-decrypt-later" attacks, it does not provide post-quantum *authentication* as an active quantum attacker can trivially forge pre-key bundles. It is thus prudent to design a suitable X3DH alternative that is *fully* post-quantum secure.

A natural direction for building such a protocol is to emulate X3DH's structure by replacing Diffie-Hellman key exchange with a cryptographic group action, such as CSIDH [Cas+18]. In order to broadly capture this protocol structure, Brendel et al. [Bre+21] introduce a primitive called *split-KEM* that captures the symmetry of e.g. Diffie-Hellman. In a split-KEM, a party B encapsulates to their partner A by using their own secret sk_B and their partner's public key pk_A to produce a ciphertext; A then decapsulates it using sk_A and pk_B. The authors define indistinguishability-based security notions and notice that Frodo [Bos+16] lattice-based key-exchange fulfills the split-KEM syntax and the weakest notion of indistinguishability they define.[1] Although they present a X3DH-like protocol, they do not define a security model, and, looking ahead, their split-KEM security notions do not suffice to construct a X3DH-like key exchange with authenticity and deniability.

In two recent works, Hashimoto et al. [Has+21; Has+22] and Brendel et al. [Bre+22] concurrently proposed instead to construct X3DH-like key exchange using KEMs directly. Since a core feature of X3DH is its *asynchronicity*, a challenge-response protocol cannot be employed using KEMs alone to provide authentication [SSW20]. Thus to ensure deniability, two seemingly different approaches were proposed: Hashimoto et al. [Has+21] apply ring signatures while Brendel et al. [Bre+22] use a flavour of designated verifier signatures; these primitives were later shown to be equivalent [Has+22].

As described in the aforementioned works, the currently most efficient post-quantum ring signatures [BKP20; ESZ22; LAZ19; LN22; Yue+21] are proven to be secure in the random oracle model [BR93] and can enjoy signatures that are a handful of kilobytes large. Often, however, the constructions do not come with a security proof in the quantum random oracle model (QROM) [Bon+11]. In this vein, parameters are generally optimistically chosen as the security loss incurred by proofs in the ROM is not taken into account when setting them, without even mentioning QROM loss, which is usually much larger. Further, security notions can differ between papers, making it less clear exactly when they are appropriate for use.

More generally, it is of interest to determine the cost (or overhead) that deniability incurs in (X3DH-like) key exchange. Towards this goal, Hashimoto et al. [Has+22] provide benchmarks for their baseline, non-deniable X3DH-like protocol based on signatures and KEMs, and Brendel et al. [Bre+22] consider parameter sizes for (but do not benchmark) existing ring and designated verifier signatures. As such, a more fine-grained and detailed evaluation will help inform practitioners on the overhead incurred by deniability in the post-quantum setting.

While the use of ring signatures to build PQ and deniable X3DH is at least theoretically understood, this far from exhausts the protocol design space. Motivated by this and the above discussion, we therefore first ask: Can we design a provably-secure, efficient and deniable post-quantum X3DH alternative that does *not* require ring signatures?

1.1 Our Results

In this work, we propose an efficient, deniable and post-quantum X3DH-like protocol without ring signatures that we call K-Waay. To summarise our contributions:

- Towards building our protocol, we revisit the split-KEM formalism proposed by Brendel et al. [Bre+21] and deduce that several additional properties, namely notions of authenticity and deniability, are needed to construct a secure X3DH-like deniable authenticated key exchange protocol (DAKE).

- We propose K-Waay, a X3DH-like DAKE that uses deniable and unforgeable split-KEM at its core. Our protocol uses signatures to sign prekeys, and then uses ephemeral KEM, long-term KEM and split-KEM for the final key exchange step. We compare the security of our protocol with the state-of-the-art in Table 1.

- The main drawback of a naive version of our protocol is that parties can run out of ephemeral keys, thus making the protocol synchronous if this happens (e.g. Alice needs to wait for Bob's fresh ephemeral key before sending a message). While such a problem would rarely occur in practice, given enough keys are uploaded on the server, we propose

[1] The construction can conceptually be seen as instantiating the lattice-based cryptographic group action from [BKP20].

Protocol	PQ Conf	PQ Auth	KCI	FS	SSR	RR	Deniability
X3DH [MP16; Coh+20]	✗	✗	✓	PFS	✓	✓	Malicious
PQXDH [KS23; Bha+23]	✓	✗	✓	PFS	✗	✗	Semi-honest+
KEM+Sigs [Has+22]	✓	✓	✓	PFS	✓	✗	✗
HKKP [Has+22]	✓	✓	✓	WFS	✓	✗	Semi-honest
SPQR [Bre+22]	✓	✓	✓	WFS	✗	✓	Semi-honest
K-Waay (Section 5)	✓	✓	✓	WFS	✓	✗	Semi-honest

Table 1: Comparison between different security properties proven for existing X3DH-like key exchange protocols, namely post-quantum confidentiality (PQ Conf), authentication (PQ Auth), resistance to key-compromise impersonation attacks when long-term keys are exposed (KCI), perfect forward secrecy (PFS) or weak forward secrecy (WFS) [Kra05], session state reveal (SSR), randomness reveal (RR) and deniability (where the judge/adversary is either honest-but-curious or is malicious and can inject messages). Protocols can be generically strengthened to handle randomness reveal by standard application of the so-called NAXOS trick [LLM07]. "KEM+Sigs" refers to the non-deniable baseline X3DH-like protocol proposed by Hashimoto et al. [Has+21; Has+22], and "HKKP" refers to their deniable X3DH protocol *without* NIZKs (their protcool with NIZKs implies maliciously-secure deniability w.r.t. a *classical* adversary). The security of PQXDH is based on the recent analysis of Bhargavan et al. [Bha+23] except that Kret and Schmidt argue it also provides at least semi-honest deniability [KS23].

a simple trick that makes the reuse of ephemeral keys possible on the receiver's side for messages they received while offline. We think this trick could be of independent interest as it – perhaps surprisingly – allows for a specific kind of key reuse for a split-KEM that is *not* IND-CCA secure.

- We prove key indistinguishability in our model that captures ephemeral key reuse and session state exposure, and prove a variant of deniability that strengthens the notion of Brendel et al. [Bre+22] by additionally leaking the victim's session state to the adversary in the security game.

- We instantiate a post-quantum split-KEM secure under our new security notions derived from the Frodo key exchange protocol (FrodoKEX) [Bos+16] based on the plain LWE assumption. The parameters we chose provide strong security guarantees, providing more than 192 bits of classical and quantum security for our core split-KEM security notions OW-CPA, decaps-OW-CPA and deniability. We then use a transform in the (Q)ROM to prove it UNF-1KCA and IND-1BatchCCA (i.e. our new unforgeability and indistinguishability definitions for split-KEM). This construction incurs a security loss as usual in the (Q)ROM, but our final split-KEM still provides around 128 (resp. 64) bits of security in the ROM (resp. QROM) assuming the adversary is limited to 2^{64} (resp. quantum) random oracle queries.

- We benchmark our protocol K-Waay using our modified version of FrodoKEX (which we call FrodoKEX+) as the split-KEM, along with standard X3DH and the two previous proposals for PQ X3DH-like AKE [Has+22; Bre+22]. We find that while K-Waay has larger prekeys, it is 6× faster compared to these. In addition, the only non-standard primitive we use in K-Waay (i.e. FrodoKEX+) is based on both an assumption (i.e. LWE) and a scheme (FrodoKEM) that have been thoroughly scrutinized by the cryptographic community. Overall, we believe our protocol is more mature and therefore suitable for short to medium-term integration compared to previous work based on ring signatures.

1.2 Technical Overview

X3DH-like key exchange. A quantum-secure X3DH-like protocol should satisfy certain properties. Apart from satisfying standard authenticated key exchange (AKE) properties like secrecy and authentication, it should also be *asynchronous*. That is, parties should be able to upload keying material to a central server, after which an initiating party can derive a session key immediately with their counterpart who may be offline. This also entails *receiver-obliviousness*, using the language of Hashimoto et al. [Has+22], as the initial key upload should not depend on the keys of any other party. Another is *deniability*, allowing parties to claim that they plausibly did not participate in the key exchange. Note that we cannot possibly ensure that parties can claim that they never uploaded prekeys as they are signed (and using primitives like ring signatures would violate receiver-obliviousness). Finally, a DAKE should, like X3DH, provides security guarantees even if the session state of a party is leaked.

Revisiting split-KEM. In an attempt to model the primitive central to X3DH-like AKE, Brendel et al. [Bre+21] introduced *split-KEM*, which is similar to a standard KEM except the encapsulator can contribute to the derived key. However, we discovered that the accompanying security definitions were not sufficient to use such a primitive as the main component of a key exchange protocol. The reason being is that their notions ensure that an encapsulated ciphertext will not leak information on its encapsulated key, but not that only the sender can send a "legitimate" ciphertext to the sender (or that only the sender and receiver can derive a common key). In other words, there is no guarantee of implicit authentication. Therefore, we introduce the notion of unforgeability against one known-ciphertext attacks for split-KEM (UNF-1KCA), which ensures that if Alice receives a message allegedly sent by Bob, either Bob really sent it or the decapsulation will fail. Jumping ahead, this will be used in the security proof of the protocol to argue that either the adversary relayed a legitimate split-KEM ciphertext to the receiver that the adversary cannot learn the decapsulation of, or the receiver aborts as the ciphertext is forged.

We also introduce an intermediary notion that we call decaps-OW-CPA, which enforces that an adversary should not be able to recover a key *decapsulated* by some party without knowing the sender's or receiver's secret key. We will prove that our lattice-based split-KEM satisfies this notion, then we will apply some transform in the (Q)ROM to obtain a UNF-1KCA split-KEM.

Finally, we also define a notion capturing deniability for split-KEM, which states that no judge J can be convinced that a party B sent a given ciphertext to A, even knowing A's secret key but assuming both parties did not deviate from the protocol. This models a setting where A communicates with B and later tries to frame the latter by giving the transcript and their own secret key to J.

Construction. As any X3DH-like protocol, our construction works in 4 phases: long-term key generation, prekey generation, sending and receiving. The first observation we make is that in X3DH, prekey bundles are signed with a long-term signing key before being uploaded to the server. This fact is often abstracted away in formal analysis as it hurts the claims one can make about the deniability of X3DH: as a signature is undeniable by definition, users cannot deny they *participated* in the protocol. Based on this, our goal was to achieve some level of peer-deniability [CF11], where parties can deny they communicated with someone in particular, and to leverage the fact that we use signatures to authenticate the prekeys. Our protocol works then as follows (see Figure 1 for a high-level overview). The long-term key pair consists of a KEM and signature key pair, the latter being used to sign the prekey, which comprises an ephemeral KEM key pair and ephemeral split-KEM key pair. The former is used for forward secrecy while the second is used for implicit authentication of the sender. Although usually ephemeral keys cannot be used for authentication as they are dynamic, in our case we can since they are authenticated (i.e. signed) by their owner. Then, the sender encapsulates against both

$$
\begin{array}{ccc}
\text{Bob } (\mathsf{lpk_A}, \mathsf{vk_A}) & & \text{Alice } (\mathsf{lpk_B}, \mathsf{vk_B}) \\
\hline
(\mathsf{esk}_\mathsf{B}^\mathsf{skem}, \mathsf{epk}_\mathsf{B}^\mathsf{skem}) \leftarrow\!\!\$\ \mathsf{KeyGenB}_\mathsf{sKEM} & & (\mathsf{esk}_\mathsf{A}^\mathsf{skem}, \mathsf{epk}_\mathsf{A}^\mathsf{skem}) \leftarrow\!\!\$\ \mathsf{KeyGenA}_\mathsf{sKEM} \\
 & & (\mathsf{esk}_\mathsf{A}^\mathsf{kem}, \mathsf{epk}_\mathsf{A}^\mathsf{kem}) \leftarrow\!\!\$\ \mathsf{KeyGen}_\mathsf{KEM} \\
 & \xleftarrow{\{\mathsf{epk}_\mathsf{A}^\mathsf{skem}, \mathsf{epk}_\mathsf{A}^\mathsf{kem}\}_{\mathsf{sk_A}}} \quad \xrightarrow{\{\mathsf{epk}_\mathsf{B}^\mathsf{skem}\}_{\mathsf{sk_B}}} & \\
\hdashline
K_\ell, \mathsf{ct}_\ell \leftarrow\!\!\$\ \mathsf{Encaps}_\mathsf{KEM}(\mathsf{lpk_A}) & & \\
K_e^\mathsf{skem}, \mathsf{ct}_e^\mathsf{skem} \leftarrow\!\!\$\ \mathsf{Encaps}_\mathsf{sKEM}(\mathsf{epk}_\mathsf{A}^\mathsf{skem}, \mathsf{esk}_\mathsf{B}^\mathsf{skem}) & & \\
K_e^\mathsf{kem}, \mathsf{ct}_e^\mathsf{kem} \leftarrow\!\!\$\ \mathsf{Encaps}_\mathsf{KEM}(\mathsf{epk}_\mathsf{A}^\mathsf{kem}) & & \\
K \leftarrow \mathsf{KDF}(K_\ell, K_e^\mathsf{skem}, K_e^\mathsf{kem}) & \xrightarrow{\mathsf{ct}_\ell, \mathsf{ct}_e^\mathsf{skem}, \mathsf{ct}_e^\mathsf{kem}} & \\
 & & K'_\ell \leftarrow \mathsf{Decaps}_\mathsf{KEM}(\mathsf{lsk_A}, \mathsf{ct}_\ell) \\
 & & K'^{\mathsf{skem}}_e \leftarrow \mathsf{Decaps}_\mathsf{sKEM}(\mathsf{epk}_\mathsf{B}^\mathsf{skem}, \mathsf{esk}_\mathsf{A}^\mathsf{skem}, \mathsf{ct}_e^\mathsf{skem}) \\
 & & K'^{\mathsf{kem}}_e \leftarrow \mathsf{Decaps}_\mathsf{KEM}(\mathsf{esk}_\mathsf{A}^\mathsf{kem}, \mathsf{ct}_e^\mathsf{kem}) \\
 & & K'_\ell \overset{?}{=} \perp \vee K'^{\mathsf{skem}}_e \overset{?}{=} \perp \vee K'^{\mathsf{kem}}_e \overset{?}{=} \perp \\
 & & K' \leftarrow \mathsf{KDF}(K'_\ell, K'^{\mathsf{skem}}_e, K'^{\mathsf{kem}}_e)
\end{array}
$$

Figure 1: High-level overview of the K-Waay protocol. Values in brackets $\{\cdot\}_\mathsf{sk}$ are signed with sk and the signature is verified upon reception. For clarity, we omit the calculation and addition of session identifier sid to KDF.

KEM public keys of the receiver, and uses their own split-KEM secret key and the receiver's public key to derive a split-KEM ciphertext. Upon decapsulation, the receiver recovers the three encapsulated keys and combines them using a PRF to derive the shared key.

Ephemeral split-KEM key reuse. The way our protocol is described above works perfectly well if the split-KEM satisfies the UNF-1KCA unforgeability notion introduced above. However, in practice, it could happen that some party, say Alice, is offline for too long and all their ephemeral split-KEM keys have been used. If that occurs, another sender would have to wait for Alice to come online and upload new keys before they can send her a message.

We fix this issue by modifying the protocol as follows: when Alice's ephemeral public keys have run out on the server, a sender can simply reuse one of them. Then, when Alice is back online, she groups the ciphertexts corresponding to the same public key and decrypts all ciphertexts in a group *at once*. If one or more of the split-KEM decapsulations in a group fails, Alice outputs $\perp$ for *all* ciphertexts and, e.g., restarts the protocol. Otherwise, Alice proceeds as before (and *never* decapsulates again using the same split-KEM key). We formally model this key reuse with an algorithm BatchReceive that takes as input a given session state and one or more messages to be received.

Security. We show this version of the protocol is secure assuming the split-KEM satisfies a stronger notion than IND-CPA that we call IND-1BatchCCA (in addition to UNF-1KCA security). This definition is the same as traditional IND-CPA (adapted to the split-KEM syntax), except the adversary can query a decapsulation oracle *once* with multiple public keys and ciphertexts, and the oracle returns $\perp$ if *one or more* of the decapsulations failed, and the resulting keys otherwise. We show that one can easily build an IND-1BatchCCA split-KEM out of a CPA secure one in the (Q)ROM, conveniently using the same transform mentioned above that builds a UNF-1KCA scheme out of a decaps-OW-CPA one.

As in previous protocols [Bre+22; Has+22], the long-term KEM provides implicit authentication of the receiver as only they can decrypt. As mentioned above, the ephemeral KEM provides forward secrecy, and the UNF-1KCA/IND-1BatchCCA split-KEM provides implicit authentication of the sender, as it guarantees that only the sender could have sent a ciphertext that correctly decapsulates (unforgeability), and no adversary knows what is inside that ciphertext (indistinguishability), even after seeing the decapsulation of one batch of ciphertexts encapsulated against the same public key (given no decapsulation failed). We note that the sender-to-receiver authentication depends both on a long-term key (i.e. the signing key) and an ephemeral one (the split-KEM key). Consequently, our model (that allows session state exposure) is more restrictive than that of Hashimoto et al. [Has+22], since in particular it suffices for the adversary to learn a receiver's ephemeral state during key exchange to forge a message that the receiver accepts. Intuitively, this is because split-KEM is effectively a symmetric primitive. Nevertheless, the security that we achieve is stronger than weak forward security without session state exposure.

Deniable split-KEM from lattices. We provide the first lattice-based split-KEM which satisfies both deniability and UNF-1KCA security. Our starting point is the Frodo key-exchange (FrodoKEX) [Bos+16], which was identified (among other schemes) as a split-KEM by Brendel et al. [Bre+21], the security of which relies on the well-known Learning with Errors (LWE) problem [Reg05]. We highlight that the vanilla construction of FrodoKEX does not enjoy the aforementioned properties.[2] Indeed, when looking closely at the security games of deniability and UNF-1KCA, partial information about the secret keys are revealed - thus making a reduction to LWE completely non-trivial. We circumvent this problem in two ways.

First, we reduce deniability of our scheme to a so-called Extended-LWE problem [AP12], where in addition to a standard LWE instance, the adversary is given a short random combination of the secret coefficients. We show that deniability of our scheme reduces straightforwardly to Extended-LWE, and then follow the methodology of Alperin-Sheriff and Peikert [AP12] to reduce it further to plain LWE.

Towards UNF-1KCA security, we slightly modify the Frodo split-KEM by introducing masking terms. As the name suggests, they are used to hide the partial information about secret keys. In Section 7.2 we discuss the necessity of this (perhaps seemingly artificial) change.

1.3 Additional Related Work

The security of X3DH has been modelled in detail by Cohn-Gordon et al. [Coh+20]. Vatandas et al. [Vat+20] investigate the deniability of X3DH and similar key exchange protocols under the deniability notion of Di Raimondo et al. [RGK06], requiring strong knowledge-of-exponent-type assumptions to prove X3DH secure. Dobson and Galbraith [DG22] propose a SIDH-based X3DH-like protocol which is unfortunately now broken [CD23]. Very recently, [Kil+23] prove a simplified version of X3DH tightly-secure in the generic group model under a new multi-user assumption supporting corruptions although do not allow the adversary to expose parties' session states.

Unger and Goldberg build a number of different DAKEs [UG15; UG18]. However, the protocols do not provide post-quantum guarantees: only in their later paper [UG18] is it suggested to add a PQ KEM for post-quantum *confidentiality* and the authors do not propose a more comprehensive hybrid protocol. Nevertheless, the protocols provide relatively strong online deniability (i.e. where a judge and a party can communicate while trying to frame another party) at the expense of stronger primitives like dual-receiver encryption and non-committing encryption.

[2]Nevertheless, we found no practical attack on deniability/UNF-1KCA for FrodoKEX.

Alwen et al. [Alw+21] introduce the notion of authenticated key encapsulation mechanism (AKEM) and some security definitions. AKEM captures the same primitive as a split-KEM, but we opted for the syntax and language of the latter as it was meant to be used in a X3DH-like protocol.

Cremers and Feltz [CF11] introduce peer-deniability, which captures the kind of participation deniability property we are after in this work, namely that a party cannot deny using a system but can deny communicating with a particular party. However, their security notion does not require the simulator to output the session key nor the judge/adversary to distinguish between the real and simulated key, and so composability issues may arise from using it.

2 Preliminaries

In this work, we denote by *efficient* adversary a probabilistic polynomial-time or quantum polynomial-time algorithm unless otherwise specified. Let $[n] = \{1, \dots, n\}$, i.e. the set of integers between 1 and n.

2.1 Key-Encapsulation Mechanism (KEM)

Definition 2.1 (KEM)**.** *A KEM* KEM *is a tuple of three efficient algorithms* $(\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *defined as follows:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$*: The key generation function takes the security parameter* λ *as input, and outputs a pair of public/secret keys* $(\mathsf{pk}, \mathsf{sk})$*.*
- $K, \mathsf{ct} \leftarrow\$ \mathsf{Encaps}(\mathsf{pk})$*: The encapsulation function takes a public key* pk *as input, and outputs a ciphertext* ct *and a key* K*.*
- $K/\perp \leftarrow \mathsf{Decaps}(\mathsf{sk}, \mathsf{ct})$*: The decapsulation function takes a secret key* sk *and a ciphertext* ct *as inputs, and outputs a key* K *or the error symbol* $\perp$*.*

Finally, we say a KEM is $(1-\delta)$*-correct if*

$$\Pr\left[K \neq K' : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda); \\ K, \mathsf{ct} \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}); \\ K' \leftarrow \mathsf{Decaps}(\mathsf{sk}, \mathsf{ct})] \end{array}\right] \leq \delta \; .$$

```
IND-CCA/CPA_KEM(A)
1 : $b \leftarrow\$ \{0,1\}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
3 : $\mathsf{ct}^*, K_0 \leftarrow\$ \mathsf{Encaps}(\mathsf{pk})$
4 : $K_1 \leftarrow\$ \mathcal{K}$
5 : $b' \leftarrow\$ \mathcal{A}^{\mathrm{DEC}}(\mathsf{pk}, \mathsf{ct}^*, K_b)$
6 : return $1_{b'=b}$
```

```
DEC(ct)
1 : if $\mathsf{ct} = \mathsf{ct}^*$ : return $\perp$
2 : $K' \leftarrow \mathsf{Decaps}(\mathsf{sk}, \mathsf{ct})$
3 : return $K'$
```

Figure 2: Indistinguishability games for KEM. In IND-CPA the adversary cannot make any query to DEC.

Definition 2.2 (KEM Indistinguishability)**.** *We consider the games defined in Fig. 2. Let* $\mathcal{K}$ *be a finite key space. A KEM scheme over* $\mathcal{K}$ $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* IND-CCA

```
SUF-CMA_Sig(A)
1 : $L \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
3 : $m^*, \sigma^* \leftarrow\$ \mathcal{A}^{\mathrm{SIGN}}(\mathsf{pk})$
4 : if $\mathsf{Vrfy}(\mathsf{pk}, m^*, \sigma^*)$ and $(m^*, \sigma^*) \notin L$
5 :   return 1
6 : return 0
```

```
SIGN(m)
1 : $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, m)$
2 : $L \leftarrow L \cup \{m, \sigma\}$
3 : return $\sigma$
```

Figure 3: SUF-CMA game.

(resp. IND-CPA*) if for any efficient adversary $\mathcal{A}$ (respectively any efficient adversary with no access to the decapsulation oracle) we have*

$$\mathsf{Adv}^{\text{ind-cca/cpa}}_{\mathsf{KEM}}(\mathcal{A}) := \left| \Pr\left[\text{IND-CCA/CPA}_{\mathsf{KEM}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2} \right| = \mathsf{negl} \ .$$

where $\Pr\left[\text{IND-CCA/CPA}_{\mathsf{KEM}}(\mathcal{A}) \Rightarrow 1\right]$ *is the probability that $\mathcal{A}$ wins the* IND-CCA/CPA$_{\mathsf{KEM}}(\mathcal{A})$ *game defined in Fig. 2.*

2.2 Signature

Definition 2.3. *A signature scheme is a tuple of three efficient algorithms* (KeyGen,Sign,Vrfy)*:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$*: The key generation function outputs a pair of keys.*
- $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, m)$*: The signing function takes as inputs a secret key* sk *and the message to sign m, and it outputs a signature σ.*
- $0/1 \leftarrow \mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$*: The verification function takes as inputs a public key* pk*, the signed message m, and the signature σ, and it outputs either* 0 *or* 1 *(for failure and success, respectively).*

Finally, we say a signature scheme is $(1-\delta)$-correct if for all messages m:

$$\Pr\left[\mathsf{Vrfy}(\mathsf{pk}, m, \sigma) = 0 : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda); \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array}\right] \leq \delta$$

Definition 2.4 (SUF-CMA security)**.** *We consider the game shown in Figure 3. We say a signature scheme* Sig *is* SUF-CMA *if for all efficient adversaries $\mathcal{A}$, we have*

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

2.3 Triple PRF

Definition 2.5 (Triple PRF)**.** *Let $F : \mathcal{K} \times \mathcal{K} \times \mathcal{K} \times D \to R$ be a function. We consider the game shown in Figure 4. We say that F is a* 3PRF *if for all efficient adversaries, we have:*

$$\mathsf{Adv}^{\text{3prf}}_{F}(\mathcal{A}) := \max_{i \in \{1,2,3\}} \left| \Pr\left[\text{PRF}_{F_i}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2} \right| = \mathsf{negl} \ .$$

where F_i denotes F keyed in its i-th argument for $i \in \{1, 2, 3\}$.

The notion of a triple PRF generalises the by now common notion of a *dual PRF* [Bel06]. A triple PRF F_{triple} can be trivially constructed in the random oracle model, or from a dual PRF F_{dual} as $F_{triple}(k_1, k_2, k_3, x) = F_{dual}(k_1, F_{dual}(k_2, k_3, x), x)$.

```
$\mathrm{PRF}_F(\mathcal{A})$
1 : Sample random function $G$
2 : $k \leftarrow\$ \mathcal{K}$
3 : $b \leftarrow\$ \{0,1\}$
4 : $b' \leftarrow\$ \mathcal{A}^{\mathrm{PRF}}(1^\lambda)$
5 : return $1_{b'=b}$
```

```
$\mathrm{PRF}(a, b, c)$
1 : if $b = 0$ :
2 :     return $F_k(a, b, c)$
3 : else :
4 :     return $G(a, b, c)$
```

Figure 4: PRF game for function F_k taking three arguments as input.

3 Split-KEM

As mentioned above, the primitive at the core of our protocol is a split-KEM, which we present in this section. It was first defined by Brendel et al. [Bre+21].

Definition 3.1 (Split-KEM). *An (asymmetric) split-KEM* sKEM *is a tuple of four efficient algorithms* $(\mathsf{KeyGenA}, \mathsf{KeyGenB}, \mathsf{Encaps}, \mathsf{Decaps})$ *defined as follows:*

- $(\mathsf{pk_A}, \mathsf{sk_A}) \leftarrow\$ \mathsf{KeyGenA}(1^\lambda)$ *(resp.* $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow\$ \mathsf{KeyGenB}(1^\lambda)$*): The key generation function of the first/second party takes the security parameter* λ *as input, and outputs a pair of public/secret keys* $(\mathsf{pk_A}, \mathsf{sk_A})$ *(resp.* $(\mathsf{pk_B}, \mathsf{sk_B})$*).*
- $K, \mathsf{ct} \leftarrow\$ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B})$*: The encapsulation function takes the public key* $\mathsf{pk_A}$ *of a party* A *and the other party's secret key* $\mathsf{sk_B}$ *as inputs, and outputs a ciphertext* ct *and a key* K*.*
- $K/\perp \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct})$*: The decapsulation function takes the secret key* $\mathsf{sk_A}$ *of a party* A*, the other party's public key* $\mathsf{pk_B}$ *and a ciphertext* ct *as inputs, and outputs a key* K *or the error symbol* $\perp$*.*

We say a split-KEM is $(1-\delta)$*-correct if*

$$\Pr\left[K \neq K' : \begin{array}{c} (\mathsf{pk_A}, \mathsf{sk_A}) \leftarrow\$ \mathsf{KeyGenA}(1^\lambda); \\ (\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow\$ \mathsf{KeyGenB}(1^\lambda); \\ K, \mathsf{ct} \leftarrow\$ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B}); \\ K' \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct})] \end{array}\right] \leq \delta .$$

Intuitively, a split-KEM is similar to a normal KEM except material from both participants is used for encapsulation (i.e. the final key will depend on both parties' secret/public keys). In a X3DH-like protocol, it can be used to implicitly authenticate the party encapsulating. In the language of Brendel et al. [Bre+21], our notion of split-KEM is "asymmetric", as it is assumed that B always encapsulates and A always decapsulates. This is sufficient for our purpose, but we note that all the results presented in this paper can be adapted to a symmetric split-KEM where $\mathsf{KeyGenA} = \mathsf{KeyGenB}$.

3.1 Security

We will need several security properties from the split-KEM to prove our whole protocol secure. We first define one-wayness (OW-CPA) for sKEM, which is very similar to the usual one for KEM and another new notion called IND-1BatchCCA. Looking ahead, we will show that any OW-CPA split-KEM can easily be transformed into a IND-1BatchCCA one in the (Q)ROM.

Definition 3.2 (split-KEM OW-CPA). *We consider the* OW-CPA *game defined in Figure 5. A split-KEM scheme* $\mathsf{sKEM} = (\mathsf{KeyGen_A}, \mathsf{KeyGenB}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* OW-CPA *if for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\text{ow-cpa}}_{\mathsf{sKEM}}(\mathcal{A}) = \Pr[\text{OW-CPA}_{\mathsf{sKEM}}(\mathcal{A}) \Rightarrow 1] = \mathsf{negl} .$$

Definition 3.3 (split-KEM IND-1BatchCCA)**.** *We consider the* IND-1BatchCCA *game defined in Figure 5. Let* $\mathcal{K}$ *be a finite key space. A split-KEM scheme over* $\mathcal{K}$ $\mathsf{sKEM} = (\mathsf{KeyGen_A}, \mathsf{KeyGenB}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* IND-1BatchCCA *if for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\text{ind-1batchcca}}_{\mathsf{sKEM}}(\mathcal{A}) := \left| \Pr\left[\text{IND-1BatchCCA}_{\mathsf{sKEM}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2} \right| = \mathsf{negl}\ .$$

```
IND-1BatchCCA_sKEM(A)
1 :  $b \leftarrow\!\!\$ \{0,1\};\ q \leftarrow 0$
2 :  $\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\!\!\$ \mathsf{KeyGenA}(1^\lambda)$
3 :  $\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\!\!\$ \mathsf{KeyGenB}(1^\lambda)$
4 :  $K_0, \mathsf{ct}^* \leftarrow\!\!\$ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B})$
5 :  $K_1 \leftarrow\!\!\$ \mathcal{K}$
6 :  $b' \leftarrow\!\!\$ \mathcal{A}^{\text{BatchDec}}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}^*, K_b)$
7 :  return $1_{b'=b}$
```

```
BatchDec({(pk_i, ct_i)}_{i=1}^d)
1 :  if $q = 1$ : return $\perp$
2 :  else : $q \leftarrow q + 1$
3 :  for $i \in \{1, \ldots, d\}$ :
4 :     if $(\mathsf{pk}_i, \mathsf{ct}_i) = (\mathsf{pk_B}, \mathsf{ct}^*)$ : return $\perp$
5 :     $K_i' \leftarrow \mathsf{Decaps}(\mathsf{pk}_i, \mathsf{sk_A}, \mathsf{ct}_i)$
6 :  if $K_1 = \perp \vee \ldots \vee K_d = \perp$ : return $\perp$
7 :  return $(K_1, \ldots, K_d)$
```

```
OW-CPA_sKEM(A)
1 :  $\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\!\!\$ \mathsf{KeyGenA}(1^\lambda);\ \mathsf{pk_B}, \mathsf{sk_B} \leftarrow\!\!\$ \mathsf{KeyGenB}(1^\lambda)$
2 :  $K^*, \mathsf{ct}^* \leftarrow\!\!\$ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B})$
3 :  $K' \leftarrow\!\!\$ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}^*)$
4 :  return $1_{K'=K^*}$
```

Figure 5: IND-1BatchCCA and OW-CPA games.

We also recall the different notions of indistinguishability for (asymmetric) split-KEM defined by Brendel et al. [Bre+21]:

```
xy-IND-CCA_sKEM(A)
1 :  $b \leftarrow\!\!\$ \{0,1\}$
2 :  $n_x \leftarrow 0; n_y \leftarrow 0;$
3 :  $\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\!\!\$ \mathsf{KeyGenA}(1^\lambda)$
4 :  $\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\!\!\$ \mathsf{KeyGenB}(1^\lambda)$
5 :  $K_0, \mathsf{ct}^* \leftarrow\!\!\$ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B})$
6 :  $K_1 \leftarrow\!\!\$ \mathcal{K}$
7 :  $b' \leftarrow\!\!\$ \mathcal{A}^{\text{DEC,ENC}}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}^*, K_b)$
8 :  return $1_{b'=b}$
```

```
ENC(pk)
1 :  if $n_y \geq y$ : return $\perp$
2 :  $n_y \leftarrow n_y + 1$
3 :  $K, \mathsf{ct} \leftarrow\!\!\$ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
4 :  if $(\mathsf{pk}, \mathsf{ct}) = (\mathsf{pk_A}, \mathsf{ct}^*)$ : return $\perp$
5 :  return $K, \mathsf{ct}$
```

```
DEC(pk, ct)
1 :  if $n_x \geq x$ : return $\perp$
2 :  $n_x \leftarrow n_x + 1$
3 :  if $(\mathsf{pk}, \mathsf{ct}) = (\mathsf{pk_B}, \mathsf{ct}^*)$ : return $\perp$
4 :  return $\mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct})$
```

Figure 6: xy-IND-CCA games for an "asymmetric" split-KEM from Brendel et al. [Bre+21], where x,y $\in$ {n, s, m}. When doing the comparison on the first line of both oracles, we assume n = 0, s = 1 and m = ∞.

Definition 3.4 (split-KEM xy-IND-CCA)**.** *We consider the* xy-IND-CCA *game defined in Figure 6. A split-KEM scheme* $\mathsf{sKEM} = (\mathsf{KeyGen_A}, \mathsf{KeyGenB}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* xy-IND-CCA*,* *with* $\mathrm{x}, \mathrm{y} \in \{\mathrm{n}, \mathrm{s}, \mathrm{m}\}$ *if for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}_{\mathsf{sKEM}}^{\text{xy-ind-cca}}(\mathcal{A}) := \left|\Pr\left[\text{xy-IND-CCA}_{\mathsf{sKEM}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2}\right| = \mathsf{negl} \ .$$

These indistinguishability notions range from nn-IND-CCA, which is similar to some kind of IND-CPA security as the adversary has no access to encapsulation or decapsulation oracles, to mm-IND-CCA, which captures strong IND-CCA security for split-KEMs. More generally, all notions are of the form xy-IND-CCA, $\mathrm{x}, \mathrm{y} \in \{\mathrm{n}, \mathrm{s}, \mathrm{m}\}$, where x (resp. y) specifies the number of queries an adversary can make to the decapsulation (resp. encapsulation) oracle (i.e. $\underline{\text{n}}$one, $\underline{\text{s}}$ingle, or $\underline{\text{m}}$any).

On the original split-KEM security. We recall that the advantage of split-KEMs over normal KEMs is that they capture the fact that the party encapsulating can contribute (static) keying material towards the shared key, whereas it is not the case with KEMs, as the encapsulation function only takes the receiving party's public key as input. In particular, this means that KEMs cannot be used for implicit authentication of the encapsulator, unlike split-KEMs. However, we argue that the original xy-IND-CCA definitions for split-KEMs [Bre+21] do not capture implicit authentication either and thus are not suited for their purpose (i.e. building an asynchronous DAKE). In fact, any IND-CPA (resp. IND-CCA) KEM can easily be converted to an (asymmetric) split-KEM satisfying nn-IND-CCA (resp. mm-IND-CCA).

More formally, imagine a setting where Alice and Bob know each other's public key, and Bob wants to implicitly authenticate to Alice using a split-KEM. In addition, we assume a mm-IND-CCA split-KEM sKEM_0 exists (note mm-IND-CCA security is the strongest so this holds for all weaker notions). We first modify sKEM_0 such that on a special ciphertext $\mathsf{ct}^\star$ not in the original ciphertext space, Decaps returns a constant key $K^\star$. Let's call this modified scheme sKEM. We observe that sKEM is still mm-IND-CCA secure as no adversary can break an honestly-generated challenge ciphertext. Now, implicit authentication means that if Alice decapsulates a ciphertext and obtains a key K, then only Bob knows K. However, in our case, any adversary can send $\mathsf{ct}^\star$ to Alice and set their own key to $K^\star$. Both the adversary and Alice will share the same key and implicit authentication does not hold. In a way, xy-IND-CCA security does not prevent forgeries.

UNF-1KCA. This leads us to define our notion of UNF-1KCA security for split-KEMs below which, along with OW-CPA (which can be turned into IND-1BatchCCA), guarantees that only Bob (and obviously Alice) can know the result of Alice's decapsulation on some ciphertext. More precisely, UNF-1KCA ensures that no adversary can forge a valid split-KEM ciphertext for B even knowing a ciphertext that was computed with respect to a public key chosen by the adversary[3], under the condition that the public key used for encapsulation and the known ciphertext are different from the pair made of A's public key and the ciphertext output by the adversary. We also define a security notion called decaps-OW-CPA that will serve as a building block to build UNF-1KCA. The decaps-OW-CPA notion ensures that it is hard for an adversary knowing a ciphertext ct (under an adversarially-chosen public key) to come up with a ciphertext ct' (possibly equal to ct) and a key K' such that the decapsulation of ct' returns K'.

Definition 3.5 (split-KEM UNF-1KCA). *We consider the* UNF-1KCA *game defined in Figure 7. A split-KEM scheme* $\mathsf{sKEM} = (\mathsf{KeyGen}_\mathsf{A}, \mathsf{KeyGenB}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* UNF-1KCA *if for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}_{\mathsf{sKEM}}^{\text{unf-1kca}}(\mathcal{A}) := \Pr\left[\text{UNF-1KCA}_{\mathsf{sKEM}}(\mathcal{A}) \Rightarrow 1\right] = \mathsf{negl} \ .$$

[3]Looking ahead, the fact that the public key is adversarially-chosen will be useful for proving security under key-compromise impersonation attacks for our full protocol.

```
UNF-1KCA_sKEM(A)
1 : $\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\!\!\$\ \mathsf{KeyGenA}(1^\lambda)$
2 : $\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\!\!\$\ \mathsf{KeyGenB}(1^\lambda)$
3 : $\mathsf{pk} \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B})$
4 : $K_\mathsf{B}, \mathsf{ct} \leftarrow\!\!\$\ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
5 : $\mathsf{ct}' \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
6 : if $(\mathsf{ct}, \mathsf{pk}) = (\mathsf{ct}', \mathsf{pk_A})$ : return $0$
7 : $K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
8 : if $K_\mathsf{A} = \perp$ : return $0$
9 : return $1$
```

```
decaps-OW-CPA_sKEM(A)
1 : $b \leftarrow\!\!\$\ \{0,1\}$
2 : $\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\!\!\$\ \mathsf{KeyGenA}(1^\lambda)$
3 : $\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\!\!\$\ \mathsf{KeyGenB}(1^\lambda)$
4 : $\mathsf{pk} \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B})$
5 : $K_\mathsf{B}, \mathsf{ct} \leftarrow\!\!\$\ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
6 : $K'_\mathsf{A}, \mathsf{ct}' \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct})$
7 : $K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
8 : if $K_\mathsf{A} = \perp$ : return $0$
9 : return $1_{K_\mathsf{A} = K'_\mathsf{A}}$
```

Figure 7: Games UNF-1KCA and decaps-OW-CPA.

```
DENY^REAL_sKEM,Sim(A)
1 : $(\mathsf{pk_A}, \mathsf{sk_A}) \leftarrow\!\!\$\ \mathsf{KeyGenA}(1^\lambda)$
2 : $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow\!\!\$\ \mathsf{KeyGenB}(1^\lambda)$
3 : $K, \mathsf{ct} \leftarrow\!\!\$\ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B})$
4 : $b \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{sk_A}, K, \mathsf{ct})$
5 : return $b$
```

```
DENY^SIM_sKEM,Sim(A)
1 : $(\mathsf{pk_A}, \mathsf{sk_A}) \leftarrow\!\!\$\ \mathsf{KeyGenA}(1^\lambda)$
2 : $(\mathsf{pk_B}, \mathsf{sk_B}) \leftarrow\!\!\$\ \mathsf{KeyGenB}(1^\lambda)$
3 : $K, \mathsf{ct} \leftarrow\!\!\$\ \mathsf{Sim}(\mathsf{pk_B}, \mathsf{sk_A})$
4 : $b \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{sk_A}, K, \mathsf{ct})$
5 : return $b$
```

Figure 8: Deniability game.

Definition 3.6 (split-KEM decaps-OW-CPA)**.** *We consider the* decaps-OW-CPA *game defined in Figure 7. A split-KEM scheme* $\mathsf{sKEM} = (\mathsf{KeyGen_A}, \mathsf{KeyGenB}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* decaps-OW-CPA *if for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\text{decaps-ow-cpa}}_{\mathsf{sKEM}}(\mathcal{A}) := \left|\Pr\left[\text{decaps-OW-CPA}_{\mathsf{sKEM}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2}\right| = \mathsf{negl}\ .$$

3.2 Deniability

We finally state the notion of split-KEM deniability we would like to achieve.

Definition 3.7 (Deniability)**.** *We consider the game shown in Figure 8. We say a split-KEM* sKEM *is* DENY *if there exists a simulator* Sim *s.t. for all efficient adversaries* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{\text{deny}}_{\mathsf{sKEM},\mathsf{Sim}}(\mathcal{A}) := \left|\Pr[\text{DENY}^{\mathsf{REAL}}_{\mathsf{sKEM},\mathsf{Sim}}(\mathcal{A}) \Rightarrow 1] - \Pr[\text{DENY}^{\mathsf{SIM}}_{\mathsf{sKEM},\mathsf{Sim}}(\mathcal{A}) \Rightarrow 1]\right| = \mathsf{negl}\ .$$

Informally, the setting considered is the following. Alice and Bob use the split-KEM to establish a shared key (we assume the public keys are only used for this one exchange), and Alice (while following the protocol) wants to frame Bob and prove that he did communicate with her. Therefore, after receiving Bob's ciphertext and deriving the key, Alice gives both public keys, the derived key, the ciphertext and her own secret key to a judge (i.e. the adversary) that must decide whether Bob actually sent the ciphertext that was used to derive the key or not. The scheme is deniable if there is a simulator that, given Alice's view, outputs a ciphertext and a key indistinguishable from the ones output by Bob. We discuss deniability further after introducing our key exchange deniability notion (Section 4.3) and in Section 7.2.

4 Model for DAKE

In this section, we describe our model for deniable authenticated key exchange (DAKE) that we tailor to the semantics and flow of X3DH.

4.1 Syntax

A DAKE DAKE is a tuple of four efficient algorithms $(\mathsf{KeyGen}, \mathsf{Init}, \mathsf{Send}, \mathsf{BatchReceive})$ defined as follows:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$ \ \mathsf{KeyGen}(1^\lambda)$. This function takes as input the security parameter λ and outputs the long-term public/secret key pair of the caller.
- $(\mathsf{st}_i, \mathsf{prek}_i) \leftarrow\!\!\$ \ \mathsf{Init}(\mathsf{sk}_i, \mathsf{role})$. This function takes as inputs a long-term secret key sk_i and a role $\mathsf{role} \in \{\mathsf{sender}, \mathsf{receiver}\}$ and outputs a session state st_i and a prekey bundle prek_i. Init models the creation of key material that will be uploaded to the public key infrastructure by both parties (e.g., a prekey bundle in X3DH). The output values depend only on the public key of party i executing the function.
- $(\mathsf{k}, m) \leftarrow\!\!\$ \ \mathsf{Send}(\mathsf{sk}_i, \mathsf{pk}_j, \mathsf{st}_i, \mathsf{prek}_j)$. This function takes as inputs the secret key of the executing party i, the public key of the intended recipient pk_j, party i's session state st_i and the (claimed) prekey bundle of the intended recipient prek_j, and outputs a key k and a message m.
- $\{k_s\}_s \leftarrow \mathsf{BatchReceive}(\mathsf{sk}_i, \mathsf{st}_i, \{\mathsf{pk}_j, \mathsf{prek}_j, m_j\}_j)$. This function takes as inputs the secret key of the executing party i, an ephemeral state of party i st_i and a vector of size $d \geq 1$ of the form $(\mathsf{pk}_j, \mathsf{prek}_j, m_j)$ for party i's session with the public key of the (claimed) sender pk_j, the (claimed) prekey bundle of party j prek_j and a message m_j, and outputs a vector of d keys $(\mathsf{k}_1, \ldots, \mathsf{k}_d)$, some or all of which may be $\perp$.

Init explicitly captures parties uploading ephemeral keys to a central server in the first protocol step. This contrasts with the formal modelling in some previous works on X3DH-like key exchange [Bre+22; Has+22] that model a three-move key exchange with a single initiator. As Init is independent of keying material from the caller's counterpart, our definition captures so-called *receiver obliviousness* [Has+22] (sometimes *post-specified peers* [CK02]), corresponding to some, but not all, key exchange protocols in the literature.

The most novel part of our primitive is $\mathsf{BatchReceive}$ which in particular captures ephemeral key reuse when uploaded ephemeral keys are exhausted. In the case of key exhaustion, when a party comes back online, they execute $\mathsf{BatchReceive}$ several times (once per ephemeral state st_i), where the number of inputs of the form $(\mathsf{pk}_j, \mathsf{prek}_j, m_j)$ in a given $\mathsf{BatchReceive}$ call corresponds to how many times st_i is re-used. Otherwise, $\mathsf{BatchReceive}$ can be used as in standard AKE with a single value $(\mathsf{pk}_j, \mathsf{prek}_j, m_j)$ as input.

4.2 Security Model

We now describe the security model we consider for our DAKE, which extends existing models in some ways to support $\mathsf{BatchReceive}$.

4.2.1 Parties and sessions

We assume that there are n parties $P_1, \ldots P_n$ (or $1, \ldots, n$) where party P_i (resp. or i) is associated with long-term key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ output by KeyGen. Each party runs one or more *sessions* (sometimes called *oracles* [Bre+21]), where the s-th session of P_i is denoted by π_i^s. Each session π_i^s is associated with the following local fields:

- sid, the session identifier or session id.
- pid, the partner identifier.
- $\mathsf{role} \in \{\perp, \mathsf{sender}, \mathsf{receiver}\}$, the role of P_i.
- $\mathsf{status} \in \{\perp, \mathsf{accept}, \mathsf{reject}\}$, the status of π_i^s.
- k, the session key.
- st, the session state.
- rand, the session randomness.

All fields are initialised to $\perp$ except rand which is initialised to uniform randomness. A session either has role sender or $\mathsf{receiver}$, and its counterpart, its *partner* pid, has the other role; note a $\mathsf{receiver}$ may have several counterparts (capturing ephemeral key reuse).

Fields pid, role, status and rand in session π_i^s are set directly by the challenger, and the rest are (sometimes implicitly) set by the underlying DAKE algorithms called by the challenger. Moreover, in light of the definition of $\mathsf{BatchReceive}$, sid, pid and k are *vectors* for a receiver ($\mathsf{role} = \mathsf{receiver}$); we sometimes write $\vec{\mathsf{sid}}$, $\vec{\mathsf{pid}}$ and $\vec{\mathsf{k}}$ for clarity to indicate this.

Suppose P_i is acting as a receiver. Initially, P_i calls Init, and then eventually calls $\mathsf{BatchReceive}$. Before this point, one or more senders P_j (i.e., parties with $\mathsf{role} = \mathsf{sender}$) may call Init and then Send with respect to the output prek from P_i's Init call (assuming honest message delivery), which output messages of the form m_j. Finally, P_i invokes $\mathsf{BatchReceive}$ with one or more m_j values as input. A party has $\mathsf{status} = \mathsf{accept}$ if and only if $\mathsf{k} \neq \perp$[4], and stores any session state after calling Init and before setting $\mathsf{status} \neq \perp$ due to a Send or $\mathsf{BatchReceive}$ call in st.

4.2.2 Partnering

We define partnering between two sessions to capture security using session identifiers:

Definition 4.1 (Partnering). *For any (i, P_j, s, t), we say that sessions π_i^s and π_j^t are partners if*

1. $\pi_i^s.\mathsf{role} \neq \pi_j^t.\mathsf{role}$.
2. *If* $\pi_i^s.\mathsf{role} = \mathsf{sender}$, *then* $\pi_i^s.\mathsf{pid} = j$ *and* $i \in \pi_j^t.\vec{\mathsf{pid}}$. *If* $\pi_i^s.\mathsf{role} = \mathsf{receiver}$, *then* $j \in \pi_i^s.\vec{\mathsf{pid}}$ *and* $i = \pi_j^t.\mathsf{pid}$.
3. *If* $\pi_i^s.\mathsf{role} = \mathsf{sender}$, *then* $\pi_i^s.\mathsf{sid} \in \pi_j^t.\vec{\mathsf{sid}} \neq \perp$. *If* $\pi_i^s.\mathsf{role} = \mathsf{receiver}$, *then* $\pi_j^t.\mathsf{sid} \in \pi_i^s.\vec{\mathsf{sid}} \neq \perp$.

Looking ahead, this definition ensures that two sessions can only be partners if they both have set $\mathsf{status} = \mathsf{accept}$. Our definition mainly differs from previous work in that there can be many senders (and thus partnered sessions) for a given receiver. Ignoring this aspect, our definition is only slightly different from that of Hashimoto et al. [Has+22] in that we restrict sid to be not equal to $\perp$; this is an artifact of the fact we model 'four-move' key exchange (including prekey uploading).

[4]In particular, $\mathsf{BatchReceive}$ may output several keys; as long as at least one of them is not $\perp$, the calling party accepts.

4.2.3 KIND Security Game

We first define key indistinguishability (KIND) and then define deniability separately. Following previous work, we define a KIND experiment played between a challenger C and adversary $\mathcal{A}$ in text below. The experiment $\mathrm{KIND}^{n}_{\mathsf{DAKE}}$ is parameterised by the DAKE DAKE and integer n, the number of parties (honest or otherwise) in the lifetime of the game's execution. The game is divided into distinct phases defined as follows.

Setup. C first uniformly samples challenge bit $b \in \{0, 1\}$. Then, for each party P_i, C calls $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$ and provides $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ and 1^λ as input to $\mathcal{A}$.

Phase 1. $\mathcal{A}$ adaptively makes any number of the following queries in any order:

- $\mathrm{EXEC}(i, s, \mathsf{prek}, m)$: $\mathcal{A}$ starts or runs the next step of execution in session π^s_i. In each call, C uses randomness tape $\pi^s_i.\mathsf{rand}$ as needed.
  - To start the execution in session π^s_i not previously started, $\mathcal{A}$ calls $\mathrm{EXEC}(i, s, \mathsf{prek}, m)$ with special input $m = (\mathtt{start}, \mathsf{sender}, j)$ (resp. $(\mathtt{start}, \mathsf{receiver}, \vec{j})$) (where $\mathtt{start}$ is defined only in the context of this game) that, if not previously called, sets $\pi^s_i.\mathsf{pid} = j$ (resp. $\pi^s_i.\mathsf{pid} = \vec{j}$) and $\pi^s_i.\mathsf{role} = \mathsf{sender}$ (resp. $\pi^s_i.\mathsf{role} = \mathsf{receiver}$); observe input prek is ignored by C. Then, C invokes $(\mathsf{st}_i, \mathsf{prek}_i) \leftarrow\$ \mathsf{Init}(\mathsf{sk}_i, \mathsf{role})$ and outputs prek_i to $\mathcal{A}$.
  - Given that P_i has started in π^s_i, $\pi^s_i.\mathsf{status} = \perp$ and $\pi^s_i.\mathsf{role} = \mathsf{sender}$, when $\mathcal{A}$ calls $\mathrm{EXEC}(i, s, \mathsf{prek}, \perp)$, C invokes $(\mathsf{k}, m) \leftarrow\$ \mathsf{Send}(\mathsf{sk}_i, \mathsf{pk}_j, \mathsf{st}_i, \mathsf{prek})$ (where $j = \pi^s_i.\mathsf{pid}$), returns output m to $\mathcal{A}$ and sets $\pi^s_i.\mathsf{status}$ to reject (resp. accept) if $\mathsf{k} = \perp$ (resp. $\mathsf{k} \neq \perp$).
  - If $\pi^s_i.\mathsf{role} = \mathsf{receiver}$ and $\pi^s_i.\mathsf{status} = \perp$, when $\mathcal{A}$ calls $\mathrm{EXEC}(i, s, \{s_j, \mathsf{prek}_j, m_j\}_{j \in \vec{j}'})$, C aborts if $\vec{j}' \neq \pi^s_i.\mathsf{pid}$ and otherwise invokes $\mathsf{k} \leftarrow \mathsf{BatchReceive}(\mathsf{sk}_i, \mathsf{st}_i, \{\mathsf{pk}_j, \mathsf{prek}_j, m_j\}_j)$ and outputs to $\mathcal{A}$ $\perp$ if $\mathsf{BatchReceive}$ fails (resp. nothing otherwise) and sets $\pi^s_i.\mathsf{status}$ to reject (resp. accept).
- $\mathrm{LTK}(i)$ outputs sk_i. P_i is hereafter *corrupted.*
- $\mathrm{REGISTER}(\mathsf{pk}_i, i)$ registers a new party P_i for $i > n$ not previously registered, sets their long-term public key to pk_i and distributes pk_i to all other oracles; P_i is immediately marked as *corrupted.*
- $\mathrm{STATE}(i, s)$ outputs $\pi^s_i.\mathsf{st}$, which is hereafter *revealed.*
- $\mathrm{KEY}(i, s, j)$ outputs $\pi^s_i.\mathsf{k}_j$ if $\pi^s_i.\mathsf{role} = \mathsf{receiver}$ and $\pi^s_i.\mathsf{status} \neq \perp$ and otherwise outputs $\pi^s_i.\mathsf{k}$.

Test. When $\mathcal{A}$ decides to move to the next phase, it issues the following query TEST which (if successful) returns either a real or random key:

- $\mathrm{TEST}(i, s, j)$: If $\pi^s_i.\mathsf{status} \neq \mathsf{accept}$, C returns $\perp$. Otherwise:
  - If $\pi^s_i.\mathsf{role} = \mathsf{sender}$, C aborts if $j \neq \pi^s_i.\mathsf{pid}$, and otherwise returns either $\pi^s_i.\mathsf{k}$ if $b = 0$ or a uniformly sampled key k if $b = 1$;
  - If $\pi^s_i.\mathsf{role} = \mathsf{receiver}$, C aborts if $j \notin \pi^s_i.\vec{\mathsf{pid}}$, and otherwise returns either $\pi^s_i.\mathsf{k}_j$ if $b = 0$ or a uniformly sampled key k if $b = 1$.

At this point, π^s_i (which we say is with respect to key j if $\pi^s_i.\mathsf{role} = \mathsf{receiver}$) is said to be the *test session.*

Phase 2. $\mathcal{A}$ adaptively issues queries as in Phase 1.

Guess, freshness and correctness. After Phase 2, $\mathcal{A}$ outputs bit b'. Suppose that $\mathcal{A}$ made query $\text{TEST}(i, s, j)$, i.e., π_i^s is the test session with respect to key j and $j \in \pi_i^s.\mathsf{pid}$ (with equality at least when $\pi_i^s.\mathsf{role} = \mathsf{sender}$). The following *freshness* conditions are checked by C; if any condition *is not satisfied*, C sets b' to a uniform bit (i.e., $\mathcal{A}$ gains no advantage):

1. $\text{KEY}(i, s, j')$ has not been queried, where j' is arbitrary if $\pi_i^s.\mathsf{role} = \mathsf{sender}$ and $j' = j$ if $\pi_i^s.\mathsf{role} = \mathsf{receiver}$.
2. If π_i^s and π_j^t are partners, then $\text{KEY}(j, t, i')$ has not been queried, where i' is arbitrary if $\pi_i^s.\mathsf{role} = \mathsf{sender}$ and $i' = i$ if $\pi_i^s.\mathsf{role} = \mathsf{receiver}$.
3. P_i is not corrupted $\underline{\text{or}}$ $\pi_i^s.\mathsf{st}$ has not been revealed.
4. If π_i^s and π_j^t are partners, then P_j is not corrupted $\underline{\text{or}}$ $\pi_j^t.\mathsf{st}$ has not been revealed.
5. If π_i^s has no partner session, then P_j is not corrupted when $\pi_i^s.\mathsf{status} = \perp$.
6. If π_i^s has no partner session, then if $\pi_i^s.\mathsf{role} = \mathsf{sender}$, for any session π_j^t such that prek_j was both output by $\mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$ and input to Send in π_i^s by C, P_j is not corrupted $\underline{\text{or}}$ $\pi_j^t.\mathsf{st}$ is not revealed.
7. If π_i^s has no partner session, then if $\pi_i^s.\mathsf{role} = \mathsf{receiver}$, for any session π_j^t such that prek_j was both output by $\mathsf{Init}(\mathsf{sk}_j, \mathsf{sender})$ and input to $\mathsf{BatchReceive}$ in π_i^s by C, $\pi_j^t.\mathsf{st}$ is not revealed $\underline{\text{and}}$ $\pi_i^s.\mathsf{st}$ is not revealed.

Then, the following *correctness* conditions are checked by C which, iterating over all relevant parties i, j, k, only consider the subset of sessions corresponding to *honest* protocol runs where $\mathcal{A}$ faithfully follows the protocol specification. If any condition *is satisfied*, C sets $b = b'$ (i.e., $\mathcal{A}$ wins):

1. There exist distinct sessions π_i^s and π_j^t such that $\pi_i^s.\mathsf{role} = \pi_j^t.\mathsf{role}$ and either 1) $\pi_i^s = \mathsf{receiver}$ and $\pi_i^s.\mathsf{sid}_j = \pi_j^t.\mathsf{sid}_i$ or 2) $\pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}$.
2. Assuming $\pi_i^s.\mathsf{role} = \mathsf{receiver}$, there exist sessions π_i^s with respect to key j and π_j^t that are partners such that $\pi_i^s.\mathsf{k}_j \neq \pi_j^t.\mathsf{k}$ (analogously when $\pi_i^s.\mathsf{role} = \mathsf{sender}$).
3. There exist distinct sessions π_i^s, π_j^t and π_k^u such that $\pi_i^s.\mathsf{status} = \pi_j^t.\mathsf{status} = \pi_k^u.\mathsf{status} = \mathsf{accept}$ and $\pi_i^s.\mathsf{sid}_k = \pi_j^t.\mathsf{sid}_k = \pi_k^u.\mathsf{sid}$ (assuming i, j are receivers here but analogously in other cases).

Finally, the game outputs 1 if and only if $b = b'$.

Security is formally captured in Definition 4.2 below.

Definition 4.2 (DAKE key indistinguishability)**.** *We consider the KIND game described above. We say a DAKE* DAKE *is* KIND *if for all efficient adversaries $\mathcal{A}$ and polynomially-bounded n (the total number of parties), we have*

$$\mathsf{Adv}^{\text{kind}}_{\mathsf{DAKE},n}(\mathcal{A}) := \left| \Pr[\text{KIND}^n_{\mathsf{DAKE}}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right| = \mathsf{negl} \ .$$

Discussion. Following previous work, we define freshness conditions to prevent the adversary from mounting trivial attacks. Conditions 1 to 5 correspond exactly to the forward-secure variant of security in [Has+22]. Due to the design of our DAKE K-Waay, we additionally restrict the adversary via conditions 6 and 7. The clauses in these conditions essentially due to the fact

```
$\mathrm{DENY}^{exp}_{\mathsf{DAKE},n,\mathsf{Sim}}(\mathcal{A})$
1 : $b \leftarrow\!\!\$ \{0,1\}$
2 : $L \leftarrow \emptyset$
3 : for $i \in [n]$ :
4 :     $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$ \mathsf{KeyGen}(1^\lambda)$
5 :     $L \leftarrow L \cup \{(\mathsf{pk}_i, \mathsf{sk}_i)\}$
6 : $b' \leftarrow\!\!\$ \mathcal{A}^{\mathrm{CHAL}}(L)$
7 : return $1_{b'=b}$
```

```
$\mathrm{CHAL}(i, j)$
1 : require $i \in [n] \wedge j \in [n]$
2 : $(k, m) \leftarrow (\bot, \bot)$
3 : $(\mathsf{st}_i, \mathsf{prek}_i) \leftarrow\!\!\$ \mathsf{Init}(\mathsf{sk}_i, \mathsf{sender})$
4 : $(\mathsf{st}_j, \mathsf{prek}_j) \leftarrow\!\!\$ \mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$
5 : if $b = 0$ : $(\mathsf{k}, m) \leftarrow\!\!\$ \mathsf{Send}(\mathsf{sk}_i, \mathsf{pk}_j, \mathsf{st}_i, \mathsf{prek}_j)$
6 : else : $(k, m) \leftarrow\!\!\$ \mathsf{Sim}(\mathsf{sk}_j, \mathsf{pk}_i, \mathsf{st}_j, \mathsf{prek}_i, \mathsf{prek}_j)$
7 : $T \leftarrow (\mathsf{prek}_i, \mathsf{prek}_j, m)$
8 : if $exp = \mathsf{true}$ : return $(\mathsf{k}, T, \mathsf{st}_j)$
9 : else : return $(\mathsf{k}, T)$
```

Figure 9: Deniability game.

that in K-Waay the only secret keying material required to call Send is an ephemeral split-KEM secret. For example, suppose that the tested π_i^s is the receiver. Due to the 'symmetric' nature of split-KEM, without these restrictions, revealing $\pi_i^s.\mathsf{st}$ allows the adversary to inject to P_i by simulating Send (akin to a key-compromise impersonation (KCI) attack using P_i's ephemeral state) and trivially distinguish. Consequently, we restrict session state exposure in this case.

Our model does not support randomness exposure or manipulation. As is standard, however, one can employ the NAXOS trick [LLM07] to obtain security given, e.g., randomness but not long-term keying material is exposed. Note also that we do not force Init to be called, e.g., by the sender or senders first or Init to be called by both the sender or senders and receiver before a party calls Send or BatchReceive.

Apart from the fact we make several extensions to typical AKE modelling to capture BatchReceive, the game is closest to that of [Has+22] except that we additionally enforce correctness checks as in [Bre+22]. To capture partnering, we consider partner and key identifiers that may be *vectors* for a receiver, such that several sender sessions may be partnered with a receiver session if, for a given sender session, it partners with a part/component of the receiver session. We do not capture semi-static keys explicitly as in [Bre+22], although in principle they could be captured in Init. Like [Has+22], our game supports message injection, session state exposure (revealing) (unlike [Bre+22]), session key exposure, long-term key exposure (corruption) and adversarial long-term key registration (also considered corruption). During execution, a single challenge test query is made by the adversary that reveals a real or random key output in some session. For BatchReceive which can output several keys, just one of the output keys are tested.

Trivial attacks. We restrict the adversary's behaviour to prevent 'trivial' attacks (e.g. directly revealing the challenge key) by defining *freshness* predicates. Due to our protocol's design, our notion restricts more than the full forward security notion under session state exposure defined by Hashimoto et al. [Has+22]. Our freshness predicates imply weak forward secrecy and implicit authentication given session state exposure is not allowed (enforced in some recent works like [Bad+15; Coh+19]). Brendel et al.'s model provides these guarantees but additionally protect against randomness exposure [Bre+22], whereas we allow exposures on session states under some conditions unlike them.

4.3 Deniability

We next introduce our security notion for a deniable DAKE. To this end, we introduce security game $\mathrm{DENY}^{exp}_{\mathsf{DAKE},\mathsf{Sim}}$ in Figure 9.

Definition 4.3 (DAKE deniability)**.** *We consider the game shown in Figure 9. We say a DAKE* DAKE *is* DENY^{exp} *for* $exp \in \{\mathsf{true}, \mathsf{false}\}$ *if there exists an efficient simulator* Sim *s.t. for all efficient adversaries* $\mathcal{A}$ *and polynomially-bounded* n*, we have*

$$\mathsf{Adv}^{\mathrm{deny}}_{\mathsf{DAKE},\mathsf{Sim},\mathsf{exp}}(\mathcal{A}) := \left|\Pr[\mathrm{DENY}^{exp}_{\mathsf{DAKE},n,\mathsf{Sim}}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}\right| = \mathsf{negl}\ .$$

Our definition captures the following deniability property. Initially, the judge $\mathcal{A}$ is given the long-term keys of all parties. $\mathcal{A}$ then observes honest protocol runs between pairs of parties (via CHAL). Depending on the challenge bit b, either Send or a simulator Sim that takes as input the secret keying material of the receiver trying to frame the sender is executed in each run. Moreover, $\mathcal{A}$ is given the prekey messages independent of b and, if the parameter *exp* is set to true, also the session state of the receiver in each protocol run. The goal of the adversary is to distinguish whether Send or Sim is being called.

Our notion $\mathrm{DENY}^{\mathsf{false}}$ corresponds most closely with that of Brendel et al. [Bre+22] which was also adopted by Cremers et al. [CZ24]. Due to how Brendel et al.'s AKE primitive is defined, they also consider semi-static key pairs which are also given to the adversary. $\mathrm{DENY}^{\mathsf{true}}$ provides stronger deniability, corresponding in practice to a receiver who co-operates with a judge by handing over the entire contents of their device. Although incomparable formally, our DAKE would not be considered deniable under a notion like that of Brendel et al. [Bre+22] since their protocol does not formally model long-term signatures. Note that our definition, like Brendel et al.'s [Bre+22], can be straightforwardly converted to a "simulation-based" notion like Definition 3.7.

Finally, observe that our definition, like that of Brendel et al. [Bre+22] does not consider deniability for the receiver but only for the sender. One could define such a notion, in which the goal is for the judge (adversary) to distinguish between the output of BatchReceive and a simulator Sim that has access to the long-term and ephemeral states of all corresponding senders and is given (honest) ciphertexts output by Send as input. Here, one could argue deniability for K-Waay using the security of the ephemeral KEM and then the KDF. A weaker definition would require the judge to distinguish between the output of Send *and* BatchReceive calls, and the output of a simulator Sim given the senders' states, which is trivial to satisfy (and thus we did not capture it) but is closer to offline semi-honest deniability (whereas the first notion sketched above is more 'online').

5 K-Waay: Post-Quantum X3DH from Split-KEM

5.1 Construction

We present our DAKE K-Waay[5] (K̲ey-e̲xchange W̲ith a̲synchrony, a̲uthentication and peer-deniability̲) in Figure 10.

Each party is associated with a long-term public/secret key pair which in K-Waay comprises of a signature and KEM key pair generated in KeyGen. In Init, ephemeral KEM and split-KEM keys for both parties are generated and the public keys are signed with the long-term signature key.

After initialisation, the sender P_i (sometimes called the initiator) invokes Send that takes the prekey prek_j output by the receiver P_j's Init call as input. After verifying the signature in prek_j, P_i encapsulates to 1) the long-term KEM key of P_j; 2) the ephemeral KEM key contained in prek_j; and 3) the ephemeral split-KEM key contained in prek_j. Note that the split-KEM provides implicit authentication (without it, Send could be simulated without secrets). P_i then combines

[5]Pronounced ké-wè [ke: wɛ]. Dated regionalism from the Neuchâtel area in Switzerland which can mean anything between *yeah* and *of course*. Coincidentally, resembles the name of a brand of raincoats that are reliable, efficient, and that protect against adversarial conditions.

```
KeyGen(1^λ)
1 : // long-term key generation
2 : $(\mathsf{kpk}, \mathsf{ksk}) \leftarrow\$ \mathsf{KeyGen}_{\mathsf{LKEM}}(1^\lambda)$
3 : $(\mathsf{spk}, \mathsf{ssk}) \leftarrow\$ \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$
4 : $\mathsf{pk} \leftarrow (\mathsf{spk}, \mathsf{kpk})$
5 : $\mathsf{sk} \leftarrow (\mathsf{ssk}, \mathsf{ksk})$
6 : return $(\mathsf{pk}, \mathsf{sk})$

Init(sk_i, role)
1 : // prekey generation/upload
2 : if $\mathsf{role} = \mathsf{sender}$ :
3 :     $(\mathsf{espk}_i, \mathsf{essk}_i) \leftarrow\$ \mathsf{KeyGenB}_{\mathsf{sKEM}}(1^\lambda)$
4 :     $\mathsf{ekpk}_i \leftarrow \perp$
5 : else :
6 :     $(\mathsf{espk}_i, \mathsf{essk}_i) \leftarrow\$ \mathsf{KeyGenA}_{\mathsf{sKEM}}(1^\lambda)$
7 :     $(\mathsf{ekpk}_i, \mathsf{eksk}_i) \leftarrow\$ \mathsf{KeyGen}_{\mathsf{EKEM}}(1^\lambda)$
8 : $\sigma_i \leftarrow\$ \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_i.\mathsf{ssk}, (\mathsf{espk}_i, \mathsf{ekpk}_i))$
9 : $\mathsf{prek}_i \leftarrow (\mathsf{espk}_i, \mathsf{ekpk}_i, \sigma_i)$
10 : return $(\mathsf{st}_i = (\mathsf{essk}_i, \mathsf{eksk}_i, \mathsf{prek}_i), \mathsf{prek}_i)$

Send(sk_i, pk_j, st_i, prek_j)
1 : $(\mathsf{essk}_i, \mathsf{eksk}_i, \mathsf{prek}_i) \leftarrow \mathsf{st}_i$
2 : $(\mathsf{espk}_j, \mathsf{ekpk}_j, \sigma_j) \leftarrow \mathsf{prek}_j$
3 : $msg \leftarrow (\mathsf{espk}_j, \mathsf{ekpk}_j)$
4 : require $\mathsf{Vrfy}_{\mathsf{Sig}}(\mathsf{pk}_j.\mathsf{spk}, msg, \sigma_j)$
5 : $(K_\ell, \mathsf{ct}_\ell) \leftarrow\$ \mathsf{Encaps}_{\mathsf{LKEM}}(\mathsf{pk}_j.\mathsf{kpk})$
6 : $(K_k, \mathsf{ct}_k) \leftarrow\$ \mathsf{Encaps}_{\mathsf{EKEM}}(\mathsf{ekpk}_j)$
7 : $(K_s, \mathsf{ct}_s) \leftarrow\$ \mathsf{Encaps}_{\mathsf{sKEM}}(\mathsf{espk}_j, \mathsf{essk}_i)$
8 : $m \leftarrow (\mathsf{ct}_\ell, \mathsf{ct}_k, \mathsf{ct}_s)$
9 : $\mathsf{sid} \leftarrow P_i||P_j||\mathsf{pk}_i||\mathsf{pk}_j||\mathsf{prek}_i||\mathsf{prek}_j||m$
10 : $\mathsf{k} \leftarrow \mathsf{KDF}(K_\ell, K_k, K_s, \mathsf{sid})$
11 : return $(\mathsf{k}, m)$

BatchReceive(sk_i, st_i, S = {pk_j, prek_j, m_j}_j)
1 : $(\mathsf{essk}_i, \mathsf{eksk}_i, \mathsf{prek}_i) \leftarrow \mathsf{st}_i$
2 : $\mathsf{fail} \leftarrow$ false; $\mathsf{k}_j \leftarrow \perp$
3 : for $j : (\mathsf{pk}_j, \mathsf{prek}_j, m_j) \in S$ :
4 :     $(\mathsf{ct}_\ell, \mathsf{ct}_k, \mathsf{ct}_s) \leftarrow m_j$
5 :     $(\mathsf{espk}_j, \mathsf{ekpk}_j, \sigma_j) \leftarrow \mathsf{prek}_j$
6 :     if $\neg\mathsf{Vrfy}_{\mathsf{Sig}}(\mathsf{pk}_j.\mathsf{spk}, (\mathsf{espk}_j, \mathsf{ekpk}_j), \sigma_j)$ :
7 :         $\mathsf{k}_j \leftarrow \perp$
8 :         continue
9 :     $K_\ell \leftarrow \mathsf{Decaps}_{\mathsf{LKEM}}(\mathsf{sk}_i.\mathsf{ksk}, \mathsf{ct}_\ell)$
10 :    $K_k \leftarrow \mathsf{Decaps}_{\mathsf{EKEM}}(\mathsf{eksk}_i, \mathsf{ct}_k)$
11 :    $K_s \leftarrow \mathsf{Decaps}_{\mathsf{sKEM}}(\mathsf{espk}_j, \mathsf{essk}_i, \mathsf{ct}_s)$
12 :    $\mathsf{sid} \leftarrow P_j||P_i||\mathsf{pk}_j||\mathsf{pk}_i||\mathsf{prek}_j||\mathsf{prek}_i||m_j$
13 :    if $K_s = \perp$ : $\mathsf{fail} \leftarrow$ true
14 :    if $(K_\ell = \perp) \vee (K_k = \perp) \vee (K_s = \perp)$ : $\mathsf{k}_j \leftarrow \perp$
15 :    else : $\mathsf{k}_j \leftarrow \mathsf{KDF}(K_\ell, K_k, K_s, \mathsf{sid})$
16 : if fail : return $\perp^{|S|}$
17 : else : return $\{\mathsf{k}_j\}_j$
```

Figure 10: K-Waay: A X3DH-like DAKE from IND-CCA KEMs EKEM and LKEM, SUF-CMA signature scheme Sig and IND-1BatchCCA and UNF-1KCA split-KEM sKEM.

the encapsulated keys using a KDF and outputs the key and its message for P_j consisting of the three encapsulation ciphertexts. Receiving is analogous: receiver P_i verifies P_j's prekey, decapsulates using its three respective secret keys and derives the session key. If P_i's prekeys have run out, it is possible that multiple P_j's have sent using the same prekey prek_i. In that case, P_i decapsulates for all sessions using the same secret keys but aborts if *any* split-KEM decapsulations failed in *any* of the sessions (a signature check failing does not however lead to the receiver aborting). We assume that for a given $\mathsf{BatchReceive}(\mathsf{sk}_i, \mathsf{st}_i, S)$ call, each element of S corresponds to a different party.

5.2 Security

Theorem 1. *Consider* $(1-\delta_{\mathsf{EKEM}})$*-correct* IND-CCA KEM EKEM*,* $(1-\delta_{\mathsf{LKEM}})$*-correct* IND-CCA KEM LKEM*,* $(1-\delta_{\mathsf{Sig}})$*-correct* SUF-CMA *signature scheme* Sig *and* $(1-\delta_{\mathsf{sKEM}})$*-correct* IND-1BatchCCA, UNF-1KCA split-KEM sKEM *and* 3PRF KDF *used to build* $\mathsf{K\text{-}Waay}$ *(Figure 10). Then, we have that for polynomially-bounded n and every efficient adversary $\mathcal{A}$ that makes at most q oracle queries, one can build an adversary $\mathcal{B}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathrm{kind}}_{\mathsf{K\text{-}Waay},n}(\mathcal{A}) \leq \frac{q}{3}\cdot\left(\delta_{\mathsf{Sig}}+\delta_{\mathsf{LKEM}}+\delta_{\mathsf{EKEM}}+\delta_{\mathsf{sKEM}}\right) &+\\ 2q^2\cdot(\epsilon_{\mathsf{EKEM}}+\epsilon_{\mathsf{LKEM}}+2\epsilon_{\mathsf{KDF}}+2\epsilon_{\mathsf{Sig}}) &+\\ q^3\cdot\left(\epsilon_{\mathsf{EKEM}}+\epsilon_{\mathsf{LKEM}}+\epsilon_{\mathsf{sKEM}}+3\epsilon_{\mathsf{KDF}}\right)&,\end{aligned}$$

where $\epsilon_{\mathsf{EKEM}} = \mathsf{Adv}^{\text{ind-cca}}_{\mathsf{EKEM}}(\mathcal{B})$*,* $\epsilon_{\mathsf{LKEM}} = \mathsf{Adv}^{\text{ind-cca}}_{\mathsf{LKEM}}(\mathcal{B})$*,* $\epsilon_{\mathsf{Sig}} = \mathsf{Adv}^{\text{suf-cma}}_{\mathsf{Sig}}(\mathcal{B})$*,* $\epsilon_{\mathsf{sKEM}} = \mathsf{Adv}^{\text{ind-1batchcca}}_{\mathsf{sKEM}}(\mathcal{B}) + \mathsf{Adv}^{\text{unf-1kca}}_{\mathsf{sKEM}}(\mathcal{B})$ *and* $\epsilon_{\mathsf{KDF}} = \mathsf{Adv}^{\text{3prf}}_{\mathsf{KDF}}(\mathcal{B})$.

Proof. Our proof proceeds by constructing sequences of hybrids, which we first summarise. Let Game Γ_1 be exactly the KIND game played with respect to DAKE $\mathsf{K\text{-}Waay}$ (Figure 10). We first transition to Game Γ_2, which differs from Game Γ_1 in that honest protocol runs, all $\mathsf{Vrfy}_{\mathsf{Sig}}$ checks in $\mathsf{BatchReceive}$ calls are removed and Decaps calls are replaced by the output of the Encaps calls in the corresponding Send calls whenever they are consistent. To this end, we invoke the correctness of $\mathsf{K\text{-}Waay}$'s building blocks. Then, we transition to Game Γ_3 in which the challenger immediately outputs the session π_i^s that the adversary makes real-or-random challenge query TEST(i, s, j^*) with respect to. We then partition $\mathcal{A}$'s possible executions of Game Γ_3 into several events.

Suppose π_i^s has a partner session (with respect to key j^* if $\pi_i^s.\mathsf{role} = \mathsf{receiver}$) (event E_p), say π_j^t. Observe that by definition of partnering and construction of the protocol (in particular by definition of sid), it follows that partnered sessions correspond to *honest* protocol runs. Then, considering π_i^s and π_j^t, if the receiver's session state, say $\pi_j^t.\mathsf{st}$, is revealed (event $E_p \wedge E_{c1}$), we reduce to the IND-CCA security of the long-term KEM LKEM, since the freshness conditions imply P_j must not have been corrupted. Otherwise (event $E_p \wedge \neg E_{c1}$), we reduce to the IND-CCA security of the ephemeral KEM EKEM. After both cases, we transition to an unwinnable game by keying KDF with the now uniformly random key output by the respective KEM call, a transition we perform repeatedly and omit from this description hereafter. Otherwise (event $\neg E_p$), we consider whether party P_i in test session π_i^s has the role sender or $\mathsf{receiver}$:

- $\pi_i^s.\mathsf{role} = \mathsf{sender}$ (event $\neg E_p \wedge E_s$): As P_j can only be corrupted after P_i accepts, we first use the SUF-CMA security of Sig to argue that P_i's Send call in the test session must be with honestly-generated input (prek). Then, let E_{c2} be the event that P_j is corrupted. Given $\neg E_p \wedge E_s \wedge E_{c2}$, we reduce to the security of EKEM, since by freshness the state $\pi_j^t.\mathsf{st}$ associated with prek must not have been exposed. Otherwise $(\neg E_p \wedge E_s \wedge \neg E_{c2})$ we reduce to the security of LKEM.

- $\pi_i^s.\mathsf{role} = \mathsf{receiver}$ (event $\neg E_p \wedge \neg E_s$): As above, we first argue using SUF-CMA security that input prek_j used in the test session's $\mathsf{BatchReceive}$ call must have been honestly generated. Then by freshness, we know that neither $\pi_i^s.\mathsf{st}$ nor $\pi_j^t.\mathsf{st}$ associated with prek_j are revealed, in which case we first reduce to the UNF-1KCA security of sKEM to prevent injections on the split-KEM ciphertext, after which we reduce to the IND-1BatchCCA security of sKEM.

Let $\mathsf{Adv}^{\mathrm{gi}}_{\mathsf{DAKE},n}(\mathcal{A})$ be the advantage of adversary $\mathcal{A}$ in winning game Game Γ_i for relevant i which we introduce below. Furthermore, let $\mathsf{Adv}^{\mathrm{gi}}_{\mathsf{DAKE},n}(\mathcal{A}, E)$ be the same advantage except restricted to event E, so in particular if $\mathsf{Adv}^{\mathrm{gi}}_{\mathsf{DAKE},n}(\mathcal{A})$ is of the form $\left|\Pr[X]-\frac{1}{2}\right|$,

$\mathsf{Adv}^{\mathrm{gi}}_{\mathsf{DAKE},n}(\mathcal{A}, E)$ is of the form $\left|\Pr[X \wedge E] - \frac{1}{2}\right|$.

Game Γ_1: This is the original key indistinguishability game.

Game Γ_2: This differs from Game Γ_1 in that, in honest protocol runs, all signature verification calls in BatchReceive calls are removed and the output of Decaps calls are replaced with the output of the corresponding Encaps call in Send. It follows at this point that the three correctness checks in the KIND game evaluate to true. Since for a given $\mathsf{BatchReceive}(\cdot, \cdot, S)$ call there must be $|S|$ corresponding Send and Init calls, there are at most $q/3$ iterations of the **for** loop in BatchReceive (counting over all such calls in a given execution of Game Γ_1). It then follows from a standard hybrid argument and the correctness of Sig, LKEM, EKEM and sKEM that:

$$\mathsf{Adv}^{\mathrm{g1}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \frac{q}{3} \cdot \left(\delta_{\mathsf{Sig}} + \delta_{\mathsf{LKEM}} + \delta_{\mathsf{EKEM}} + \delta_{\mathsf{sKEM}}\right) .$$

Game Γ_3: This differs from Game Γ_3 in that the challenger immediately outputs the session π_i^s that the adversary $\mathcal{A}$ calls $\mathrm{TEST}(i, s, j^*)$ with respect to. Noting that there are at most q such possible sessions and applying a standard argument, it follows that:

$$\mathsf{Adv}^{\mathrm{g2}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq q \cdot \mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}) .$$

Case 1: Test session π_i^s is partnered (Game Γ_{3a} and Game Γ_{3b}):

Game $\Gamma_{3a.1}$: Let E_p be the event that test session π_i^s has a partner, say π_j^t. Let E_{c1} be the event that the ephemeral state st of the receiver (in π_i^s and π_j^t) is revealed. Games Game $\Gamma_{3a.i}$ are defined given $E_p \wedge E_{c1}$. Game $\Gamma_{3a.1}$ differs from Game Γ_3 in that the game initially outputs π_j^t, the partner of π_i^s (observe that $j = j^*$ where j^* is defined in the previous hop), as well as a bit indicting whether π_i^s is the sender or receiver. By the same reasoning as above, we have

$$\mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}, E_p \wedge E_{c1}) \leq 2q \cdot \mathsf{Adv}^{\mathrm{g3a.1}}_{\mathsf{DAKE},n}(\mathcal{A}) .$$

Game $\Gamma_{3a.2}$: Game $\Gamma_{3a.2}$ differs from Game $\Gamma_{3a.1}$ in that the output key K in the call to LKEM.Encaps and the corresponding LKEM.Decaps call or calls (which are guaranteed to exist given E_p, and K is identical by definition of Game Γ_2) made in the test and partner sessions with respect to the receiver's public key and secret key, respectively, are replaced with a key k uniformly sampled by the challenger. Observe that since E_{c1} holds, by freshness, P_j cannot be corrupted, and thus we reduce to the security of LKEM.

Let $\mathcal{A}'$ be a IND-CCA adversary who simulates for Game $\Gamma_{3a.1}$/Game $\Gamma_{3a.2}$ adversary $\mathcal{A}$ as follows. Let pk be the IND-CCA challenge public key, (ct^*, K^*) be the challenge ciphertext and key respectively.

In the Setup phase, $\mathcal{A}'$ uniformly samples bit b_{sim}, calls $(\mathsf{pk}_\ell, \mathsf{sk}_\ell) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$ locally for $\ell \neq k$ where k is the sender, sets $\mathsf{pk}_k \leftarrow \mathsf{pk}$, and returns $\{\mathsf{pk}_1, \ldots \mathsf{pk}_n\}$ and 1^λ to $\mathcal{A}$. Observe here (and later for Game $\Gamma_{3b.2}$) that, since E_p holds, we have matching sid values for test session π_i^s and partner π_j^t. Note by construction of sid, the presence of substring $\mathsf{prek}_i||\mathsf{prek}_j$ and m in the common value sid implies that Send must have been called honestly in π_i^s and also in BatchReceive for tuple $(\mathsf{pk}_j, \mathsf{prek}_j, m_j)$ in π_j^t for E_p to hold. Thus, we do not need to consider injections in the test session itself (although we have to in general in the BatchReceive call).

Before proceeding, we argue that $\mathcal{A}'$ can simulate on behalf of parties with a maliciously-registered long-term key locally, which applies here and in the rest of the proof. Since π_i^s is partnered, as argued above, π_i^s and π_j^t correspond to honest (completed) executions, and so neither P_i and P_j can be malicious. For unpartnered sessions, since Send and BatchReceive cannot be called by the game, the test session π_i^s cannot be corrupted itself (since testing requires $\pi_i^s.\mathsf{status} \neq \bot$), and otherwise condition 5 restricts the non-tested party P_j from being corrupted, thus precluding its key from being registered maliciously. Finally, computation involving messages or prekey bundles from maliciously-registered parties does not require any secret material not already known to $\mathcal{A}'$.

In Phase 1, when $\mathcal{A}$ calls $\mathrm{EXEC}(k', \cdot, \cdot, \cdot)$ where k' corresponds to the sender in π_i^s and π_j^t and the challenger is supposed to invoke Send, $\mathcal{A}'$ replaces the call to $\mathsf{Encaps}_{\mathsf{LKEM}}$ with the output (ct^*, K^*), and otherwise simulates locally. When $\mathcal{A}$ calls $\mathrm{EXEC}(k, \cdot, \cdot, \cdot)$ corresponding to the receiver in π_i^s and π_j^t and the challenger is supposed to invoke $\mathsf{BatchReceive}$, $\mathcal{A}'$ replaces the output of the relevant $\mathsf{Decaps}_{\mathsf{LKEM}}$ calls corresponding to either i or j, depending on who is the receiver, with K^* and the output of other calls $\mathsf{Decaps}_{\mathsf{LKEM}}$ with the output obtained from $\mathrm{DEC}(\cdot)$; $\mathcal{A}'$ otherwise simulates locally.

In the Test phase, i.e. when $\mathcal{A}$ calls $\mathrm{TEST}(i, s, j)$, $\mathcal{A}'$ simulates with respect to bit b_{sim}. $\mathcal{A}'$ then simulates Phase 2 as above and the rest of the game locally, ultimately outputting the same bit as $\mathcal{A}$; observe that $\mathcal{A}'$ can efficiently evaluate the freshness conditions. Since $\mathcal{A}'$ perfectly simulates Game $\Gamma_{3a.1}$ when playing with respect to challenge bit 0 and Game $\Gamma_{3a.2}$ when it is 1, it follows that:

$$\mathsf{Adv}^{\mathrm{g3a.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3a.2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{ind-cca}}_{\mathsf{LKEM}}(\mathcal{A}') \ .$$

Game $\Gamma_{3a.3}$: This differs from Game $\Gamma_{3a.2}$ in that, for the test and partner sessions, the call to KDF made in Send and the corresponding calls made in $\mathsf{BatchReceive}$ with respect to ciphertext ct_ℓ output by Send are replaced with uniformly sampled keys. Let $\mathcal{A}'$ be a PRF adversary playing with respect to KDF keyed in its first argument simulating for Game $\Gamma_{3a.2}$/Game $\Gamma_{3a.3}$ adversary $\mathcal{A}$ as follows. $\mathcal{A}'$ simulates locally all calls except the Send and $\mathsf{BatchReceive}$ calls made in π_i^s and π_j^t, where it replaces the relevant calls $\mathsf{KDF}(K_\ell, K_k, K_s, \mathsf{sid})$ with the call $\mathrm{PRF}(K_k, K_s, \mathsf{sid})$. Since K_ℓ is uniform (by definition of Game $\Gamma_{3a.2}$) and, by definition of freshness, K_ℓ is not revealed to $\mathcal{A}$, the simulation is perfect and we have:

$$\mathsf{Adv}^{\mathrm{g3a.2}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3a.3}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{3prf}}_{\mathsf{KDF}}(\mathcal{A}') \ .$$

Finally, we have $\mathsf{Adv}^{\mathrm{g3a.3}}_{\mathsf{DAKE},n}(\mathcal{A}) = 0$ since the output of TEST is identical regardless of the challenge bit and it is not otherwise used by the challenger or leaked to the adversary.

Game $\Gamma_{3b.1}$: We now consider the case when $E_p \wedge \neg E_{c1}$, i.e. the case where the receiver's session state st in π_i^s and π_j^t is not revealed. Game $\Gamma_{3b.1}$ differs from Game Γ_3 in that the game initially outputs π_j^t, the partner of π_i^s, as well as a bit indicating whether π_i^s is the sender or receiver. Since Game $\Gamma_{3b.1}$ is exactly Game $\Gamma_{3a.1}$, we have

$$\mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}, E_p \wedge \neg E_{c1}) \leq q \cdot \mathsf{Adv}^{\mathrm{g3b.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \ .$$

Game $\Gamma_{3b.2}$: In Game $\Gamma_{3b.2}$, the output of $\mathsf{Encaps}_{\mathsf{EKEM}}$ and the corresponding $\mathsf{Decaps}_{\mathsf{EKEM}}$ call or calls in the test session are replaced with a uniformly random key k. IND-CCA adversary $\mathcal{A}'$ simulates for Game $\Gamma_{3b.1}$/Game $\Gamma_{3b.2}$ adversary $\mathcal{A}$ as follows. $\mathcal{A}'$ follows the same broad approach as the adversary defined in the hop between Game $\Gamma_{3a.1}$ and Game $\Gamma_{3a.2}$. In particular, $\mathcal{A}'$ simulates the receiver in their session's call to Init except it uses the IND-CCA challenge public key pk, replaces the output of EKEM in the test session Encaps and the corresponding Decaps calls with the challenge ciphertext and key and replaces other Decaps calls with calls to oracle DEC. By the same reasoning as before, it follows that:

$$\mathsf{Adv}^{\mathrm{g3b.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3b.2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{ind-cca}}_{\mathsf{EKEM}}(\mathcal{A}') \ .$$

Game $\Gamma_{3b.3}$: This replaces the relevant outputs of KDF in the test session with a uniformly random key. As in Game $\Gamma_{3a.3}$, this game is now unwinnable, i.e. $\mathsf{Adv}^{\mathrm{g3b.3}}_{\mathsf{DAKE},n}(\mathcal{A}) = 0$. As before, we reduce to the security of KDF, except now we key KDF in the PRF game with the second argument K_k. We then arrive at:

$$\mathsf{Adv}^{\mathrm{g3b.2}}_{\mathsf{DAKE},n}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{g3b.3}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{3prf}}_{\mathsf{KDF}}(\mathcal{A}') \ .$$

Case 2: Test session π_i^s is unpartnered and $\pi_i^s.\mathsf{role} = \mathsf{sender}$ (Game Γ_{3c}):

Game $\Gamma_{3c.1}$: Let $\pi_i^s.\mathsf{pid} = j$ and E_s be the event that $\pi_i^s.\mathsf{role} = \mathsf{sender}$. Game $\Gamma_{3c.1}$ differs from Game Γ_3 in that the challenger immediately outputs j. By a standard argument, we have:

$$\mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}, \neg E_p \wedge E_s) \le q \cdot \mathsf{Adv}^{\mathrm{g3c.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \ .$$

Game $\Gamma_{3c.2}$: This differs from Game $\Gamma_{3c.1}$ in that the challenger aborts if the call $\mathsf{Send}(\mathsf{sk}_i, \mathsf{pk}_j, \mathsf{st}_i, \mathsf{prek}_j)$ in the test session is such that prek_j was not previously output by a call to $\mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$. Note that by freshness condition 5 that P_j must not be corrupted until after $\pi_i^s.\mathsf{status}$ is changed from $\perp$, which, by definition of E_s, means until after it is set to accept. In order for Send to accept on input $\mathsf{prek}_j = (\mathsf{espk}_j, \mathsf{ekpk}_j, \sigma_j)$ not previously output by $\mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$ (and thus for the game to abort), $\mathcal{A}$ needs to find a different prek_j such that $\mathsf{VrfySig}(\mathsf{pk}_j.\mathsf{spk}, (\mathsf{espk}_j, \mathsf{ekpk}_j), \sigma_j)$ (by construction of Send). Using this observation, we reduce to the SUF-CMA security of Sig.

Let $\mathcal{A}'$ be a SUF-CMA adversary simulating for Game $\Gamma_{3c.1}$ adversary $\mathcal{A}$. Let pk be the SUF-CMA challenge public key. In the Setup phase, $\mathcal{A}'$ sets $\mathsf{pk}_j = \mathsf{pk}$ and otherwise simulates locally. In particular, unlike in previous hops, $\mathcal{A}'$ also samples the random Game $\Gamma_{3c.1}$ bit. In each subsequent phase, for each call $\mathrm{EXEC}(j, u, \cdot, m)$ such that $m = (\mathtt{start}, \mathsf{role}, \cdot)$, $\mathcal{A}'$ replaces the $\mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_j.\mathsf{ssk}, (\mathsf{espk}_j, \mathsf{ekpk}_j))$ call in $\mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$ by a call to $\mathrm{SIGN}((\mathsf{espk}_j, \mathsf{ekpk}_j))$, and otherwise simulates the call locally. When the challenger calls $\mathsf{Send}(\cdot, \mathsf{pk}_j, \cdot, \mathsf{prek}_j)$ where $\mathsf{prek}_j = (\mathsf{espk}_j, \mathsf{ekpk}_j, \sigma_j)$, $\mathcal{A}'$ checks whether 1) $(\mathsf{espk}_j, \mathsf{ekpk}_j)$ was previously queried to SIGN which output σ_j and 2) $\mathsf{Vrfy}_{\mathsf{Sig}}(\mathsf{pk}, (\mathsf{espk}_j, \mathsf{ekpk}_j), \sigma_j) = 1$. Given 1) and 2) both hold, $\mathcal{A}'$ returns $(m, \sigma) = ((\mathsf{espk}_j, \mathsf{ekpk}_j), \sigma_j)$ to its challenger. $\mathcal{A}'$ otherwise simulates locally, aborting if $\mathcal{A}$ outputs a bit. The simulation is perfect and it follows that:

$$\mathsf{Adv}^{\mathrm{g3c.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{g3c.2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{suf-cma}}_{\mathsf{Sig}}(\mathcal{A}') \ .$$

Game $\Gamma_{3c.3}$: In Game $\Gamma_{3c.3}$, the challenger initially outputs π_j^t, where π_j^t is the session that prek is output by $\mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$ and input to the Send call in test session π_i^s. By a standard failure event argument, we have:

$$\mathsf{Adv}^{\mathrm{g3c.2}}_{\mathsf{DAKE},n}(\mathcal{A}) \le q \cdot \mathsf{Adv}^{\mathrm{g3c.3}}_{\mathsf{DAKE},n}(\mathcal{A}) \ .$$

Game $\Gamma_{3c.4a.1}$: Let E_{c2} be the event that P_j is corrupted. We construct hybrid sequence Game $\Gamma_{3c.4a}$ (resp. Game $\Gamma_{3c.4b}$) to deal with the case that E_{c2} holds (resp. does not hold). Game $\Gamma_{3c.4a.1}$ differs from Game $\Gamma_{3c.3}$ in that the output of $\mathsf{Encaps}_{\mathsf{EKEM}}$ in the Send call in test session π_i^s and of the (possible) corresponding $\mathsf{Decaps}_{\mathsf{EKEM}}$ calls in π_j^t are replaced with uniformly random output. By freshness, P_j's session state $\pi_j^t.\mathsf{st}$ associated with prek input to the test Send call is not revealed.

IND-CCA adversary $\mathcal{A}'$ simulates for Game $\Gamma_{3c.4a.1}$ adversary $\mathcal{A}$ as follows. Let $(\mathsf{pk}, k, \mathsf{ct})$ the challenge public key, key and corresponding ciphertext (respectively) of $\mathcal{A}'$. $\mathcal{A}'$ embeds pk in session π_j^t by replacing the public key output by $\mathsf{KeyGen}_{\mathsf{EKEM}}$ in $\mathsf{Init}(\mathsf{sk}_j, \mathsf{receiver})$ with pk, which outputs prek_j. Upon prek_j being input to Send in the test session, $\mathcal{A}'$ replaces the output of $\mathsf{Encaps}_{\mathsf{EKEM}}$ with (k, ct). When the challenger calls $\mathsf{BatchReceive}(\mathsf{sk}_j, \cdot, \{\cdot, \cdot, m_{j'} = (\cdot, \mathsf{ct}', \cdot)\}_{j'})$ in session π_j^t, if $\mathsf{ct}' = \mathsf{ct}$, $\mathcal{A}'$ replaces the output of $\mathsf{Decaps}_{\mathsf{EKEM}}$ with k; else, $\mathcal{A}'$ replaces the call

$\mathsf{Decaps}_{\mathsf{EKEM}}(\cdot, \mathsf{ct}')$ with the call DEC(ct'). $\mathcal{A}'$ otherwise simulates locally and outputs the same bit as $\mathcal{A}$. By similar reasons to before, we have:

$$\mathsf{Adv}^{\mathrm{g3c.3}}_{\mathsf{DAKE},n}(\mathcal{A}, E_{c2}) \leq \mathsf{Adv}^{\mathrm{g3c.4a.1}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{ind-cca}}_{\mathsf{EKEM}}(\mathcal{A}') \ .$$

Game $\Gamma_{3c.4a.2}$: This replaces the output of KDF in the Send and $\mathsf{BatchReceive}$ calls as before in the test session and π_j^t with uniformly random keys. By the exact same argument as for Game $\Gamma_{3b.3}$, we have $\mathsf{Adv}^{\mathrm{g3c.4a.2}}_{\mathsf{DAKE},n}(\mathcal{A}) = 0$ and

$$\mathsf{Adv}^{\mathrm{g3c.4a.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3c.4a.2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{3prf}}_{\mathsf{KDF}}(\mathcal{A}') \ .$$

Game $\Gamma_{3c.4b.1}$: We assume $\neg E_{c2}$, i.e. that P_j is not corrupted. We reduce to the IND-CCA security of LKEM. The reduction follows the same high-level strategy as previous hops (embedding the challenge pk in pk_j and the challenge in the test Send call and possibly the corresponding $\mathsf{BatchReceive}$ call), noting that non-challenge $\mathsf{Decaps}_{\mathsf{LKEM}}(\mathsf{sk}_j, \cdot)$ queries are replaced with calls to DEC. We then have:

$$\mathsf{Adv}^{\mathrm{g3c.3}}_{\mathsf{DAKE},n}(\mathcal{A}, \neg E_{c2}) \leq \mathsf{Adv}^{\mathrm{g3c.4b.1}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{ind-cca}}_{\mathsf{LKEM}}(\mathcal{A}') \ .$$

Game $\Gamma_{3c.4b.2}$: As in Game $\Gamma_{3c.4a.2}$, this replaces the output of KDF in the Send and $\mathsf{BatchReceive}$ calls in π_i^s and π_j^t with a uniformly random key. As argued several times above, it follows that $\mathsf{Adv}^{\mathrm{g3c.4b.2}}_{\mathsf{DAKE},n}(\mathcal{A}) = 0$ and

$$\mathsf{Adv}^{\mathrm{g3c.4b.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3c.4b.2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{3prf}}_{\mathsf{KDF}}(\mathcal{A}') \ .$$

Case 3: Test session π_i^s is unpartnered and $\pi_i^s.\mathsf{role} = \mathsf{receiver}$ (Game Γ_{3d}):

Game $\Gamma_{3d.1}$: Game $\Gamma_{3d.1}$ differs from Game Γ_3 in that the challenger immediately outputs j, the third argument in $\mathcal{A}$'s TEST(i, s, j) call. As for Game $\Gamma_{3c.1}$, we have

$$\mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}, \neg E_s) \leq q \cdot \mathsf{Adv}^{\mathrm{g3d.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \ .$$

Game $\Gamma_{3d.2}$: This differs from Game $\Gamma_{3d.1}$ in that the challenger aborts if the call $\mathsf{BatchReceive}(\mathsf{sk}_i, \mathsf{st}_i, \{\mathsf{pk}_{j'}, \mathsf{prek}_{j'}, m\}_{j'})$ in the test session is such that prek_j was not previously output by a call to $\mathsf{Init}(\mathsf{sk}_j, \mathsf{sender})$. As in Game $\Gamma_{3c.2}$, P_j must not be corrupted until after $\pi_i^s.\mathsf{status}$ is set to accept. By reducing to SUF-CMA security essentially as in Game $\Gamma_{3c.2}$, it follows that:

$$\mathsf{Adv}^{\mathrm{g3d.1}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3d.2}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{suf-cma}}_{\mathsf{Sig}}(\mathcal{A}) \ .$$

Game $\Gamma_{3d.3}$: This differs from Game $\Gamma_{3d.2}$ in that the challenger initially outputs π_j^t, the session that generated prek_j which formed part of the input to $\mathsf{BatchReceive}$ in the test session π_i^s. By a standard argument we have:

$$\mathsf{Adv}^{\mathrm{g3d.2}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq q \cdot \mathsf{Adv}^{\mathrm{g3d.3}}_{\mathsf{DAKE},n}(\mathcal{A}) \ .$$

Game $\Gamma_{3d.4}$: This differs from Game $\Gamma_{3d.3}$ in that the challenger aborts if the $\mathsf{Send}(\mathsf{sk}_j, \mathsf{pk}_i, \mathsf{st}_j, \mathsf{prek}_i)$ call in session π_j^t (if it exists) and the relevant component in the $\mathsf{BatchReceive}$ call in the test session π_i^s were not both with respect to honestly generated split-KEM keying material (namely, an honestly generated split-KEM public key from prek_i and prek_j from the previous hop) and the same split-KEM ciphertext. By freshness, neither of the two ephemeral states $\pi_i^s.\mathsf{st}$ and $\pi_j^t.\mathsf{st}$ are revealed. Consequently, we reduce to the UNF-1KCA security of split-KEM sKEM.

UNF-1KCA adversary $\mathcal{A}'$ simulates for Game $\Gamma_{3d.3}$/Game $\Gamma_{3d.4}$ adversary $\mathcal{A}$ as follows. Let $(\mathsf{pk_A}, \mathsf{pk_B})$ be the two challenge public keys given to $\mathcal{A}'$. In the $\mathsf{Init}(\mathsf{sk}_i, \mathsf{receiver})$ call in session π_i^s, $\mathcal{A}'$ simulates except replaces the call to $\mathsf{KeyGenA_{sKEM}}$ by $\mathsf{pk_A}$. Similarly, in the $\mathsf{Init}(\mathsf{sk}_j, \mathsf{sender})$ call in session π_j^t, $\mathcal{A}'$ replaces $\mathsf{KeyGenB_{sKEM}}$ by $\mathsf{pk_B}$. In the $\mathsf{Send}(..., \mathsf{prek})$ call in session π_j^t where $\mathsf{prek} = (\mathsf{espk}, ...)$, $\mathcal{A}'$ outputs espk to its UNF-1KCA challenger, receives $(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}, K_B)$ from its challenger, and replaces the call to $\mathsf{Encaps_{sKEM}}$ with tuple (ct, K_B). Finally, when the $\mathsf{BatchReceive}(\mathsf{sk}_i, \cdot, \{\cdot, \mathsf{prek}_{j'}, m = (\cdot, \cdot, \mathsf{ct}_s)\}_{j'})$ call in test session π_i^s is made, $\mathcal{A}'$ outputs ct_s corresponding to $j' = j$ to its challenger. As the simulation is perfect and the probability that $\mathcal{A}'$ wins is exactly the probability that 1) $(\mathsf{ct}, \mathsf{pk_A}) \neq (\mathsf{ct}_s, \mathsf{pk})$ and 2) relevant $\mathsf{Decaps_{sKEM}}$ call in $\mathsf{BatchReceive}$ outputs $k \neq \perp$, it follows by a standard failure event argument that:

$$\mathsf{Adv}^{\mathrm{g3d.3}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3d.4}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{unf-1kca}}_{\mathsf{sKEM}}(\mathcal{A}') \ .$$

<u>Game $\Gamma_{3d.5}$</u>: This differs from Game $\Gamma_{3d.4}$ in that the output k of the relevant test session split-KEM decapsulation and the corresponding encapsulation (if it exists) are both replaced by a uniformly random key. Note that by definition of Game $\Gamma_{3d.4}$, $\mathcal{A}$ can only input an honestly generated split-KEM ciphertext to the $\mathsf{BatchReceive}$ call in the test session from P_j and that the split-KEM public key in P_j's corresponding Send call (if it exists) must be honestly generated. We therefore reduce to the IND-1BatchCCA security of sKEM. We embed the IND-1BatchCCA keys $\mathsf{pk_A}$, $\mathsf{pk_B}$ in the simulation as in the previous hop. When $\mathcal{A}$ queries $\mathrm{EXEC}(i, s, S = \{s_{j'}, \mathsf{prek}_{j'}, m_{j'}\}_{j'})$, $\mathcal{A}'$ replaces all $\mathsf{Decaps_{sKEM}}$ calls involving $\mathsf{sk_A}$ *except* the call corresponding to the test session by the output of its query to oracle BatchDec, replaces this final $\mathsf{Decaps_{sKEM}}$ call with the IND-1BatchCCA challenge key and otherwise simulates locally. It follows that:

$$\mathsf{Adv}^{\mathrm{g3d.4}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3d.5}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{ind-1batchcca}}_{\mathsf{sKEM}}(\mathcal{A}') \ .$$

<u>Game $\Gamma_{3d.6}$</u>: This game replaces the relevant invocation of KDF in the test session's $\mathsf{BatchReceive}$ call by a uniformly random value. Note as usual that $\mathsf{Adv}^{\mathrm{g3d.6}}_{\mathsf{DAKE},n}(\mathcal{A}) = 0$. By keying KDF in its third argument as a PRF and a standard argument it follows that:

$$\mathsf{Adv}^{\mathrm{g3d.5}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3d.6}}_{\mathsf{DAKE},n}(\mathcal{A}) + \mathsf{Adv}^{\mathrm{3prf}}_{\mathsf{KDF}}(\mathcal{A}') \ .$$

Finally note that by the triangle inequality, we have, among other inequalities:

$$\mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}, E_p) + \mathsf{Adv}^{\mathrm{g3}}_{\mathsf{DAKE},n}(\mathcal{A}, \neg E_p) \ .$$

The result follows using this observation and by combining the sequences of hybrids together in a standard way. □

Theorem 2. *Consider deniable split-KEM* sKEM *with simulator* $\mathsf{Sim_{sKEM}}$ *used to build* $\mathsf{K\text{-}Waay}$ *(Figure 10). Then, we have that for every efficient adversary $\mathcal{A}$ that makes at most q oracle queries, there exists an efficient* Sim *s.t. one can build an adversary $\mathcal{B}$ such that for* $\exp \in \{\mathsf{true}, \mathsf{false}\}$ *we have:*

$$\mathsf{Adv}^{\mathrm{deny}}_{\mathsf{K\text{-}Waay},\mathsf{Sim},\exp}(\mathcal{A}) \leq q \cdot \mathsf{Adv}^{\mathrm{deny}}_{\mathsf{sKEM},\mathsf{Sim_{sKEM}}}(\mathcal{B}) \ .$$

Proof. We construct a sequence of hybrids and reduce to the deniability of sKEM (i.e. $\mathrm{DENY}_{\mathsf{sKEM},\mathsf{Sim_{sKEM}}}$ security) in each step. Before this, we define the simulator Sim that we use in the proof, which uses the simulator $\mathsf{Sim_{sKEM}}$ as a subroutine.

Observe in $\mathsf{K\text{-}Waay}$ that, given an honestly generated prek_j, any party with knowledge only of public keying material can simulate all steps in Send except for the $\mathsf{Encaps_{sKEM}}$ call which requires sender P_i's secret key. Thus, our simulator Sim (Figure 11) simulates these steps and

```
Sim(sk_j, pk_i, st_j, prek_i, prek_j)
1 : $(\mathsf{essk}_j, \mathsf{eksk}_j, \mathsf{prek}_j) \leftarrow \mathsf{st}_j$
2 : $(\mathsf{espk}_i, \mathsf{ekpk}_i, \sigma_i) \leftarrow \mathsf{prek}_i$
3 : $(\mathsf{espk}_j, \mathsf{ekpk}_j, \sigma_j) \leftarrow \mathsf{prek}_j$
4 : $\mathsf{spk} \leftarrow \mathsf{GetPK}(\mathsf{sk}_j.\mathsf{ssk})$
5 : require $\mathsf{Vrfy}_{\mathsf{Sig}}(\mathsf{spk}, (\mathsf{espk}_j, \mathsf{ekpk}_j), \sigma_j) = 1$
6 : $(K_\ell, \mathsf{ct}_\ell) \leftarrow\!\$\ \mathsf{Encaps}_{\mathsf{LKEM}}(\mathsf{pk}_j.\mathsf{kpk})$
7 : $(K_k, \mathsf{ct}_k) \leftarrow\!\$\ \mathsf{Encaps}_{\mathsf{EKEM}}(\mathsf{ekpk}_j)$
8 : $(K_s, \mathsf{ct}_s) \leftarrow\!\$\ \mathsf{Sim}_{\mathsf{sKEM}}(\mathsf{espk}_i, \mathsf{essk}_j)$
9 : $m \leftarrow (\mathsf{ct}_\ell, \mathsf{ct}_k, \mathsf{ct}_s)$
10 : $\mathsf{sid} \leftarrow P_i||P_j||\mathsf{pk}_i||\mathsf{pk}_j||\mathsf{prek}_i||\mathsf{prek}_j||m$
11 : $\mathsf{k} \leftarrow \mathsf{KDF}(K_\ell, K_k, K_s, \mathsf{sid})$
12 : return $(\mathsf{k}, m)$
```

Figure 11: Simulator Sim for the deniability game where we assume function GetPK(sk) that takes a signature secret key as input and outputs the corresponding public key.

since it takes the receiver's key sk_j as input it can also invoke the deniability simulator $\mathsf{Sim}_{\mathsf{sKEM}}$ to complete the call.

Let Γ_0 be the DAKE DENY game instantiated with K-Waay. For $i \in [q]$, let Γ_i be the same as Γ_{i-1} except that in the i-th CHAL call, the call to Send is replaced with a call to Sim. Note that the steps executed in Send only differ in that it calls $\mathsf{Encaps}_{\mathsf{sKEM}}$ rather than $\mathsf{Sim}_{\mathsf{sKEM}}$.

For $i \in [q]$, let $\mathcal{B}$ be a split-KEM DENY adversary with input $(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{B}, K, \mathsf{ct})$ from its challenger playing $\text{DENY}^{\mathsf{REAL}}$ given $\mathcal{A}$ is playing Γ_{i-1} and $\text{DENY}^{\mathsf{SIM}}$ if it is playing Γ_i. $\mathcal{B}'$ locally simulates long-term public key generation and the first $i-1$ calls to CHAL. When $\mathcal{A}$ makes their i-th call to CHAL, $\mathcal{B}$ simulates CHAL until it reaches the **if** statement except that it replaces the output of calls KeyGenA/KeyGenB calls in Init calls with $\mathsf{pk}_\mathsf{A}/\mathsf{pk}_\mathsf{B}$. Then, instead of executing the **if/else** block in CHAL, $\mathcal{B}$ simulates Sim except that it replaces the output of the call to $\mathsf{Sim}_{\mathsf{sKEM}}$ with (K, ct). $\mathcal{B}$ then simulates locally, and returns (k, T, st_r) (where st_r contains sk_B) if $exp = \mathsf{true}$ and returns (k, T) otherwise. $\mathcal{B}$ continues simulating locally and finally outputs the same bit as $\mathcal{A}$. Noting that DAKE deniability game $\text{DENY}_{\mathsf{K\text{-}Waay},\mathsf{Sim}}$ considers only honest executions of K-Waay, it follows that the simulation is perfect, and so by $\text{DENY}_{\mathsf{sKEM}}$ security we have

$$\left|\mathsf{Adv}^{\Gamma_{i-1}}_{\mathsf{DAKE}}(\mathcal{A}) - \mathsf{Adv}^{\Gamma_i}_{\mathsf{DAKE}}(\mathcal{A})\right| \leq \mathsf{Adv}^{\text{deny}}_{\mathsf{sKEM},\mathsf{Sim}_{\mathsf{sKEM}}}(\mathcal{B}) \ .$$

By application of the triangle inequality and telescoping sums:

$$\left|\mathsf{Adv}^{\Gamma_0}_{\mathsf{DAKE}}(\mathcal{A}) - \mathsf{Adv}^{\Gamma_q}_{\mathsf{DAKE}}(\mathcal{A})\right| \leq q \cdot \mathsf{Adv}^{\text{deny}}_{\mathsf{sKEM},\mathsf{Sim}_{\mathsf{sKEM}}}(\mathcal{B}) \ .$$

To complete the proof, observe that $\mathsf{Adv}^{\Gamma_q}_{\mathsf{DAKE},n}(\mathcal{A}) = 0$ since CHAL behaves identically independent of challenge bit b. □

6 Deniable Split-KEM from Lattices

In this section we build an efficient deniable split-KEM under the hardness of LWE. We start by introducing briefly several concepts of lattice-based cryptography that we use to design the scheme.

6.1 Lattice Toolbox

L_∞ and L_α norms. We start by recalling what the L_∞ and L_α norms over $\mathbb{Z}_q$ are. For an element w in $\mathbb{Z}_q$, we write $\|w\|_\infty$ to mean $|\langle w\rangle_q|$. Then, we define the L_∞ and L_α norms for $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ over $\mathbb{Z}_q$ as follows:

$$\|\mathbf{w}\|_\infty = \max_{j\in[n]} \|w_j\|_\infty, \quad \|\mathbf{w}\|_\alpha = \sqrt[\alpha]{\|w_1\|_\infty^\alpha + \ldots + \|w_n\|_\infty^\alpha}.$$

By default, $\|\mathbf{w}\| := \|\mathbf{w}\|_2$.

Probability distributions. For a finite set S, we define $\mathcal{U}(S)$ to be the uniform distribution on S. We will also use the binomial distribution Bin_1 which is defined as: $\mathsf{Bin}_1(-1) = \mathsf{Bin}_1(1) = 1/4$ and $\mathsf{Bin}_1(0) = 1/2$.

Rounding functions. Given two parameters q and $B < \log q - 1$, we define the rounding function $\lfloor\cdot\rceil_{q,2}$ and the cross-rounding function $\langle\cdot\rangle_{q,B}$ as follows:

$$\lfloor\cdot\rceil_{q,B} : v \mapsto \left\lfloor \frac{2^B}{q} \cdot v \right\rceil \bmod 2^B, \quad \langle\cdot\rangle_{q,B} : v \mapsto \left\lfloor \frac{2^{B+1}}{q} \cdot v \right\rfloor \bmod 2 \ ,$$

for any $v \in \mathbb{Z}_q$.

Reconciliation function. We recall the (generalized) reconciliation mechanism from Bos et al. and Peikert [Bos+16; Pei14], which for every approximate agreement in $\mathbb{Z}_q$ allows extracting shared bits. We refer the reader to the aforementioned works for more details. Let q be a positive integer. Let B be the number of bits we want to extract from one coefficient in $\mathbb{Z}_q$ so that $B < \log q - 1$. Now, for any $v \in \mathbb{Z}_q$, which is represented as an integer in $[0, q)$, we define the following functions.

Definition 6.1 (Randomised doubling function (dbl)). *For any $v \in \mathbb{Z}_q$, we define $\mathsf{dbl}(\cdot)$ as follows:*

$$\mathsf{dbl}(v) : v \mapsto 2v - e, \quad e \leftarrow\$ \ \mathsf{Bin}_1$$

Then, we have the following property which comes from [Bos+16, Claim 3.1].

Lemma 1. *Let q be odd. If $v \in \mathbb{Z}_q$ is uniformly random and $\bar{v} \leftarrow\$ \ \mathsf{dbl}(v) \in \mathbb{Z}_{2q}$, then $\lfloor\bar{v}\rceil_{2q,B}$ is uniformly random given $\langle\bar{v}\rangle_{2q,B}$.*

Now, we are ready to define the reconciliation function $\mathsf{Rec} : \mathbb{Z}_{2q} \times \mathbb{Z}_2 \to \mathbb{Z}_{2^B}$.

Definition 6.2 (Reconciliation function (Rec)). *For any $w \in \mathbb{Z}_{2q}$ and bit $b \in \{0,1\}$, let v be the closest element to $w \in \mathbb{Z}_{2q}$ s.t. $\langle v\rangle_{2q,B} = b$. Then, we define Rec as*

$$\mathsf{Rec}(w, b) := \lfloor v\rceil_{2q,B} \ .$$

The next result gives an important property of the reconciliation function Rec, as described by Peikert [Pei14, Section 3.2].

Lemma 2. *Let q be odd and $\bar{v} \leftarrow\$ \ \mathsf{dbl}(v)$. If $|v - w| \le \lfloor \frac{q}{2^{B+2}} \rfloor$ then*

$$\mathsf{Rec}(2w, \langle\bar{v}\rangle_{2q,B}) = \lfloor\bar{v}\rceil_{2q,B}.$$

Finally, we define the $\mathsf{HelpRec} : \mathbb{Z}_q \mapsto \{0,1\}$ function as follows:

Definition 6.3 ($\mathsf{HelpRec}$ function). *On any input $v \in \mathbb{Z}_q$,*

$$\mathsf{HelpRec}(v) := \langle\bar{v}\rangle_{2q,B}, \quad \textit{where } \bar{v} \leftarrow \mathsf{dbl}(v) \ .$$

All the functions above can be naturally generalized to take as input vectors and matrices over $\mathbb{Z}_q$ by applying the function to each of the coefficients.

Learning-with-Errors. The security of our lattice constructions relies on the Learning-with-Errors (LWE) problem introduced by Regev [Reg05]. In this paper we will consider the case where both the secret and error coefficients come from a probability distribution over $\mathbb{Z}$.

Definition 6.4 ($\mathsf{LWE}_{n,m,\chi,q}$)**.** *Let $n, m \in \mathbb{N}$ and χ be a probability distribution over $\mathbb{Z}$. The* LWE *problem asks the adversary $\mathcal{A}$ to distinguish between the following two cases:*

1. $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q)$ *for* $\mathbf{A} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n\times m})$*, a secret* $\mathbf{s} \leftarrow\$ \chi^m$ *and error* $\mathbf{e} \leftarrow\$ \chi^n$*,*
2. $(\mathbf{A}, \mathbf{t}) \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n\times m}) \times \mathcal{U}(\mathbb{Z}_q^n)$*.*

We say that an efficient algorithm $\mathcal{A}$ solves $\mathsf{LWE}_{n,m,\chi,q}$ if it can distinguish between the two distributions above with non-negligible probability.

6.2 Extended-LWE

Our proof of deniability for the split-KEM will involve a new security assumption, which we call the Extended-LWE problem (ELWE). Intuitively, it is similar to the plain LWE problem, but the adversary is now also given random linear combinations of the secrets and errors.

Definition 6.5 ($\mathsf{ELWE}_{n,m,\bar{n},\chi,q}$)**.** *Let $n, m \in \mathbb{N}$ and χ be a probability distribution over $\mathbb{Z}$. The* ELWE *problem asks the adversary $\mathcal{A}$ to distinguish between the following two cases:*

1. $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q, \mathbf{Z}, \mathbf{W}, \mathbf{Z}\mathbf{s} + \mathbf{W}\mathbf{e} \bmod q)$ *for* $\mathbf{A} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n\times m})$*, secret* $\mathbf{s} \leftarrow\$ \chi^m$*, error* $\mathbf{e} \leftarrow\$ \chi^n$*, and* $(\mathbf{Z}, \mathbf{W}) \leftarrow\$ \chi^{\bar{n}\times m} \times \chi^{\bar{n}\times n}$*,*
2. $(\mathbf{A}, \mathbf{t}, \mathbf{Z}, \mathbf{W}, \mathbf{Z}\mathbf{s} + \mathbf{W}\mathbf{e} \bmod q)$ *for* $\mathbf{A} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n\times m})$*,* $\mathbf{t} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^n)$*, secret* $\mathbf{s} \leftarrow\$ \chi^m$*, error* $\mathbf{e} \leftarrow\$ \chi^n$*, and* $(\mathbf{Z}, \mathbf{W}) \leftarrow\$ \chi^{\bar{n}\times m} \times \chi^{\bar{n}\times n}$*.*

We say that an efficient algorithm $\mathcal{A}$ solves $\mathsf{ELWE}_{n,m,\bar{n},k,\chi,q}$ if it can distinguish between the two distributions above with non-negligible probability.

This problem is a natural generalization of the Extended-LWE problem by Alperin-Sheriff and Peikert [AP12], where now $(\mathbf{Z}, \mathbf{W})$ are matrices and not just vectors. Here, we also simplify the definition and assume that the coefficients of $\mathbf{Z}$ and $\mathbf{W}$ come from the same distribution χ as the secrets and errors.

We show in the following theorem that the hardness of this newly introduced ELWE problem reduces to the hardness of LWE.

Theorem 3. *Let q be an odd prime and χ be symmetric around 0. If there is an efficient adversary $\mathcal{A}$ which wins $\mathsf{ELWE}_{n,m,\bar{n},\chi,q}$ with probability ε, then there also exists an efficient adversary $\mathcal{B}$ which wins $\mathsf{LWE}_{n+m,m,\chi,q}$ with probability at least $\delta_{\mathsf{elwe}} \cdot \varepsilon - \mathsf{negl}(n)$ where*

$$\delta_{\mathsf{elwe}} := \Pr\left[\mathbf{Z}(\mathbf{e} - \mathbf{d}) = \mathbf{0} \pmod q : \mathbf{Z} \leftarrow\$ \chi^{\bar{n}\times(n+m)}, \mathbf{e}, \mathbf{d} \leftarrow\$ \chi^{n+m}\right]. \tag{1}$$

Proof. We prove the statement by introducing a sequence of LWE-type games Γ_i. We start with $\Gamma^1 := \mathsf{ELWE}_{n,m,\bar{n},\chi,q}$ and give an efficient reduction from Γ_i to Γ_{i+1}. In the end, we finish with $\mathsf{LWE}_{n+m,m,\chi,q}$. We denote $\mathsf{Adv}_i(\mathcal{A})$ to be the probability that $\mathcal{A}$ wins Γ_i. Then, the proof follows by the composition of the constant number of efficient reductions.

Game Γ_1: This is the standard $\mathsf{ELWE}_{n,m,\bar{n},\chi,q}$ game. The adversary $\mathcal{A}$ wins this game with probability ε.

Game Γ_2: Here, we consider the ELWE-type game where the secret vector is uniformly random. Namely, the challenger samples the public $\mathbf{A} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^{(n+m)\times m})$, secret $\mathbf{s} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^m)$, error $\mathbf{e} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^{n+m})$ as well as the hint matrix $\mathbf{Z} \leftarrow\!\!\$\, \chi^{\bar{n}\times(n+m)}$. Then it flips a bit $b \leftarrow\!\!\$\, \mathcal{U}(\{0,1\})$. If $b = 0$ then the challenger computes

$$\mathbf{t} := \mathbf{A}\mathbf{s} + \mathbf{e}$$

and otherwise it samples $\mathbf{t} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^{n+m})$. The challenger outputs $(\mathbf{A}, \mathbf{t}, \mathbf{Z}, \mathbf{Z}\mathbf{e})$.

Lemma 3. *For every efficient adversary $\mathcal{A}$, there is an efficient adversary $\mathcal{B}$ such that* $\mathsf{Adv}_2(\mathcal{B}) \geq \mathsf{Adv}_1(\mathcal{A}) - \mathsf{negl}(n)$.

Proof. The reduction follows similarly as in the one by Applebaum et al. [App+09]. Suppose the algorithm $\mathcal{B}$ is given a tuple $(\mathbf{A}, \mathbf{t}, \mathbf{Z}, \mathbf{h})$ from Γ_2. With probability at most $1/q^{(n+m)-m-1} \leq 1/q^{n-1}$, matrix $\mathbf{A}$ is not full-rank. Let us exclude that case and assume without loss of generality that we can write

$$\mathbf{A} := \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \end{bmatrix}, \quad \mathbf{Z} := \begin{bmatrix} \mathbf{Z}_0 & \mathbf{Z}_1 \end{bmatrix}, \quad \text{and} \quad \mathbf{t} := \begin{bmatrix} \mathbf{t}_0 \\ \mathbf{t}_1 \end{bmatrix}$$

where $\mathbf{A}_1 \in \mathbb{Z}_q^{n\times m}$ and the matrix $\mathbf{A}_0 \in \mathbb{Z}_q^{m\times m}$, which contains the first m rows of $\mathbf{A}$, is invertible. Thus, define $\mathbf{A}' := \mathbf{A}_1\mathbf{A}_0^{-1} \in \mathbb{Z}_q^{n\times m}$, and $\mathbf{t}' := \mathbf{A}'\mathbf{t}_0 - \mathbf{t}_1 \in \mathbb{Z}_q^n$. Then, it runs $\mathcal{A}$ on input

$$\big(\mathbf{A}', \mathbf{t}', \mathbf{Z}_0, -\mathbf{Z}_1, \mathbf{h}\big)$$

and returns what $\mathcal{A}$ outputs.

Suppose that $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ where $\mathbf{s} \in \mathbb{Z}_q^m$ and $\mathbf{e} := (\mathbf{e}_0, \mathbf{e}_1) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^n$. Then

$$\mathbf{t}' = \mathbf{A}'\mathbf{t}_0 - \mathbf{t}_1 = \mathbf{A}_1\mathbf{A}_0^{-1}(\mathbf{A}_0\mathbf{s} + \mathbf{e}_0) - (\mathbf{A}_1\mathbf{s} + \mathbf{e}_1) = \mathbf{A}'\mathbf{e}_0 - \mathbf{e}_1$$

which is a valid LWE instance since χ is symmetric around 0. Also, if $\mathbf{A}$ is uniformly random among all nonsingular matrices, then $\mathbf{A}'$ and $\mathbf{B}'$ are statistically close to uniformly random matrices over $\mathbb{Z}_q$. As for the hints, note that

$$\mathbf{h} = \mathbf{Z}_0\mathbf{e}_0 + \mathbf{Z}_1\mathbf{e}_1 = \mathbf{Z}_0\mathbf{e}_0 + (-\mathbf{Z}_1)(-\mathbf{e}_1),$$

so $\mathbf{h}$ is a well-formed hint for Γ_1.

On the other hand, if $\mathbf{t}$ is uniformly random, then so is $\mathbf{t}'$. It can be argued similarly as before that all the other components follow the distribution for $b = 1$. □

Game Γ_3: We consider the knapsack version of ELWE. Here, the challenger samples the public $\mathbf{G} :=\leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^{n\times(n+m)})$, secret $\mathbf{e} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^{n+m})$ and the hint matrix $\mathbf{Z} \leftarrow\!\!\$\, \chi^{\bar{n}\times(n+m)}$. Then it flips a bit $b \leftarrow\!\!\$\, \mathcal{U}(\{0,1\})$. If $b = 0$ then the challenger computes $\mathbf{t} := \mathbf{G}\mathbf{e}$, and otherwise it samples $\mathbf{t} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^n)$. Finally, the challenger outputs $(\mathbf{G}, \mathbf{t}, \mathbf{Z}, \mathbf{Z}\mathbf{e})$.

Lemma 4. *For every efficient adversary $\mathcal{A}$, there is an efficient adversary $\mathcal{B}$ such that* $\mathsf{Adv}_3(\mathcal{B}) \geq \mathsf{Adv}_2(\mathcal{A}) - \mathsf{negl}(n)$.

Proof. The reduction is similar to the proof of Micciancio and Mol [MM11, Lemma 4.9]. Suppose the algorithm $\mathcal{B}$ is given a tuple $(\mathbf{G}, \mathbf{t}, \mathbf{Z}, \mathbf{h})$ from Γ_3. Then, $\mathcal{B}$ can construct a randomized matrix $\mathbf{A} \in \mathbb{Z}_q^{(n+m)\times m}$ whose columns generate the kernel of $\mathbf{G}$. In particular, if $\mathbf{G}$ is uniformly random, then so are $(\mathbf{A}, \mathbf{B})$, up to the constraint that they are nonsingular. Then, $\mathcal{B}$ computes any solution $\mathbf{r}$ such that $\mathbf{G}\mathbf{r} = \mathbf{t}$. Finally, it samples a uniformly random $\mathbf{s} \leftarrow\!\!\$\, \mathcal{U}(\mathbb{Z}_q^m)$ and runs $\mathcal{A}$ on input

$$(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{r}, \mathbf{Z}, \mathbf{h})$$

and returns what $\mathcal{A}$ outputs.

Suppose that $\mathbf{Ge} = \mathbf{t} = \mathbf{Gr}$. By definition of the matrix $\mathbf{A}$, $\mathbf{G}(\mathbf{r} - \mathbf{e}) = \mathbf{0}$ implies that there exists some vector $\mathbf{x} \in \mathbb{Z}_q^m$ such that $\mathbf{r} - \mathbf{e} = \mathbf{Ax}$. Thus,

$$\mathbf{As} + \mathbf{r} = \mathbf{A}(\mathbf{s} + \mathbf{x}) + \mathbf{e}$$

which is a valid LWE instance since $\mathbf{s}+\mathbf{x}$ is still uniformly random over $\mathbb{Z}_q^m$. As for the hints, we still have $\mathbf{h} = \mathbf{Ze}$ and thus $\mathcal{B}$ correctly simulates Γ_2 for $b = 0$. The case $b = 1$ follows by arguing that $\mathbf{t}$ is uniformly random and if $\mathbf{G}$ is nonsingular then $\mathbf{r}$ must be uniformly random. □

Game Γ_4: This game is a plain knapsack LWE problem. The challenger samples the public $\mathbf{G} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n\times(n+m)})$ and a secret $\mathbf{e} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n+m})$. Then it flips a bit $b \leftarrow\$ \mathcal{U}(\{0,1\})$. If $b = 0$ then the challenger computes $\mathbf{t} := \mathbf{Ge}$, and otherwise it samples $\mathbf{t} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^n)$. Finally, the challenger outputs $(\mathbf{G}, \mathbf{t})$.

Lemma 5. *For every efficient adversary $\mathcal{A}$, there is an efficient adversary $\mathcal{B}$ such that $\mathsf{Adv}_4(\mathcal{B}) \geq \delta_{\mathsf{elwe}} \cdot \mathsf{Adv}_3(\mathcal{A})$.*

Proof. We follow the proof strategy from [AP12, Theorem 1]. Suppose the algorithm $\mathcal{B}$ is given a tuple $(\mathbf{G}_0, \mathbf{G}_1, \mathbf{t})$ from Γ_4. Then, it samples $\mathbf{Z} \leftarrow\$ \chi^{\bar{n}\times(n+m)}$, $\mathbf{d} \leftarrow\$ \chi^{n+m}$ and a matrix $\mathbf{V} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n\times\bar{n}})$. Further, it sets

$$\mathbf{G}' := \mathbf{G} - \mathbf{VZ} \quad \text{and} \quad \mathbf{t}' := \mathbf{t} - \mathbf{VZd}.$$

Finally, it runs $\mathcal{A}$ on input

$$\big(\mathbf{G}', \mathbf{t}', \mathbf{Z}, \mathbf{Zd}\big)$$

and returns what $\mathcal{A}$ outputs.

Clearly, if $\mathbf{G}$ (resp. $\mathbf{t}$) is uniformly random then so is $\mathbf{G}'$ (resp. $\mathbf{t}'$). Hence, the case $b = 1$ follows directly. Suppose $b = 0$ and thus $\mathbf{t} = \mathbf{Ge}$. Then, we have

$$\mathbf{t}' = \mathbf{t} - \mathbf{VZd} = \mathbf{Ge} - \mathbf{VZd} = \mathbf{G}'\mathbf{e} + \mathbf{V}(\mathbf{Ze} - \mathbf{Zd}).$$

Hence, if $\mathbf{Ze} = \mathbf{Zd}$ then $([\mathbf{G}_0' \ \mathbf{G}_1], \mathbf{t}', \mathbf{Z}, \mathbf{Zd})$ is indeed a valid knapsack ELWE tuple. This happens exactly with probability at most δ_{elwe} by definition. Otherwise, $\mathbf{V}(\mathbf{Ze} - \mathbf{Zd})$ is a uniformly random vector over $\mathbb{Z}_q$, and so is $\mathbf{t}'$. Thus, the tuple output by $\mathcal{B}$ follows the case $b = 1$ for Γ_3. The statement now follows by simple calculation. □

Game Γ_5: Here, we consider the plain LWE game. Recall that the challenger samples the public $\mathbf{A} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{(n+m)\times m})$, secret $\mathbf{s} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^m)$, error $\mathbf{e} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n+m})$. Then it flips a bit $b \leftarrow\$ \mathcal{U}(\{0,1\})$. If $b = 0$ then the challenger computes $\mathbf{t} := \mathbf{As} + \mathbf{e}$, and otherwise it samples $\mathbf{t} \leftarrow\$ \mathcal{U}(\mathbb{Z}_q^{n+m})$. At the end, the challenger outputs $(\mathbf{A}, \mathbf{t})$.

Lemma 6. *For every efficient adversary $\mathcal{A}$, there is an efficient adversary $\mathcal{B}$ such that $\mathsf{Adv}_5(\mathcal{B}) \geq \mathsf{Adv}_4(\mathcal{A}) - \mathsf{negl}(n)$.*

Proof. The reduction is identical to the one of Micciancio and Mol [MM11, Lemma 4.8] which we recall for completeness. Suppose the algorithm $\mathcal{B}$ is given a tuple $(\mathbf{A}, \mathbf{t})$ from Γ_5. If $\mathbf{A}$ is full-rank, then $\mathcal{B}$ can construct a (randomized) matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times(n+m)}$ whose rows generate all the vectors $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{A} = \mathbf{0}$. Also, if $\mathbf{A}$ is chosen at random among all full-rank matrices, then $\mathbf{G}$ is also distributed statistically close to a uniformly random. Then, $\mathcal{B}$ outputs $(\mathbf{G}, \mathbf{Gt})$ to $\mathcal{A}$ and returns what $\mathcal{A}$ outputs.

Suppose $b = 0$ and $\mathbf{t} = \mathbf{As} + \mathbf{e}$. Then $\mathbf{Gt} = \mathbf{GAs} + \mathbf{e} = \mathbf{Ge}$, which is the correct instance of Γ_4 for $b = 0$. On the other hand, if $\mathbf{t}$ is uniformly random, then so is $\mathbf{Gt}$. □

The statement of the theorem now follows by combining all the previous lemmas using reduction composition. □

```
$\mathsf{KeyGenA}(1^\lambda)$
1 : $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\$\ \chi(\mathbb{Z}_q^{n\times\bar{n}})$
2 : $\mathbf{F}_\mathsf{A} \leftarrow\!\$\ \chi^{\bar{n}\times\bar{n}}$
3 : $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
4 : $\mathsf{pk}_\mathsf{A} \leftarrow (\mathbf{A}, \mathbf{B}_\mathsf{A})$
5 : $\mathsf{sk}_\mathsf{A} \leftarrow (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A})$
6 : return $(\mathsf{pk}_\mathsf{A}, \mathsf{sk}_\mathsf{A})$
```

```
$\mathsf{Encaps}(\mathsf{pk}_\mathsf{A} = (\mathbf{A}, \mathbf{B}_\mathsf{A}), \mathsf{sk}_\mathsf{B} = (\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B}, \mathbf{F}_\mathsf{B}))$
1 : // We assume B encapsulates
2 : $\mathbf{E}_\mathsf{B} \leftarrow\!\$\ \chi^{\bar{n}\times\bar{n}}$
3 : $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}$
4 : $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
5 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
6 : return $(\mathbf{K}, \mathsf{ct})$
```

```
$\mathsf{KeyGenB}(1^\lambda)$
1 : $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\$\ \chi^{\bar{n}\times n}$
2 : $\mathbf{F}_\mathsf{B} \leftarrow\!\$\ \chi^{\bar{n}\times\bar{n}}$
3 : $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
4 : $\mathsf{pk}_\mathsf{B} \leftarrow (\mathbf{A}, \mathbf{B}_\mathsf{B})$
5 : $\mathsf{sk}_\mathsf{B} \leftarrow (\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B}, \mathbf{F}_\mathsf{B})$
6 : return $(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{B})$
```

```
$\mathsf{Decaps}(\mathsf{pk}_\mathsf{B} = (\mathbf{A}, \mathbf{B}_\mathsf{B}), \mathsf{sk}_\mathsf{A} = (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A}), \mathsf{ct})$
1 : $\mathbf{V}' \leftarrow \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{F}_\mathsf{A}$
2 : $\mathbf{K}' \leftarrow \mathsf{Rec}(2\mathbf{V}', \mathsf{ct})$
3 : return $\mathbf{K}'$
```

Figure 12: Our variant of FrodoKEX [Bos+16] expressed as a split-KEM. The matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ is assumed to be a public parameter and sampled uniformly at random.

6.3 Construction

We can now present our Frodo-inspired [Bos+16] split-KEM, which we call FrodoKEX+. The scheme is given in Figure 12. The key generation works as follows. The public key pk_A for party A is a pair $(\mathbf{A}, \mathbf{B}_\mathsf{A})$, where $\mathbf{A}$ is a uniformly random matrix over $\mathbb{Z}_q$ given as a public parameter, and $\mathbf{B}_\mathsf{A} := \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$ where $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\$\ \chi^{n\times\bar{n}}$. The secret key becomes a pair $\mathsf{sk}_\mathsf{A} = (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A})$. Similarly, the public key pk_B for party B is a pair $(\mathbf{A}, \mathbf{B}_\mathsf{B})$, where $\mathbf{B}_\mathsf{B} := \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$, while the secret key is $\mathsf{sk}_\mathsf{B} = (\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B})$, where $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\$\ \chi^{\bar{n}\times n}$.

Then, B samples a matrix $\mathbf{E}_\mathsf{B} \leftarrow\!\$\ \chi^{\bar{n}\times\bar{n}}$ and computes the matrix $\mathbf{V} := \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}$. Next, it computes $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$ and $\mathbf{K} \leftarrow \mathsf{Rec}(\mathbf{V}, \mathsf{ct})$. Then, B outputs ct. Then, party A decapsulates as follows: given $(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathsf{ct})$, it computes $\mathbf{V}' = \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{F}_\mathsf{A}$ and $\mathbf{K}' = \mathsf{Rec}(2\mathbf{V}', \mathsf{ct})$. Finally, A returns the key $\mathbf{K}'$.

We note that the construction can easily be made symmetric, in the sense that A could encapsulate using B's public key by changing the order of matrices when multiplying in Encaps such that the dimensions match. Then, Decaps can be modified similarly such that B can decapsulate the resulting ciphertext.

6.4 Security Analysis

Lemma 7 (Correctness)**.** *Let χ be a symmetric distribution around 0 and δ_corr be the following probability:*

$$\Pr\left[|\langle \mathbf{s}, \mathbf{d}\rangle + e + f| > \frac{q}{2^{B+2}} : \mathbf{s}, \mathbf{d} \leftarrow\!\$\ \chi^{2n}, e, f \leftarrow\!\$\ \chi\right]. \tag{2}$$

Then, sKEM *defined in Fig. 12 is $(\bar{n}^2\delta_\mathsf{corr})$-correct.*

Proof. Suppose $(\mathsf{pk}_\mathsf{A}, \mathsf{sk}_\mathsf{A}) \leftarrow\!\$\ \mathsf{KeyGenA}(1^\lambda)$ and $(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{B}) \leftarrow\!\$\ \mathsf{KeyGenB}(1^\lambda)$, and let

$$(\mathbf{K}, \mathsf{ct}) \leftarrow\!\$\ \mathsf{Encaps}(\mathsf{pk}_\mathsf{A}, \mathsf{sk}_\mathsf{B}) \quad \text{and} \quad \mathbf{K}' \leftarrow\!\$\ \mathsf{Decaps}(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathsf{ct}).$$

We want to prove that $\mathbf{K} = \mathbf{K}'$. By definition of encapsulation, we know that $\mathbf{K} = \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$ where $\mathsf{ct} = \mathsf{HelpRec}(\mathbf{V})$ and

$$\mathbf{V} = \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B} = \mathbf{S}_\mathsf{B}\mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{S}_\mathsf{B}\mathbf{D}_\mathsf{A} + \mathbf{E}_\mathsf{B}.$$

Thus, by Lemma 2, $\mathbf{K} = \lfloor \mathbf{V} \rceil_{2q,2^B}$. On the other hand,

$$\mathbf{V}' = \mathbf{B}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{F}_{\mathsf{A}} = \mathbf{S}_{\mathsf{B}}\mathbf{A}\mathbf{S}_{\mathsf{A}} + \mathbf{D}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{F}_{\mathsf{A}}$$

which implies that $\mathbf{V} - \mathbf{V}' = \mathbf{S}_{\mathsf{B}}\mathbf{D}_{\mathsf{A}} + \mathbf{E}_{\mathsf{B}} - \mathbf{D}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} - \mathbf{F}_{\mathsf{A}}$. If $\|\mathbf{V} - \mathbf{V}'\|_\infty < \frac{q}{2^{B+2}}$ then by Lemma 2 we must have

$$\mathbf{K}' = \mathsf{Rec}(2\mathbf{V}', \mathsf{HelpRec}(\mathbf{V})) = \lfloor \mathbf{V} \rceil_{2q,2^B} = \mathbf{K}$$

so correctness holds. Now, using the fact that χ is symmetric around 0, the probability $\|\mathbf{V} - \mathbf{V}'\|_\infty > \frac{q}{2^{B+2}}$ can be upper-bounded using the union bound as follows:

$$\Pr\left[\|\mathbf{S}_{\mathsf{B}}\mathbf{D}_{\mathsf{A}} + \mathbf{E}_{\mathsf{B}} - \mathbf{D}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} - \mathbf{F}_{\mathsf{A}}\|_\infty > \frac{q}{2^{B+2}}\right] \leq \bar{n}^2 \cdot \Pr\left[|\mathbf{s}_0^T\mathbf{d}_0 + \mathbf{s}_1^T\mathbf{d}_1 + e + f| > \frac{q}{2^{B+2}}\right]$$

where $\mathbf{s}_0, \mathbf{s}_1, \mathbf{d}_0, \mathbf{d}_1 \leftarrow\!\!\$\ \chi^n$ and $e, f \leftarrow\!\!\$\ \chi$. This concludes the proof. $\square$

6.4.1 OW-CPA Security

Next, we focus on proving OW-CPA security.

Lemma 8 (OW-CPA Security)**.** *Let $\bar{n} = O(\lambda)$ and χ be a symmetric distribution over $[-\gamma, \gamma]$ for any $\gamma > 0$. Then, under the $\mathsf{LWE}_{n,n,\chi,q}$ and $\mathsf{LWE}_{n+\bar{n},n,\chi,q}$ assumptions, for every efficient adversary $\mathcal{A}$, the probability of $\mathcal{A}$ winning the OW-CPA game is at most $2^{-B\bar{n}^2} + \mathsf{negl}(\lambda)$.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the OW-CPA game. We prove the statement using the hybrid games described explicitly in Fig. 13. In each game Γ_i, we define ε_i to be the probability that the efficient adversary $\mathcal{A}$ wins the security game.

Game Γ_1: This is the standard OW-CPA game.

Game Γ_2: Instead of computing $\mathbf{B}_{\mathsf{A}} \leftarrow \mathbf{A}\mathbf{S}_{\mathsf{A}} + \mathbf{D}_{\mathsf{A}}$, the experiment samples $\mathbf{B}_{\mathsf{A}} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n\times\bar{n}})$. One can naturally build an efficient adversary, which can solve the $\mathsf{LWE}_{n,n,\chi,q}$ problem with probability at least $\frac{1}{\bar{n}}|\varepsilon_2 - \varepsilon_1|$. Hence, we deduce that this probability is negligible.

Game Γ_3: Here, the experiment computes the values $\mathbf{B}_{\mathsf{B}}$ and $\mathbf{V}$ differently. Namely, instead of computing:

$$\begin{bmatrix}\mathbf{B}_{\mathsf{B}} & \mathbf{V}\end{bmatrix} := \mathbf{S}_{\mathsf{B}}\begin{bmatrix}\mathbf{A} & \mathbf{B}_{\mathsf{A}}\end{bmatrix} + \begin{bmatrix}\mathbf{D}_{\mathsf{B}} & \mathbf{E}_{\mathsf{B}}\end{bmatrix},$$

it samples

$$\begin{bmatrix}\mathbf{B}_{\mathsf{B}} & \mathbf{V}\end{bmatrix} \leftarrow\!\!\$\ \mathcal{U}(\mathbb{Z}_q^{\bar{n}\times(n+\bar{n})}).$$

Thus, one can naturally construct an efficient reduction which solves $\mathsf{LWE}_{n+\bar{n},n,\chi,q}$ with probability at least $\frac{1}{\bar{n}}|\varepsilon_6 - \varepsilon_5|$.

Finally, it is easy to see that in Γ_3 the matrix $\mathbf{V}$ is actually uniformly random over $\mathbb{Z}_q$. Hence by Lemma 1, for the adversary $\mathcal{A}$, which is given ct, the key $\mathbf{K}$ looks uniformly random. Therefore, the probability of guessing the key is bounded by $2^{-\bar{n}^2 B}$. $\square$

6.4.2 Deniability

We will use the (transposed) matrix version of ELWE where the secrets and errors are now matrices. In particular, we will be interested in the problem of distinguishing between

$$(\mathbf{A}, \mathbf{S}\mathbf{A} + \mathbf{E} \bmod q, \mathbf{Z}, \mathbf{W}, \mathbf{S}\mathbf{Z} + \mathbf{E}\mathbf{W} \bmod q)$$

```
$\Gamma_1(\mathcal{A})$
1 :  $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\!\$\; \chi(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{F}_\mathsf{A} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$
3 :  $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
4 :  $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times n}$
5 :  $\mathbf{F}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$
6 :  $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
7 :  $\mathbf{E}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$
8 :  $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}$
9 :  $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
10 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
11 : $\mathbf{K}' \leftarrow\!\!\$\; \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B}, \mathsf{ct})$
12 : return $1_{\mathbf{K}=\mathbf{K}'}$
```

```
$\Gamma_2(\mathcal{A})$
1 :  $\mathbf{B}_\mathsf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times n}$
3 :  $\mathbf{F}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$
4 :  $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
5 :  $\mathbf{E}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$
6 :  $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}$
7 :  $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
8 :  $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
9 :  $\mathbf{K}' \leftarrow\!\!\$\; \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B}, \mathsf{ct})$
10 : return $1_{\mathbf{K}=\mathbf{K}'}$
```

```
$\Gamma_3(\mathcal{A})$
1 :  $\mathbf{B}_\mathsf{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{B}_\mathsf{B} \leftarrow\!\!\$\; \mathcal{U}(\mathbb{Z}_q^{n\times\overline{n}})$
3 :  $\mathbf{V} \leftarrow\!\!\$\; \mathcal{U}(\mathbb{Z}_q^{\overline{n}\times\overline{n}})$
4 :  $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
5 :  $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
6 :  $\mathbf{K}' \leftarrow\!\!\$\; \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B}, \mathsf{ct})$
7 :  return $1_{\mathbf{K}=\mathbf{K}'}$
```

Figure 13: Security games for the proof of Lemma 8. The lines in blue highlight the main differences from the previous game.

and

$$(\mathbf{A}, \mathbf{T}, \mathbf{Z}, \mathbf{W}, \mathbf{SZ} + \mathbf{EW} \bmod q)$$

where $\mathbf{S} \leftarrow\!\!\$\; \chi^{\overline{n}\times m}$, $\mathbf{E} \leftarrow\!\!\$\; \chi^{\overline{n}\times n}$ and $\mathbf{T} \leftarrow\!\!\$\; \mathcal{U}(\mathbb{Z}_q^{\overline{n}\times n})$. This problem can be reduced to ELWE with reduction loss $\overline{n}$ via a standard hybrid argument.

We are ready to prove deniability of the split-KEM based on Extended-LWE. Intuitively, matrices $(\mathbf{S}, \mathbf{E}) := (\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B})$ will be the secret and error constructed by party B, which are hidden from the adversary, while $(\mathbf{Z}, \mathbf{W}) := (\mathbf{D}_\mathsf{A}, \mathbf{S}_\mathsf{A})$ will be the error and the secret generated by $\mathbf{A}$ which are given as input to the simulator. The key observation is that the additional hint provided as $\mathbf{SZ} + \mathbf{EW} \bmod q$ will be used to simulate the "shared key" $\mathbf{V}$ (before applying the reconciliation function).

Theorem 4 (Deniability)**.** *Let $\overline{n} = \mathsf{poly}(\lambda)$. Then, the* sKEM *defined in Figure 12 is deniable under the* $\mathsf{ELWE}_{n,n,\overline{n},\chi,q}$ *and* $\mathsf{LWE}_{n,n,\chi,q}$ *assumptions.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the deniability game. We prove the statement using the hybrid games defined in Fig. 15. In each game Γ_i, we define ε_i to be the probability that the efficient adversary $\mathcal{A}$ outputs $b = 1$.

Game Γ_1: This is the standard (real) deniability experiment, which we recall here. First, both $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\!\$\; \chi^{n\times\overline{n}}$ and $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times n}$ and $\mathbf{F}_\mathsf{A}, \mathbf{F}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$ are sampled. Then, the public keys $\mathbf{B}_\mathsf{A} = \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$ and $\mathbf{B}_\mathsf{B} = \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$ are computed. The encapsulation algorithm samples $\mathbf{E}_\mathsf{B} \leftarrow\!\!\$\; \chi^{\overline{n}\times\overline{n}}$ and sets $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}$. Finally, the experiment runs $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$ and $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$ and eventually outputs

$$(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B}, \mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A}, \mathbf{K}, \mathsf{ct})$$

to the adversary $\mathcal{A}$.

Game Γ_2: The experiment is identical to the previous one, apart from the fact that now $\mathbf{V}$ is explicitly computed as $\mathbf{V} = \mathbf{S}_\mathsf{B}\mathbf{D}_\mathsf{A} - \mathbf{D}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{E}_\mathsf{B}$. Clearly, $\varepsilon_1 = \varepsilon_2$ since

$$\begin{aligned}\mathbf{V} &= \mathbf{S}_\mathsf{B}\mathbf{D}_\mathsf{A} - \mathbf{D}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{E}_\mathsf{B}\\ &= \mathbf{S}_\mathsf{B}\mathbf{D}_\mathsf{A} - \mathbf{D}_\mathsf{B}\mathbf{S}_\mathsf{A} + (\mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B})\mathbf{S}_\mathsf{A} + \mathbf{E}_\mathsf{B}\\ &= \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}.\end{aligned}$$

Game Γ_3: Here, the experiment follows Γ_2 with the only difference being that the experiment samples $\mathbf{B}_\mathsf{B}$ uniformly at random from $\mathbb{Z}_q^{\overline{n}\times n}$ instead of computing $\mathbf{B}_\mathsf{B} = \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$.

```
Sim(A, B_B, S_A, D_A, F_A)
1 :  $\mathbf{S}_{\mathsf{sim}}, \mathbf{D}_{\mathsf{sim}} \leftarrow\!\!\$\; \chi^{\bar{n}\times n}$
2 :  $\mathbf{E}_{\mathsf{sim}} \leftarrow\!\!\$\; \chi^{\bar{n}\times\bar{n}}$
3 :  $\mathbf{V}_{\mathsf{sim}} \leftarrow \mathbf{S}_{\mathsf{sim}}\mathbf{D}_{\mathsf{A}} - \mathbf{D}_{\mathsf{sim}}\mathbf{S}_{\mathsf{A}} + \mathbf{B}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{E}_{\mathsf{sim}}$
4 :  $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V}_{\mathsf{sim}})$
5 :  $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}_{\mathsf{sim}}, \mathsf{ct})$
6 :  return $(\mathbf{K}, \mathsf{ct})$
```

Figure 14: Simulator for the deniability game.

Lemma 9. *There exists an efficient algorithm $\mathcal{B}$ that solves the $\mathsf{ELWE}_{n,n,\bar{n},\chi,q}$ problem with probability at least $\frac{1}{\bar{n}}|\varepsilon_3 - \varepsilon_2|$.*

Proof. We provide a reduction $\mathcal{B}$ to the (transposed) matrix-version of the Extended-LWE problem as described above. Namely, the reduction is given a tuple of matrices $(\mathbf{A}, \mathbf{B}, \mathbf{Z}, \mathbf{W}, \mathbf{H})$. Then, it sets $\mathbf{S}_{\mathsf{A}} := -\mathbf{W}$, $\mathbf{D}_{\mathsf{A}} := \mathbf{Z}$ and $\mathbf{B}_{\mathsf{B}} := \mathbf{B}$. Further, the reduction samples $\mathbf{F}_{\mathsf{A}} \leftarrow\!\!\$\; \chi^{\bar{n}\times\bar{n}}$ and computes

$$\mathbf{B}_{\mathsf{A}} := \mathbf{A}\mathbf{S}_{\mathsf{A}} + \mathbf{D}_{\mathsf{A}} \quad \text{and} \quad \mathbf{V} := \mathbf{H} + \mathbf{B}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{E}_{\mathsf{B}}$$

where $\mathbf{E}_{\mathsf{B}} \leftarrow\!\!\$\; \chi^{\bar{n}\times\bar{n}}$. Finally, the reduction runs $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$ and $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$ and outputs $(\mathbf{A}, \mathbf{B}_{\mathsf{A}}, \mathbf{B}_{\mathsf{B}}, \mathbf{S}_{\mathsf{A}}, \mathbf{D}_{\mathsf{A}}, \mathbf{F}_{\mathsf{A}}, \mathbf{K}, \mathsf{ct})$ to the adversary.

Suppose the input tuple received by $\mathcal{B}$ is a true Extended-LWE instance, i.e. $\mathbf{B}_{\mathsf{B}} = \mathbf{B} = \mathbf{S}_{\mathsf{B}}\mathbf{A} + \mathbf{D}_{\mathsf{B}}$ for $\mathbf{S}_{\mathsf{B}}, \mathbf{D}_{\mathsf{B}} \leftarrow\!\!\$\; \chi^{\bar{n}\times n}$. This implies that $\mathbf{H} = \mathbf{S}_{\mathsf{B}}\mathbf{Z} + \mathbf{D}_{\mathsf{B}}\mathbf{W} = \mathbf{S}_{\mathsf{B}}\mathbf{D}_{\mathsf{A}} - \mathbf{D}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}}$ and hence

$$\mathbf{V} = \mathbf{H} + \mathbf{B}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{E}_{\mathsf{B}} = \mathbf{S}_{\mathsf{B}}\mathbf{D}_{\mathsf{A}} - \mathbf{D}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{B}_{\mathsf{B}}\mathbf{S}_{\mathsf{A}} + \mathbf{E}_{\mathsf{B}}.$$

This implies that when the input tuple is the Extended-LWE instance then $\mathcal{B}$ perfectly simulates the output of Γ_2 [6]. On the other hand, if $\mathbf{B}_{\mathsf{B}}$ is uniformly random then $\mathcal{B}$ perfectly simulates the output of Γ_3. Finally the statement follows by further reducing the matrix-version of ELWE to the standard one. □

Game Γ_4: First, we rename the variables $(\mathbf{S}_{\mathsf{B}}, \mathbf{D}_{\mathsf{B}}, \mathbf{E}_{\mathsf{B}}) := (\mathbf{S}_{\mathsf{sim}}, \mathbf{D}_{\mathsf{sim}}, \mathbf{E}_{\mathsf{sim}})$. Further, instead of picking $\mathbf{B}_{\mathsf{B}}$ uniformly at random, the experiment now samples alternative secrets/errors $\mathbf{S}_{\mathsf{B}}, \mathbf{D}_{\mathsf{B}} \leftarrow\!\!\$\; \chi^{\bar{n}\times n}$ for B and sets $\mathbf{B}_{\mathsf{B}} := \mathbf{S}_{\mathsf{B}}\mathbf{A} + \mathbf{D}_{\mathsf{B}}$. The rest is identical as in Γ_3.

Lemma 10. *There exists an efficient algorithm $\mathcal{B}'$ that solves the $\mathsf{LWE}_{n,n,\chi,q}$ problem with probability at least $\frac{1}{\bar{n}}|\varepsilon_3 - \varepsilon_2|$.*

Proof. We describe a reduction $\mathcal{B}$ which solves the matrix-version of LWE. Then, the reduction to plain LWE follows by a hybrid argument. First, $\mathcal{B}$ is given a tuple $(\mathbf{A}, \mathbf{B})$ where either $\mathbf{B} = \mathbf{S}_{\mathsf{B}}\mathbf{A} + \mathbf{D}_{\mathsf{B}}$ for short $\mathbf{S}_{\mathsf{B}}, \mathbf{D}_{\mathsf{B}}$ or $\mathbf{B}$ is uniformly random. In either case, only given $\mathbf{A}$ and $\mathbf{B}$, the reduction $\mathcal{B}$ can simulate the rest of Γ_3 (and Γ_4). If $\mathbf{B} = \mathbf{S}_{\mathsf{B}}\mathbf{A} + \mathbf{D}_{\mathsf{B}}$ then this becomes Γ_4, and when $\mathbf{B}_{\mathsf{B}}$ is uniformly random then $\mathcal{B}$ simulates Γ_3. □

Finally, we present the simulator in Fig. 14. Γ_4 can now be alternatively described in the following way. The experiment first samples $\mathbf{S}_{\mathsf{A}}, \mathbf{D}_{\mathsf{A}} \leftarrow\!\!\$\; \chi^{n\times\bar{n}}$ and $\mathbf{S}_{\mathsf{B}}, \mathbf{D}_{\mathsf{B}} \leftarrow\!\!\$\; \mathbb{Z}_q^{\bar{n}\times n}$ and $\mathbf{F}_{\mathsf{A}} \leftarrow\!\!\$\; \chi^{\bar{n}\times\bar{n}}$. Further, the public keys are defined as $\mathbf{B}_{\mathsf{A}} = \mathbf{A}\mathbf{S}_{\mathsf{A}} + \mathbf{D}_{\mathsf{A}}$ and $\mathbf{B}_{\mathsf{B}} = \mathbf{S}_{\mathsf{B}}\mathbf{A} + \mathbf{D}_{\mathsf{B}}$. Finally, it runs $(\mathbf{K}, \mathsf{ct}) \leftarrow\!\!\$\; \mathsf{Sim}(\mathbf{A}, \mathbf{B}_{\mathsf{B}}, \mathbf{S}_{\mathsf{A}}, \mathbf{D}_{\mathsf{A}}, \mathbf{F}_{\mathsf{A}})$ and outputs $(\mathbf{A}, \mathbf{B}_{\mathsf{A}}, \mathbf{B}_{\mathsf{B}}, \mathbf{S}_{\mathsf{A}}, \mathbf{D}_{\mathsf{A}}, \mathbf{F}_{\mathsf{A}}, \mathbf{K}, \mathsf{ct})$. Thus, deniability follows by simply combining the previous lemmas. □

[6] We used the fact that χ is symmetric around 0 to argue that $\mathbf{S}_{\mathsf{A}} := -\mathbf{W}$ is correctly distributed.

```
Γ_1(A)
 1 : $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\!\$\ \chi(\mathbb{Z}_q^{n\times\bar{n}})$
 2 : $\mathbf{F}_\mathsf{A} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 3 : $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
 4 : $\mathsf{pk}_\mathsf{A} = (\mathbf{A}, \mathbf{B}_\mathsf{A})$
 5 : $\mathsf{sk}_\mathsf{A} = (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A})$
 6 : $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times n}$
 7 : $\mathbf{F}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 8 : $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
 9 : $\mathsf{pk}_\mathsf{B} = (\mathbf{A}, \mathbf{B}_\mathsf{B})$
10 : $\mathbf{E}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
11 : $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{B}_\mathsf{A} + \mathbf{E}_\mathsf{B}$
12 : $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
13 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
14 : $b \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathbf{K}, \mathsf{ct})$
15 : return $b$
```

```
Γ_2(A)
 1 : $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\!\$\ \chi(\mathbb{Z}_q^{n\times\bar{n}})$
 2 : $\mathbf{F}_\mathsf{A} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 3 : $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
 4 : $\mathsf{pk}_\mathsf{A} = (\mathbf{A}, \mathbf{B}_\mathsf{A})$
 5 : $\mathsf{sk}_\mathsf{A} = (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A})$
 6 : $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times n}$
 7 : $\mathbf{F}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 8 : $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
 9 : $\mathsf{pk}_\mathsf{B} = (\mathbf{A}, \mathbf{B}_\mathsf{B})$
10 : $\mathbf{E}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
11 : $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{D}_\mathsf{A} - \mathbf{D}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{E}_\mathsf{B}$
12 : $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
13 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
14 : $b \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathbf{K}, \mathsf{ct})$
15 : return $b$
```

```
Γ_3(A)
 1 : $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\!\$\ \chi(\mathbb{Z}_q^{n\times\bar{n}})$
 2 : $\mathbf{F}_\mathsf{A} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 3 : $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
 4 : $\mathsf{pk}_\mathsf{A} = (\mathbf{A}, \mathbf{B}_\mathsf{A})$
 5 : $\mathsf{sk}_\mathsf{A} = (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A})$
 6 : $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times n}$
 7 : $\mathbf{F}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 8 : $\mathbf{B}_\mathsf{B} \leftarrow \mathcal{U}(\mathbb{Z}_q^{\bar{n}\times n})$
 9 : $\mathsf{pk}_\mathsf{B} = (\mathbf{A}, \mathbf{B}_\mathsf{B})$
10 : $\mathbf{E}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
11 : $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{D}_\mathsf{A} - \mathbf{D}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{E}_\mathsf{B}$
12 : $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
13 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
14 : $b \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathbf{K}, \mathsf{ct})$
15 : return $b$
```

```
Γ_4(A)
 1 : $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\!\$\ \chi(\mathbb{Z}_q^{n\times\bar{n}})$
 2 : $\mathbf{F}_\mathsf{A} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 3 : $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
 4 : $\mathsf{pk}_\mathsf{A} = (\mathbf{A}, \mathbf{B}_\mathsf{A})$
 5 : $\mathsf{sk}_\mathsf{A} = (\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A}, \mathbf{F}_\mathsf{A})$
 6 : $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times n}$
 7 : $\mathbf{F}_\mathsf{B} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
 8 : $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
 9 : $\mathsf{pk}_\mathsf{B} = (\mathbf{A}, \mathbf{B}_\mathsf{B})$
10 : $\mathbf{E}_\mathsf{sim} \leftarrow\!\!\$\ \chi^{\bar{n}\times\bar{n}}$
11 : $\mathbf{S}_\mathsf{sim}, \mathbf{D}_\mathsf{sim} \leftarrow\!\!\$\ \chi^{\bar{n}\times n}$
12 : $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{sim}\mathbf{D}_\mathsf{A} - \mathbf{D}_\mathsf{sim}\mathbf{S}_\mathsf{A} + \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{E}_\mathsf{sim}$
13 : $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
14 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
15 : $b \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathbf{K}, \mathsf{ct})$
16 : return $b$
```

Figure 15: Security games for the proof of Theorem 4. The lines in blue highlight the main differences from the previous game. The lines in gray correspond to the simulator defined in Fig. 14.

6.4.3 Decaps-OW-CPA Security

Finally, we show that our split-KEM satisfies the decaps-OW-CPA security notion (see Definition 3.6).

Lemma 11 (decaps-OW-CPA Security). *Let $\bar{n} = O(\lambda)$, m be such that the ciphertext space of* sKEM *is $\{0,1\}^m$, and χ be a probability distribution over $[-\gamma, \gamma]$ symmetric around 0 for any $\gamma > 0$. Suppose $\mathsf{LWE}_{n+\bar{n},n,\chi,q}$ is hard. Then, for every efficient algorithm $\mathcal{A}$, the probability of winning the* decaps-OW-CPA *game is at most $2^m \cdot (\delta_\mathsf{cpa}^{\bar{n}^2} + \mathsf{negl}(\lambda))$ where*

$$\delta_\mathsf{cpa} := \max_{\substack{\mathsf{ct}\in\{0,1\} \\ u\in\mathbb{Z}_{2B}}} \Pr_{w\leftarrow\$\mathbb{Z}_q} [\mathsf{Rec}(2w, \mathsf{ct}) = u] \ . \tag{3}$$

Proof. Let $\mathcal{A}$ be an efficient adversary against the decaps-OW-CPA game. We prove the statement using the hybrid games described explicitly in Fig. 16. In each game Γ_i, we define ε_i to be the probability that the efficient adversary $\mathcal{A}$ wins the security game.

Game Γ_1: This is the standard decaps-OW-CPA game corresponding to the sKEM in Fig. 12.

Game Γ_2: In this game, the ciphertext ct is not given to the adversary anymore. Note that the first phase adversary outputting $\mathbf{B}$ is now useless and it can be removed, along with the operations needed to compute ct. Given the ciphertext space is $\{0,1\}^m$ for some $m \in \mathbb{Z}$, we have $\epsilon_2 \geq \frac{1}{2^m}\epsilon_1$ as any adversary in Γ_2 can simulate the view of an adversary in Γ_1 by guessing ct.

Game Γ_3: In this game, the only change is that instead of computing $\mathbf{B}_\mathsf{B} = \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$, it is picked uniformly at random from $\mathbb{Z}_q^{\bar{n}\times n}$. The indistinguishability between Γ_3 and Γ_2 follows directly from $\mathsf{LWE}_{n,n,\chi,q}$.

Game Γ_4: Now, instead of computing $\mathbf{B}_\mathsf{A}$ and $\mathbf{V}'$ as:

$$\begin{bmatrix}\mathbf{B}_\mathsf{A}\\ \mathbf{V}'\end{bmatrix} = \begin{bmatrix}\mathbf{A}\\ \mathbf{B}_\mathsf{B}\end{bmatrix}\mathbf{S}_\mathsf{A} + \begin{bmatrix}\mathbf{D}_\mathsf{A}\\ \mathbf{F}_\mathsf{A}\end{bmatrix},$$

the experiment samples $\mathbf{B}_\mathsf{A} \leftarrow\!\!\$\ \mathcal{U}(\mathbb{Z}_q^{n\times\bar{n}})$ and $\mathbf{V}' \leftarrow\!\!\$\ \mathcal{U}(\mathbb{Z}_q^{\bar{n}\times\bar{n}})$ uniformly at random. Then, the reduction executes Lines 4 to 6 of Γ_4. Clearly there is an efficient adversary which solves $\mathsf{LWE}_{n+\bar{n},n,\chi,q}$ with probability at least $\frac{1}{\bar{n}}|\varepsilon_4 - \varepsilon_3|$.

Finally, since $\mathbf{V}'$ is uniformly random, the probability that any adversary wins Γ_4, i.e. $\mathbf{K}_\mathsf{A} = \mathbf{K}'_\mathsf{A}$, can be upper-bounded by $\delta_\mathsf{cpa}^{\bar{n}^2}$ by definition of δ_cpa. The statement now follows by combining the previous hybrid games. □

Remark. We highlight that the construction does not require a super-polynomial modulus. To see this, let us first focus on δ_cpa in (3). In the instantiation we set $B = O(1), \bar{n} = O(\sqrt{\lambda})$ and $\gamma = O(1)$. Further, δ_cpa can be bounded by $2^{-B} + 1/q$, and if $q = \mathsf{poly}(\lambda)$ then $\delta_\mathsf{cpa} \leq (2^{-B} + 1/q)^{\bar{n}^2} = \mathsf{negl}(\lambda)$. Hence, the winning probability in Lemma 11 is negligible for $q = \mathsf{poly}(\lambda)$.

Now, let us move on to Equation 2. Recall that n is a dimension responsible for hardness of LWE, say $n = \mathsf{poly}(\lambda)$. Suppose as in Lemma 11 that the distribution χ outputs integers between $-\gamma$ and γ. Then, to obtain $\delta_\mathsf{corr} = 0$, we need to pick $q > 2^{B+2}(2n\gamma^2 + 2\gamma) = \mathsf{poly}(\lambda)$, hence asymptotically the modulus can still be polynomial. In practice (and in this work), we would allow negligible δ_corr, and thus we further decrease the modulus.

6.5 Building a UNF-1KCA and IND-1BatchCCA Split-KEM

We have proven so far that the modified version of FrodoKEX given above is decaps-OW-CPA and OW-CPA. We show now that any scheme satisfying both these properties can easily be transformed into a UNF-1KCA and IND-1BatchCCA split-KEM in the ROM and QROM. The construction is similar to the T_CH transform introduced by Huguenin-Dumittan and Vaudenay [HV22] translated to the split-KEM setting. We present it in Figure 17. Then, the following theorem states the security guarantees of the resulting split-KEM.

Theorem 5. *Let* sKEM_0 *be any split-KEM and* $\mathsf{sKEM} := \mathsf{T}_\mathsf{CH}^\mathsf{skem}(\mathsf{sKEM}_0)$ *be the split-KEM obtained from applying the* $\mathsf{T}_\mathsf{CH}^\mathsf{skem}$ *transform (Figure 17) to* sKEM_0*. Then, in the ROM, we have*

$\Gamma_1(\mathcal{A})$

```
1 :  $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\$\, \chi(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{F}_\mathsf{A} \leftarrow\!\$\, \chi^{\overline{n}\times\overline{n}}$
3 :  $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
4 :  $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\$\, \chi^{\overline{n}\times n}$
5 :  $\mathbf{F}_\mathsf{B} \leftarrow\!\$\, \chi^{\overline{n}\times\overline{n}}$
6 :  $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
7 :  $\mathbf{B} \leftarrow\!\$\, \mathcal{A}(\mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B})$
8 :  $\mathbf{E}_\mathsf{B} \leftarrow\!\$\, \chi^{\overline{n}\times\overline{n}}$
9 :  $\mathbf{V} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{B} + \mathbf{E}_\mathsf{B}$
10 : $\mathsf{ct} \leftarrow \mathsf{HelpRec}(\mathbf{V})$
11 : $\mathbf{K} \leftarrow \mathsf{Rec}(2\mathbf{V}, \mathsf{ct})$
12 : $\mathbf{K}'_\mathsf{A}, \mathsf{ct}' \leftarrow\!\$\, \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B}, \mathsf{ct})$
13 : $\mathbf{V}' \leftarrow \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{F}_\mathsf{A}$
14 : $\mathbf{K}_\mathsf{A} \leftarrow \mathsf{Rec}(2\mathbf{V}', \mathsf{ct}')$
15 : **return** $1_{\mathbf{K}_\mathsf{A}=\mathbf{K}'_\mathsf{A}}$
```

$\Gamma_2(\mathcal{A})$

```
1 :  $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\$\, \chi(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{F}_\mathsf{A} \leftarrow\!\$\, \chi^{\overline{n}\times\overline{n}}$
3 :  $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
4 :  $\mathbf{S}_\mathsf{B}, \mathbf{D}_\mathsf{B} \leftarrow\!\$\, \chi^{\overline{n}\times n}$
5 :  $\mathbf{B}_\mathsf{B} \leftarrow \mathbf{S}_\mathsf{B}\mathbf{A} + \mathbf{D}_\mathsf{B}$
6 :  $\mathbf{K}'_\mathsf{A}, \mathsf{ct}' \leftarrow\!\$\, \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B})$
7 :  $\mathbf{V}' \leftarrow \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{F}_\mathsf{A}$
8 :  $\mathbf{K}_\mathsf{A} \leftarrow \mathsf{Rec}(2\mathbf{V}', \mathsf{ct}')$
9 :  **return** $1_{\mathbf{K}_\mathsf{A}=\mathbf{K}'_\mathsf{A}}$
```

$\Gamma_3(\mathcal{A})$

```
1 :  $\mathbf{S}_\mathsf{A}, \mathbf{D}_\mathsf{A} \leftarrow\!\$\, \chi(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{F}_\mathsf{A} \leftarrow\!\$\, \chi^{\overline{n}\times\overline{n}}$
3 :  $\mathbf{B}_\mathsf{A} \leftarrow \mathbf{A}\mathbf{S}_\mathsf{A} + \mathbf{D}_\mathsf{A}$
4 :  $\mathbf{B}_\mathsf{B} \leftarrow\!\$\, \mathcal{U}(\mathbb{Z}_q^{\overline{n}\times n})$
5 :  $\mathbf{K}'_\mathsf{A}, \mathsf{ct}' \leftarrow\!\$\, \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B})$
6 :  $\mathbf{V}' \leftarrow \mathbf{B}_\mathsf{B}\mathbf{S}_\mathsf{A} + \mathbf{F}_\mathsf{A}$
7 :  $\mathbf{K}_\mathsf{A} \leftarrow \mathsf{Rec}(2\mathbf{V}', \mathsf{ct}')$
8 :  **return** $1_{\mathbf{K}_\mathsf{A}=\mathbf{K}'_\mathsf{A}}$
```

$\Gamma_4(\mathcal{A})$

```
1 :  $\mathbf{B}_\mathsf{A} \leftarrow\!\$\, \mathcal{U}(\mathbb{Z}_q^{n\times\overline{n}})$
2 :  $\mathbf{B}_\mathsf{B} \leftarrow\!\$\, \mathcal{U}(\mathbb{Z}_q^{\overline{n}\times n})$
3 :  $\mathbf{V}' \leftarrow\!\$\, \mathcal{U}(\mathbb{Z}_q^{\overline{n}\times\overline{n}})$
4 :  $\mathbf{K}'_\mathsf{A}, \mathsf{ct}' \leftarrow\!\$\, \mathcal{A}(\mathbf{A}, \mathbf{B}_\mathsf{A}, \mathbf{B}_\mathsf{B})$
5 :  $\mathbf{K}_\mathsf{A} \leftarrow \mathsf{Rec}(2\mathbf{V}', \mathsf{ct}')$
6 :  **return** $1_{\mathbf{K}_\mathsf{A}=\mathbf{K}'_\mathsf{A}}$
```

Figure 16: Security games for the proof of Lemma 11. The lines in blue highlight the main differences from the previous game.

```
KeyGen_sKEM(1^λ)
  1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}_{\mathsf{sKEM}_0}(1^\lambda)$
  2 : return $(\mathsf{pk}, \mathsf{sk})$

Decaps_sKEM(pk_B, sk_A, (ct, t))
  1 : $K_0' \leftarrow \mathsf{Decaps}_{\mathsf{sKEM}_0}(\mathsf{pk_B}, \mathsf{sk_A}, (\mathsf{ct}, t))$
  2 : if $H'(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}, K_0') \neq t$ :
  3 :     return $\perp$
  4 : return $H(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}, K_0')$

Encaps_sKEM(pk_A, sk_B)
  1 : $K_0, \mathsf{ct} \leftarrow\$ \mathsf{Encaps}_{\mathsf{sKEM}_0}(\mathsf{pk_A}, \mathsf{sk_B})$
  2 : $t \leftarrow H'(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}, K_0)$
  3 : $K \leftarrow H(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}, K_0)$
  4 : return $K, (\mathsf{ct}, t)$
```

Figure 17: $\mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}$ transform for split-KEMs. We assume that $\mathsf{pk_B}$ can be derived from $\mathsf{sk_B}$ or is contained in it.

that for any efficient UNF-1KCA *adversary $\mathcal{A}$, one can build efficient $\mathcal{B}$ and $\mathcal{C}$ adversaries s.t.*

$$\mathsf{Adv}_{\mathsf{sKEM}}^{\mathrm{unf-1kca}}(\mathcal{A}) \leq \frac{q_{H'}^2 + 1}{2^s} + (q_H + q_{H'} + 1) \cdot \mathsf{Adv}_{\mathsf{sKEM}_0}^{\mathrm{decaps-ow-cpa}}(\mathcal{C}) \ ,$$

where q_H and $q_{H'}$ are the number of queries made by $\mathcal{A}$ to the random oracles H and H', respectively, and s is the output size of both random oracles. In the QROM, the bound becomes

$$\mathsf{Adv}_{\mathsf{sKEM}}^{\mathrm{unf-1kca}}(\mathcal{A}) \leq \frac{8(q_H + q_{H'})^2}{2^{2s}} + \epsilon + 2(2(q_H + q_{H'}) + 1)^2 \cdot \mathsf{Adv}_{\mathsf{sKEM}_0}^{\mathrm{decaps-ow-cpa}}(\mathcal{B}) \ ,$$

where $\epsilon := \frac{2}{2^s} + 8\sqrt{2/2^s} + \frac{40e^2(q_{H'}+2)^3+2}{2^s}$.

Proof. We defer the proof to Appendix A. □

Similarly, we have that the $\mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}$ transform makes an IND-1BatchCCA scheme out of an OW-CPA one, which is stated in the following theorem.

Theorem 6. *Let sKEM_0 be any split-KEM and $\mathsf{sKEM} := \mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}(\mathsf{sKEM}_0)$ be the split-KEM obtained from applying the $\mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}$ transform (Figure 17) to sKEM_0. Then, in the ROM, we have that for any efficient* IND-1BatchCCA *adversary $\mathcal{A}$, one can build efficient $\mathcal{B}$ s.t.*

$$\mathsf{Adv}_{\mathsf{sKEM}}^{\mathrm{ind-1batchcca}}(\mathcal{A}) \leq \frac{q_{H'}^2 + d}{2^s} + 2(q_H + q_{H'} + d) \cdot \mathsf{Adv}_{\mathsf{sKEM}_0}^{\mathrm{ow-cpa}}(\mathcal{B})$$

where q_H and $q_{H'}$ are the number of queries made by $\mathcal{A}$ to the random oracles H and H', respectively, s is the output size of both random oracles, and d is the number of tuples submitted to the IND-1BatchCCA oracle BatchDec. In the QROM, the previous bound becomes

$$\mathsf{Adv}_{\mathsf{sKEM}}^{\mathrm{ind-1batchcca}}(\mathcal{A}) \leq \delta + \epsilon_1 + \epsilon_2 + \epsilon_3 + 2(q_H + d + q_{H'})\sqrt{2\mathsf{Adv}_{\mathsf{sKEM}_0}^{\mathrm{ow-cpa}}(\mathcal{B})} \ ,$$

where δ is the correctness error, $\epsilon_1 = \frac{40e^2(q_{H'}+d+1)^3+2}{2^s}$, $\epsilon_2 = 8d(d+2q_{H'}+1)\sqrt{2/2^s}$ *and* $\epsilon_3 = \frac{4d}{2^s}$.

Proof. As the proof is nearly identical to the proof of IND-qCCA security of the T_{CH} transform for PKE/KEM [HV22], we defer it to the Appendix B. □

6.6 Concrete Instantiation

In Table 2 we propose a parameter set for FrodoKEX+ where we aim for 256-bit security before applying the transform and 128-bit (resp. 64-bit) security after the transform assuming 2^{64} random oracle (resp. quantum random oracle) queries. In addition, we give the security terms in Table 3. We show in the following how these parameters were computed, where we set $(B, \bar{n}) = (4, 8)$.

n	$\bar{n}$	q	B	χ	$\lvert t\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{ct}\rvert$
1452	8	31751	4	$\mathcal{U}(\{-1,1\})$	64B	21.3KB	72B

Table 2: Concrete parameters for our lattice-based split-KEM. We note that in practice, we do not need to include the whole matrix $\mathbf{A}$ in the public key pk, but rather the seed for the pseudorandom function to generate it (as is the case in this table). The ciphertext ct comprises the original split-KEM ciphertext (8B) and the tag t (64B).

$\bar{n}^2\delta_{\mathsf{corr}}$ (2)	δ_{elwe} (1)	δ_{cpa} (3)
2^{-48}	2^{-46}	$2^{-3.9996}$

Table 3: Correctness and security terms.

6.6.1 Correctness error and security loss

One of the main challenges in instantiating our FrodoKEX variant is computing δ_{corr} and δ_{elwe} from Equations 2 and 1. They are related to the correctness error and the security loss of ELWE. To this end, we introduce a simple distribution χ which will allow us to efficiently compute these values. Namely, we set χ to be a uniform distribution over the set $\{-1,1\}$. Clearly, it is symmetric around 0 and has standard deviation equal to 1.

Another useful property of this distribution is that a product XY, where $X,Y \leftarrow\!\!\$\ \chi$, still follows the distribution of χ. Based on this observation, we have

$$\delta_{\mathsf{corr}} = \Pr_{\substack{X_1,\ldots,X_{2n+1}\leftarrow\$\chi\\ E\leftarrow\$\chi}}\left[\left|\sum_{i=1}^{2n+1} X_i + E\right| > \frac{q}{2^{B+2}}\right].$$

We can directly compute this term using Laurent polynomials. Namely, define

$$P(X) := \Pr_{X\leftarrow\$\chi}[X=1]\cdot X + \Pr_{X\leftarrow\$\chi}[X=-1]\cdot X^{-1} = \frac{1}{2}\cdot\left(X+X^{-1}\right).$$

Then, using the convolution properties, we observe that the probability of $X_1+\ldots+X_{2n+2}=k$, for some $-2n-2\le k\le 2n+2$, is equal to the k-th coefficient of the polynomial $P(X)^{2n+2}$. Hence, we calculate δ_{corr} by computing $P(X)^{2n+2}$ and summing all the k-th coefficients, such that $2n+2\ge |k| > \frac{q}{2^{B+2}}$.

We now turn into computing δ_{elwe}. The first step is the analysis of the following random variable $\mathbf{v}=\frac{1}{2}\cdot(e-d)\cdot\mathbf{z}$, where $e,d\leftarrow\!\!\$\ \chi$ and $\mathbf{z}\leftarrow\!\!\$\ \chi^{\bar{n}}$. We denote this distribution as $\mathcal{V}$. By simple calculation we get:

$$\Pr[\mathbf{v}=\mathbf{a}] = \Pr\left[\frac{1}{2}\cdot(e-d)\cdot\mathbf{z}=\mathbf{a}\right] = \begin{cases}\frac{1}{2} \text{ if } \mathbf{a}=\mathbf{0}\\ \frac{1}{2^{\bar{n}+1}} \text{ if } \mathbf{a}\in\{-1,1\}^{\bar{n}}\\ 0 \text{ otherwise}\end{cases}$$

Then, the multivariate Laurent polynomial corresponding to $\mathbf{v}$ has an elegant form:

$$P(X_1,\ldots,X_{\bar{n}}) = \frac{1}{2} + \frac{1}{2^{\bar{n}+1}}\prod_{i=1}^{\bar{n}}(X_i+X_i^{-1}).$$

As before, we observe that δ_{elwe} is the probability that for $\mathbf{v}_1,\ldots,\mathbf{v}_{n+m}\leftarrow\!\!\$\ \mathcal{V}$,

$$2\cdot(\mathbf{v}_1+\ldots+\mathbf{v}_{n+m}) = \mathbf{0} \pmod q \iff \mathbf{v}_1+\ldots+\mathbf{v}_{n+m}=\mathbf{0}^{7}.$$

[7]This holds as long as $n+m<q/2$ since then no modulo overflow occurs.

In terms of the newly defined Laurent polynomials, δ_{elwe} is the constant coefficient of:

$$P(X_1,\ldots,X_{\bar{n}})^{n+m} = \textstyle\sum_{j=0}^{n+m}\binom{n+m}{j}\frac{1}{2^{n+m-j}}\cdot\frac{1}{2^{(\bar{n}+1)j}}\cdot\prod_{i=1}^{\bar{n}}(X_i+X_i^{-1})^j.$$

We now look at the constant coefficient of each of the $n+m+1$ terms of the sum. The first observation is that

$$(X_i+X_i^{-1})^j = \sum_{k=0}^{j}\binom{j}{k}X_i^{(j-k)}X_i^{-k} = \sum_{k=0}^{j}\binom{j}{k}X_i^{(j-2k)}.$$

Hence, the constant coefficient of the expression above is 0 if j is odd, and $\binom{j}{j/2}$ when j is even. Consequently, the constant coefficient of $\prod_{i=1}^{\bar{n}}(X_i+X_i^{-1})^j$ is either 0, for odd j, or $\binom{j}{j/2}^{\bar{n}}$ for even j. Hence, we conclude that

$$\delta_{\mathsf{elwe}} = \sum_{j\text{ even}}\binom{n+m}{j}\frac{1}{2^{n+m-j}}\cdot\frac{1}{2^{(\bar{n}+1)j}}\cdot\binom{j}{\frac{j}{2}}^{\bar{n}}$$

which can then be computed efficiently for our parameters. Finally, δ_{cpa} can be straightforwardly computed for small primes [8], such as $\approx 2^{15}$.

6.6.2 Hardness of Extended-LWE

We measure the hardness following the methodology used for the original FrodoKEX [Bos+16] for fair comparison, and refer to it for more details on the attacks. Here, the main bottleneck of setting the parameters is the reduction loss between ELWE and plain LWE. Taking this into account for the parameters proposed above, we aim for 307-bit classical LWE security.

We consider the primal and dual BKZ attacks [SE94; CN11]. As a subroutine, the BKZ algorithm with block-size b uses an algorithm for the shortest vector problem (SVP) in lattices of dimension b. As in Frodo [Bos+16], for precautionary purposes we only count the cost of one such call (even though in practice it will run the SVP sub-algorithm polynomially many times). The lower-bound on the time complexity of one call is given by about $b2^{cb}$ CPU cycles, where $c \approx 0.292$ for classical attacks, and $c \approx 0.265$ for quantum attacks (see Laarhoven [Laa16, Section 14.2.10]). For 307-bit classical security, this corresponds to the block size being 1018, and the root Hermite factor being ≈ 1.0020 (in the quantum setting these parameters correspond to 279 bits of security). Further, we estimate the hardness of LWE against known attacks using the LWE estimator by Albrecht et al. [APS15]. Namely, we run the estimator under both “sieving” and “enumeration”, and set the final root Hermite factor δ as the largest root Hermite factor returned by the program. Finally, we make sure that $\delta_{\mathsf{cpa}}^{\bar{n}^2} \approx 2^{-256}$ for the decaps-OW-CPA proof.

7 Benchmarks, Comparison and Discussion

Hereafter, we refer to the X3DH-like protocol of Brendel et al. [Bre+22] as SPQR, and the baseline deniable protocol (i.e., with ring signatures and without NIZKs) by Hashimoto et al. [Has+22] as HKKP.

7.1 Benchmarks

Security of the relevant non-standard primitive. Like K-Waay, SPQR and HKKP can each be implemented using only (soon to be) standardised primitives, except for a single primitive in each case, we consider here the security of the relevant non-standard primitives. In

[8] One can formally prove that δ_{cpa} can be bounded by $\frac{1}{2^B}+\frac{1}{q}$, but we compute the value directly instead.

Scheme	Cl. (C)	Cl. (Q)	ROM bnd	QROM bnd	Assumption
FrodoKEX+	128	64	$(q_H + d)/2^{192}$	$(q_H + d)/2^{128}$	LWE
Raptor [LAZ19]	114	103	?	✗	NTRU
DualRing-LB [Yue+21]	(128)	(64)	?	✗	MSIS, MLWE
Falafl [BKP20]	128	64	?	✗	MSIS, MLWE

Table 4: Security comparison between FrodoKEX+and several post-quantum RS. 'Cl.' stands for claimed number of security bits. DualRing-LB's authors do not seem to make a clear security claim, we thus assume NIST level I. '?' indicates that no bound is explicitly given for the security, '✗' indicates that no proof is provided in the QROM.

the case of K-Waay it is a split-KEM, here implemented using a variant of FrodoKEX passed through the $\mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}$ transform (that we call FrodoKEX+), and in the case of both HKKP and SPQR it is a ring signature scheme, or RS (or a designated-verifier signature scheme (DVS) derived from RS). The authors of both SPQR and HKKP proposed possible implementations for the RS without picking one in particular. The most efficient one for a ring of size 2 we are aware of that has an existing C implementation is Raptor [LAZ19] which we use for the benchmarks below. Other candidates would be Falafl [BKP20] or DualRing-LB [Yue+21].

We present in Table 4 a summary of the security claims, approximate leading factor in the bounds in the (Q)ROM, and assumptions for these non-standard primitives. We note that none of these primitives are proven secure in the standard model and all are based on lattices.

First, we note that parameters for these RS schemes were chosen *before* the reduction in the ROM was performed. That is, a primitive P based on lattices is built, parameters are chosen such that P satisfies the security claim, then P is used to build a RS in the ROM, which incurs a loss factor that usually depends on the number of queries to the random oracle q_H. In particular, it is common to have at least a q_H factor in the security bound (e.g. if the adversary can make 2^{64} queries to the RO, the security level is reduced by 64 bits). Therefore, the claimed security level does not match the *provable* security level. In the QROM, the security loss is usually greater: square root and q_H^2 or q_H^3 losses are quite common, however these schemes have not been proven secure in this model.

We took a different approach in designing a split-KEM with a conservative assumption (i.e., plain LWE) and choice of parameters. Therefore, FrodoKEX+ with our proposed parameters achieve 128 (resp. 64) bits of classical (resp. quantum) security *after* the (Q)ROM proof. We provide the (approximate) highest terms of both the ROM and QROM security bounds in Table 4. These satisfy our security claims as long as $q_H + d \leq 2^{64}$, where d is the number of public key/ciphertext tuples allowed in the IND-1BatchCCA game. In K-Waay, d corresponds to the number of distinct users trying to communicate with an offline receiver after all prekeys have run out, and thus should typically be small.

The reason behind the approximations and lack of QROM proofs for PQ ring signatures is likely the youth of the field and the speed at which it is evolving. Still, we believe it is worth noting as it makes any comparison between our protocol and previous ones quite difficult.

Benchmarking. The protocols we benchmarked are: our own implementation of the X3DH protocol; Brendel et al.'s [Bre+22] protocol SPQR based on PQ KEMs, a signature scheme and DVS; Hashimoto et al.'s [Has+22] protocol HKKP based on PQ KEMs, a signature scheme and RS; a baseline protocol made only with PQ KEMs and a signature scheme similar to the *non-deniable* variant of HKKP; and our protocol K-Waay based on FrodoKEX+ as a PQ split-KEM, PQ KEMs and a signature scheme.[9]

[9]SPQR and HKKP do not formally treat (regular) signatures, but we include them for fair comparison and because a practical system would use signatures for prekey bundles.

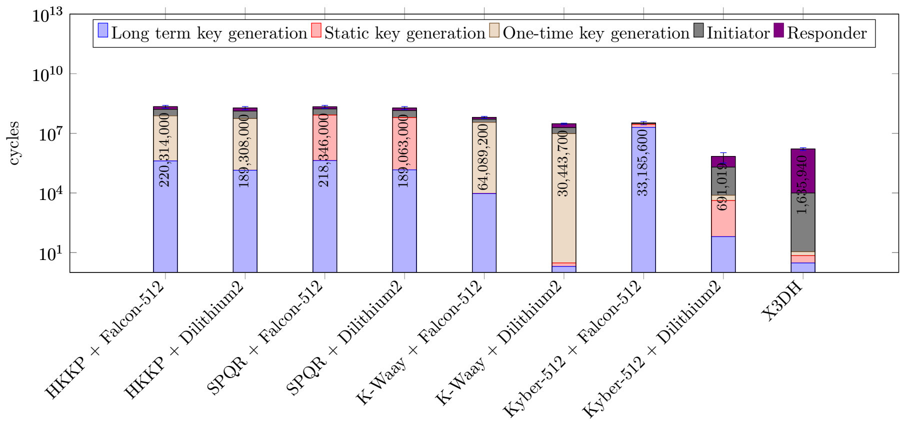

Figure 18: Speed benchmark for X3DH protocols.

We chose Kyber-512 as the KEM, both Falcon-512 and Dilithium2 for signatures, and Raptor for ring signatures. We implemented both HKKP and SPQR with signed prekeys as is the case in Signal's implementation of X3DH. That is, a PQ signature key pair is part of the long-term key, and ephemeral keys uploaded to server are signed with it. Note that this is make explicit in K-Waay as the ephemeral keys are signed with the long-term one. The authors of HKKP show that this is not necessary in their protocol, however not doing so weakens perfect forward secrecy.

We built the different protocols in C using the `liboqs` library[10] for Kyber, Falcon, and Dilithium, the Raptor implementation provided by the authors[11], and a modified version of the `lwe-frodo` library[12] with scaled parameters to properly simulate FrodoKEX+. More precisely, the modulus was set to the first power of 2 larger than the modulus in FrodoKEX+, the addition of the noise during decapsulation was also added, and the noise distributions were modified to match the ones of FrodoKEX+. We did not optimise the scheme in any way (e.g. by using AVX instructions or parallelisation) and we leave this as future work. For the sake of completeness, we also provide a reference implementation of FrodoKEX+ in Rust[13] for the interested reader. All benchmarks were run on a virtual machine running Ubuntu 22.04 with 2 cores of an Intel i7-9750H running at 2.60GHz and 4GB of RAM allocated.

Speed. For the speed benchmark, we measured how many cycles each protocol takes in one execution. We summarise our results in a logarithmic graph on Figure 18 (note that the internal division of the bars is linear).

Fixing the choice of KEM and signature scheme, our protocol K-Waay, depending on the choice of KEM and signature scheme, is between 3 and 6 times faster than the previous proposals even with our relatively conservative parameter choice. In our protocol K-Waay using Dilithium2, most cycles are spent in the ephemeral key generation, while using Falcon makes the static key generation as expensive as the ephemeral key one. Overall, one can see that Falcon, while more compact than Dilithium2, has a great impact on efficiency. For instance, K-Waay with Dilithium2 is faster than the non-deniable scheme using Kyber and Falcon.

Apart from Falcon, we see that the most time-consuming primitives are the non-standard ones, i.e., ring signatures and split-KEM. Hence, we see that the KEM+SIG protocol (HKKP's baseline proposal) performs even better than X3DH, which shows once again that lattice-based schemes can be faster than their classical counterparts. However, we recall that it does not

[10] `https://github.com/open-quantum-safe/liboqs`
[11] `https://github.com/zhenfeizhang/raptor`
[12] `https://github.com/lwe-frodo/lwe-frodo`
[13] `https://github.com/lehugueni/frodokexp-rust`

Scheme	|lpk|	|prek|	|ct|
K-Waay + Dilithium	2112	24520	1632
K-Waay + Falcon	1697	22790	1632
HKKP [Has+22]	1700	1700	4056
HKKP [Has+22] + Dilithium2	3012	4120	4056
HKKP [Has+22] + Falcon	2597	2390	4056
SPQR [Bre+22]	3400	1632	4824
SPQR [Bre+22] + Dilithium2	4712	4052	4824
SPQR [Bre+22] + Falcon	4297	2322	4824

Table 5: Size comparison in bytes between K-Waay instantiated with FrodoKEX+, HKKP [Has+22] and SPQR [Bre+22]. We also computed the sizes for both HKKP and SPQR implemented with signed prekey bundles.

provide deniability. At last, we note that X3DH is the only construction that spends more time in sending and receiving than generating keys. Our protocol's Send and Receive (i.e. BatchReceive with a single input message) procedures are very fast.

Data size. In Table 5, we provide for each scheme the size of the long-term keys, the prekeys (output by Init in our DAKE syntax), and the ciphertext output by the sender. We computed for both HKKP and SPQR the size with and without long-term signatures. We see that K-Waay compares well in terms of long-term public key and ciphertext size as both are smaller than signed HKKP and SPQR. However, the prekeys are much larger as one could expect from a LWE-based scheme and due to our conservative setting of parameters.

7.2 Advantages, Limitations and Discussion

Running out of ephemeral keys. The main disadvantage of our protocol is that running out of ephemeral keys requires the receiver to abort if *any* of the sessions that used the same prekey is bogus. If this happens, then a malicious party could mount some kind of denial of service (DoS) attack against the user that was offline for too long by sending a bogus split-KEM ciphertext. There is an obvious trade-off between the risk of such an attack happening and the number of ephemeral keys uploaded on the server (and thus also storage). We leave the analysis and the mitigation of such a threat as future work, but we believe that if a reasonable amount of prekeys are uploaded, creating fake accounts is difficult (e.g., by requiring a phone number as in Signal), and/or users are online often enough, such an attack would be difficult to mount. Furthermore, several practical mitigations are possible. For instance, if the receiver (i.e. the victim) received a bogus ciphertext among the n ciphertexts sent for the same prekey while offline, they can restart K-Waay with each the n parties but as the initiator, which will probably succeed. The victim could also send n new prekeys to the n initiators directly, making sure the protocol will succeed at the next iteration. This would make the attack less useful as it could only *delay* communication and not prevent it.

We also think it is worth mentioning that the trick we propose might be easy to misimplement. In particular, it is crucial that no information about which split-KEM ciphertext failed leaks if such a situation occurs. That is, precautions should be taken such that leakage via side-channels in the scope of the system designer's threat model are prevented.

split-KEM instead of ring signatures. The fact that we use a primitive similar to a post-quantum KEM allows us to leverage the extensive literature on the topic and existing safe/optimised implementations. This also gives good security guarantees as post-quantum KEMs have been heavily scrutinised as part of the NIST standardisation process. For example, as mentioned above, our proposed lattice-based implementation is based on a key-exchange variant of FrodoKEM, which is itself the PQ KEM recommended by the German Federal Office for Information Security (BSI) [Inf23]. Overall we think that a split-KEM such as FrodoKEX+ is more mature and closer to being usable in practice than ring signatures.

On the necessity of modifying FrodoKEX. Currently, our split-KEM significantly differs from the original FrodoKEX in two aspects: (i) the modulus for our construction has to be prime in order for our reduction from Extended-LWE to LWE to hold[14], and (ii) we have to introduce additional masking terms to prove UNF-1KCA security. However, we believe that both changes are artefacts of the security proofs, and the original FrodoKEX split-KEM should be (up to a reasonable security loss) deniable.

There are alternative reduction techniques from Extended-LWE to LWE in the literature [Bou+21; Bra+13], which do not rely on having an odd modulus at the cost of using discrete Gaussian error distributions with large parameters. It is thus an interesting research problem to efficiently reduce Extended-LWE to LWE for even modulus with small reduction loss. In practice, the most efficient LWE attacks do not consider the structure of the modulus, so intuitively this should translate to the Extended-LWE setting[15].

As for our second main modification, it is unclear how to argue decaps-OW-CPA security without the additional masking terms.

Deniability. While the signature on the ephemeral public keys might give the impression that our protocol is less deniable than X3DH or previous PQ alternatives, this is actually not the case. The reason is that prekey bundles in these protocols are signed as well, but this detail is abstracted away in the analysis (i.e. it is assumed that all parties have received and authenticated <u>all</u> public keys before the protocol actually starts). While this kind of analysis allows for strong deniability claims, in practice these protocols do not satisfy something stronger than some kind of peer-deniability. The exception is the ring signature based variant by Hashimoto et al. [Has+22], where the prekey bundle is not necessarily signed. However, in this variant, the authors can only prove the security of their protocol in a weaker model (i.e. it satisfies a weaker notion of forward secrecy). Overall, if deniability should not come at the price of security, peer-deniability seems like the best notion one can achieve in these DAKEs.

We wished to provide a transparent model for peer-deniability, where the upload of signed ephemeral keys is made explicit. We also strengthen the deniability definition of Brendel et al. [Bre+22] by allowing the exposure of one of the parties (i.e. the receiving one, which would be the malicious party trying to frame the sender). While our protocol satisfies our stronger (in terms of key exposure) notion of deniability, we believe both previous PQ X3DH alternatives satisfy it as well. Indeed, in these schemes, the ephemeral keys are KEM and RS keys only, which are deniable. Hence, exposing these should not harm deniability.

Hashimoto et al. [Has+22] consider a strong notion of deniability where the adversary is malicious (i.e. can arbitrarily deviate from the protocol) and show how to modify HKKP such that it is secure against such a threat. However, such deniability comes at the expense of NIZKs, which are complex, expensive and are not always proven secure in the QROM when random oracles are used. Moreover, as in other deniable systems against malicious adversaries, non-falsifiable assumptions (i.e., knowledge-type assumptions) are required to prove the security. In addition, it seems difficult to defend against adversaries actively trying to frame a given user in messaging in practice [GPA19; CCH23]; for example, an adversary could also simply ask questions that would identify the victim with good probability. Because of these reasons, we do not consider such a notion of deniability here.

[14] Recall FrodoKEX [Bos+16] uses a power-of-two modulus for efficiency.

[15] Recently, various frameworks have been developed [Dac+20; Dac+23], which measure concrete hardness of LWE given hints of specific form, such as linear combination of secrets with short random coefficients.

To contextualise our results, we remark here that cryptographic deniability, which is targeted by this work and all previous work on deniable X3DH key exchange, translates to deniability on a *system* level only if the application preserves deniability. For example, Collins et al. [CCH23] observe that Signal as currently deployed does not provide this kind of 'practical' deniability for ordinary users. Suppose Bob is trying to frame Alice and hands over their phone that contains a transcript of communication between Alice and Bob to a judge. Because Signal authenticates users (either directly or indirectly through Signal sealed sender [Lun18]), unless Bob was able to modify their phone (which depends on the technical expertise of Bob), the judge can deduce that the conversation plausibly took place as in the transcript, regardless of the cryptographic protocols employed underneath. It is interesting future work to further explore deniability on the broader system level and practical deniability [Rei+23; YGS23].

An optimisation. As presented in Section 5, the K-Waay protocol generates a signature for each ephemeral public key uploaded. This can easily be optimised by signing the whole prekey bundle containing several ephemeral keys. This way, the server needs to store only one PQ signature for each user. The downside is that now each user needs to download the whole bundle to verify the signature. This offers a trade-off between data stored at the server and sent to clients.

Improving security and efficiency. Note that we proved the key indistinguishability of K-Waay in a model that considers state exposures like Hashimoto et al.'s [Has+22] but is nonetheless weaker (our protocol is provably-secure under a notion similar to that of [Bre+22]). This is mainly since our protocol only uses ephemeral split-KEM keys. As noted by Brendel et al. [Bre+21], however, it seems much more difficult to construct split-KEM secure under several encapsulation/decapsulation queries, which we leave again as important future work.

An interesting line of research would be to try to build other unforgeable IND-1BatchCCA split-KEMs that are more efficient (mostly in key and ciphertext size). One obvious direction would be to work over structured lattices [LS15; LPR10; Ste+09]. Indeed, Ring/Module-LWE with hints (similar to our Extended-LWE problem) have already been analysed from a theoretical point of view, e.g. [Bou+21; Mer+22]. We also believe that our techniques can also be applied in the ring setting. However, for security purposes one needs to take a ring dimension d to be at least linear in the security parameter λ which becomes problematic when proving deniability. Indeed, the leaked hint is informally the product of secret keys of both parties. Thus, in the ring setting the hint would be at least a single polynomial, which contains $d = O(\lambda)$ coefficients. We predict that this would result with much larger reduction loss than what we have now. However, the concrete analysis is left as future work.

On a more practical side, it would be informative to benchmark our protocol and others in a real-life scenario or something close to it, and to implement other ring signatures schemes to have a more complete comparison. In light of the recent deployment of the PQXDH protocol [KS23], it would be prudent also to benchmark this and compare it with our protocols, replacing our choice of Kyber-512 with Kyber-1024 to be consistent with PQXDH which uses the latter KEM (note that the non-standard primitive used for the deniable PQ protocols will nonetheless be more of a bottleneck).

One could also try to build one-time ring signatures that are both efficient and provably secure (possibly in the QROM). In turn, these could possibly be used to build efficient ephemeral split-KEMs. For instance, Scafuro et al. [SZ21] designed an efficient linkable one-time ring signatures from hash functions alone and proved the security of the scheme in the ROM. It would be of interest to understand whether split-KEMs can be built out of such a construction or a variant, and/or to prove the security in the QROM. Different parameter sets to achieve higher level of security could also be provided and benchmarked.

References

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. "The double ratchet: Security notions, proofs, and modularization for the signal protocol". In: *EUROCRYPT*. Springer. 2019, pp. 129–158.

[Alw+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. "Analysing the HPKE standard". In: *EUROCRYPT 2021*. Springer. 2021, pp. 87–116.

[AP12] Jacob Alperin-Sheriff and Chris Peikert. "Circular and KDM Security for Identity-Based Encryption". In: *Public Key Cryptography*. Vol. 7293. Lecture Notes in Computer Science. Springer, 2012, pp. 334–352.

[App+09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. "Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems". In: *CRYPTO*. Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 595–618.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. *On the concrete hardness of Learning with Errors*. Cryptology ePrint Archive, Report 2015/046. `https://ia.cr/2015/046`. 2015.

[Bad+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. "Tightly-Secure Authenticated Key Exchange". In: *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9014. Lecture Notes in Computer Science. Springer, 2015, pp. 629–658. DOI: `10.1007/978-3-662-46494-6\_26`. URL: `https://doi.org/10.1007/978-3-662-46494-6%5C_26`.

[Bel06] Mihir Bellare. "New proofs for NMAC and HMAC: Security without collision-resistance". In: *CRYPTO 2006*. Springer. 2006, pp. 602–619.

[Ber06] Daniel J Bernstein. "Curve25519: new Diffie-Hellman speed records". In: *International Workshop on Public Key Cryptography*. Springer. 2006, pp. 207–228.

[Bha+23] Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. *An Analysis of Signal's PQXDH*. `https://cryspen.com/post/pqxdh/` Accessed: 23.10.23. 2023.

[Bie+22] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. "A More Complete Analysis of the Signal Double Ratchet Algorithm". In: *CRYPTO 2022*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13507. Lecture Notes in Computer Science. Springer, 2022, pp. 784–813. DOI: `10.1007/978-3-031-15802-5\_27`.

[BKP20] Ward Beullens, Shuichi Katsumata, and Federico Pintore. "Calamari and Falafl: Logarithmic (Linkable) Ring Signatures from Isogenies and Lattices". In: *ASIACRYPT (2)*. Vol. 12492. Lecture Notes in Computer Science. Springer, 2020, pp. 464–492.

[Bon+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *ASIACRYPT*. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 41–69.

[Bos+16] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. "Frodo: Take off the ring! practical, quantum-secure key exchange from LWE". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 1006–1018.

[Bos+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM". In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P*. 2018, pp. 353–367.

[Bou+21] Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, and Weiqiang Wen. "On the Hardness of Module-LWE with Binary Secret". In: *CT-RSA*. Vol. 12704. Lecture Notes in Computer Science. Springer, 2021, pp. 503–526.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS*. ACM, 1993, pp. 62–73.

[Bra+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. "Classical Hardness of Learning with Errors". In: *CoRR* abs/1306.0281 (2013).

[Bre+21] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. "Towards post-quantum security for signal's X3DH handshake". In: *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*. Springer. 2021, pp. 404–430.

[Bre+22] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. "Post-quantum asynchronous deniable key exchange and the signal handshake". In: *IACR International Conference on Public-Key Cryptography*. Springer. 2022, pp. 3–34.

[Can+22] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. "Universally Composable End-to-End Secure Messaging". In: *CRYPTO 2022*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. Lecture Notes in Computer Science. Springer, 2022, pp. 3–33. DOI: `10.1007/978-3-031-15979-4\_1`.

[Cas+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven D. Galbraith. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 395–427. DOI: `10.1007/978-3-030-03332-3\_15`.

[CCH23] Daniel Collins, Simone Colombo, and Loïs Huguenin-Dumittan. "Real World Deniability in Messaging". In: *Cryptology ePrint Archive* (2023).

[CD23] Wouter Castryck and Thomas Decru. "An efficient key recovery attack on SIDH". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2023, pp. 423–447.

[CF11] Cas Cremers and Michele Feltz. *One-round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability*. Cryptology ePrint Archive, Paper 2011/300. 2011. URL: `https://eprint.iacr.org/2011/300`.

[CK02] Ran Canetti and Hugo Krawczyk. "Security analysis of IKE's signature-based key-exchange protocol". In: *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer. 2002, pp. 143–161.

[CN11] Yuanmi Chen and Phong Q. Nguyen. "BKZ 2.0: Better Lattice Security Estimates". In: *ASIACRYPT*. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 1–20.

[Coh+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. "Highly efficient key exchange protocols with optimal tightness". In: *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer. 2019, pp. 767–797.

[Coh+20] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. "A Formal Security Analysis of the Signal Messaging Protocol". In: *J. Cryptol.* 33.4 (2020), pp. 1914–1983. DOI: `10.1007/s00145-020-09360-1`.

[Col+24] Daniel Collins, Loïs Huguenin-Dumittan, Ngoc Khanh Nguyen, Nicolas Rolin, and Serge Vaudenay. "K-Waay: Fast and Deniable Post-Quantum X3DH without Ring Signatures". In: *33rd USENIX Security Symposium (USENIX Security 24)*. `https://www.usenix.org/conference/usenixsecurity24/presentation/collins`. 2024.

[CZ24] Cas Cremers and Mang Zhao. "Secure Messaging with Strong Compromise Resilience, Temporal Privacy, and Immediate Decryption". In: *IEEE S&P*. 2024.

[Dac+20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. "LWE with Side Information: Attacks and Concrete Security Estimation". In: *CRYPTO (2)*. Vol. 12171. Lecture Notes in Computer Science. Springer, 2020, pp. 329–358.

[Dac+23] Dana Dachman-Soled, Huijing Gong, Tom Hanson, and Hunter Kippen. "Revisiting Security Estimation for LWE with Hints from a Geometric Perspective". In: *Advances in Cryptology – CRYPTO 2023*. Ed. by Helena Handschuh and Anna Lysyanskaya. Cham: Springer Nature Switzerland, 2023, pp. 748–781.

[DG22] Samuel Dobson and Steven D. Galbraith. "Post-Quantum Signal Key Agreement from SIDH". In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*. Ed. by Jung Hee Cheon and Thomas Johansson. Vol. 13512. Lecture Notes in Computer Science. Springer, 2022, pp. 422–450. DOI: `10.1007/978-3-031-17234-2\_20`.

[DH76] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654. DOI: `10.1109/TIT.1976.1055638`. URL: `https://doi.org/10.1109/TIT.1976.1055638`.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. "Concurrent zero-knowledge". In: *Journal of the ACM (JACM)* 51.6 (2004), pp. 851–898.

[Don+22] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. "Online-Extractability in the Quantum Random-Oracle Model". In: *EUROCRYPT 2022*. Ed. by Orr Dunkelman and Stefan Dziembowski. Cham: Springer International Publishing, 2022, pp. 677–706. ISBN: 978-3-031-07082-2.

[Duc+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.1 (2018), pp. 238–268.

[ESZ22] Muhammed F. Esgin, Ron Steinfeld, and Raymond K. Zhao. "MatRiCT^{+}: More Efficient Post-Quantum Private Blockchain Payments". In: *IEEE Symposium on Security and Privacy*. IEEE, 2022, pp. 1281–1298.

[GPA19] Lachlan J Gunn, Ricardo Vieitez Parra, and N Asokan. "Circumventing Cryptographic Deniability with Remote Attestation". In: *Proceedings on Privacy Enhancing Technologies* 3 (2019), pp. 350–369.

[Has+21] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. "An Efficient and Generic Construction for Signal's Handshake (X3DH): Post-Quantum, State Leakage Secure, and Deniable". In: *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II.* Ed. by Juan A. Garay. Vol. 12711. Lecture Notes in Computer Science. Springer, 2021, pp. 410–440. DOI: `10.1007/978-3-030-75248-4\_15`.

[Has+22] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. "An efficient and generic construction for signal's handshake (X3DH): post-quantum, state leakage secure, and deniable". In: *Journal of Cryptology* 35.3 (2022), p. 17.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. "A modular analysis of the Fujisaki-Okamoto transformation". In: *Theory of Cryptography Conference.* Springer. 2017, pp. 341–371.

[HV22] Loïs Huguenin-Dumittan and Serge Vaudenay. "On IND-qCCA security in the ROM and its applications: CPA security is sufficient for TLS 1.3". In: *EUROCRYPT 2022.* Springer. 2022, pp. 613–642.

[Inf23] BSI - German Federal Office for Information Security. *BSI TR-01102-1.* `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html`. 2023.

[JMZ23] Haodong Jiang, Zhi Ma, and Zhenfeng Zhang. "Post-Quantum Security of Key Encapsulation Mechanism against CCA Attacks with a Single Decapsulation Query". In: *Cryptology ePrint Archive* (2023).

[Kil+23] Eike Kiltz, Jiaxin Pan, Doreen Riepel, and Magnus Ringerud. "Multi-user CDH problems and the concrete security of NAXOS and HMQV". In: *Cryptographers' Track at the RSA Conference.* Springer. 2023, pp. 645–671.

[Kra05] Hugo Krawczyk. "HMQV: A high-performance secure Diffie-Hellman protocol". In: *Annual international cryptology conference.* Springer. 2005, pp. 546–566.

[KS23] Ehren Kret and Rolfe Schmidt. *The PQXDH Key Agreement Protocol.* `https://signal.org/docs/specifications/pqxdh/pqxdh.pdf`. 2023.

[Laa16] Thijs Laarhoven. *Search problems in cryptography: from fingerprinting to lattice sieving.* English. Proefschrift. Feb. 2016.

[LAZ19] Xingye Lu, Man Ho Au, and Zhenfei Zhang. "Raptor: a practical lattice-based (linkable) ring signature". In: *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17.* Springer. 2019, pp. 110–130.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. "Stronger security of authenticated key exchange". In: *Provable Security: First International Conference, ProvSec 2007, Wollongong, Australia, November 1-2, 2007. Proceedings 1.* Springer. 2007, pp. 1–16.

[LN22] Vadim Lyubashevsky and Ngoc Khanh Nguyen. "BLOOM: Bimodal Lattice One-out-of-Many Proofs and Applications". In: *ASIACRYPT (4).* Vol. 13794. Lecture Notes in Computer Science. Springer, 2022, pp. 95–125.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *EUROCRYPT.* Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 1–23.

[LS15] Adeline Langlois and Damien Stehlé. "Worst-case to average-case reductions for module lattices". In: *Des. Codes Cryptogr.* 75.3 (2015), pp. 565–599.

[Lun18] Joshua Lund. *Technology preview: Sealed sender for Signal.* `https://signal.org/blog/sealed-sender/`. Last visited on 13-09-2023. 2018.

[Mer+22] Jose Maria Bermudo Mera, Angshuman Karmakar, Tilen Marc, and Azam Soleimanian. "Efficient Lattice-Based Inner-Product Functional Encryption". In: *Public Key Cryptography (2).* Vol. 13178. Lecture Notes in Computer Science. Springer, 2022, pp. 163–193.

[MM11] Daniele Micciancio and Petros Mol. "Pseudorandom Knapsacks and the Sample Complexity of LWE Search-to-Decision Reductions". In: *CRYPTO.* Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 465–484.

[MP16] Moxie Marlinspike and Trevor Perrin. "The x3dh key agreement protocol". In: *Open Whisper Systems* 283 (2016).

[Pei14] Chris Peikert. "Lattice Cryptography for the Internet". In: *PQCrypto.* Vol. 8772. Lecture Notes in Computer Science. Springer, 2014, pp. 197–219.

[PM16] Trevor Perrin and Moxie Marlinspike. "The double ratchet algorithm". In: *GitHub wiki* (2016).

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing.* 2005, pp. 84–93.

[Rei+23] Nathan Reitinger, Nathan Malkin, Omer Akgul, Michelle L Mazurek, and Ian Miers. "Is Cryptographic Deniability Sufficientf Non-Expert Perceptions of Deniability in Secure Messaging". In: *2023 IEEE Symposium on Security and Privacy (SP).* IEEE. 2023, pp. 274–292.

[RGK06] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. "Deniable authentication and key exchange". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006.* Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM, 2006, pp. 400–409. DOI: `10.1145/1180405.1180454`.

[SE94] Claus-Peter Schnorr and M. Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Math. Program.* 66 (1994), pp. 181–199.

[Sho94] Peter W. Shor. "Polynominal time algorithms for discrete logarithms and factoring on a quantum computer". In: *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings.* Ed. by Leonard M. Adleman and Ming-Deh A. Huang. Vol. 877. Lecture Notes in Computer Science. Springer, 1994, p. 289. DOI: `10.1007/3-540-58691-1\_68`.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Post-Quantum TLS Without Handshake Signatures". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM, 2020, pp. 1461–1480. DOI: `10.1145/3372297.3423350`.

[Ste+09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. "Efficient Public Key Encryption Based on Ideal Lattices". In: *ASIACRYPT.* Vol. 5912. Lecture Notes in Computer Science. Springer, 2009, pp. 617–635.

[SZ21] Alessandra Scafuro and Bihan Zhang. "One-time traceable ring signatures". In: *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II 26.* Springer. 2021, pp. 481–500.

[UG15] Nik Unger and Ian Goldberg. "Deniable Key Exchanges for Secure Messaging". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 1211–1223. DOI: `10.1145/2810103.2813616`. URL: `https://doi.org/10.1145/2810103.2813616`.

[UG18] Nik Unger and Ian Goldberg. "Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging". In: *Proc. Priv. Enhancing Technol.* 2018.1 (2018), pp. 21–66. DOI: `10.1515/popets-2018-0003`.

[US] US National Security Agency. *Announcing the Commercial National Security Algorithm Suite 2.0.* `https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF`.

[Vat+20] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. "On the Cryptographic Deniability of the Signal Protocol". In: *ACNS 2020*. Ed. by Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi. Vol. 12147. Lecture Notes in Computer Science. Springer, 2020, pp. 188–209. DOI: `10.1007/978-3-030-57878-7\_10`.

[YGS23] Tarun Kumar Yadav, Devashish Gosain, and Kent Seamons. "Cryptographic Deniability: A Multi-perspective Study of User Perceptions and Expectations". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 3637–3654.

[Yue+21] Tsz Hon Yuen, Muhammed F Esgin, Joseph K Liu, Man Ho Au, and Zhimin Ding. "DualRing: generic construction of ring signatures with efficient instantiations". In: *CRYPTO 2021*. Springer. 2021, pp. 251–281.

```
COLL(A)
1 : $(x_1, t_1), \dots, (x_m, t_m) \leftarrow \mathcal{A}^{\mathcal{S}}$
2 : for $i \in \{1, \dots, m\}$ : $t'_i \leftarrow \mathcal{S}(x_i)$
3 : for $i \in \{1, \dots, m\}$ : $x^{\mathsf{e}}_i \leftarrow \mathsf{Ext}(t_i)$
4 : if $\exists i : x^{\mathsf{e}}_i \neq x_i$ and $t_i = t'_i$ :
5 :     return 1
6 : return 0
```

Figure 19: Collision game for Property 8 Definition A.1.

A Proof of Theorem 5

A.1 QROM preliminaries

In this section, we assume the random oracles output values in $\{0,1\}^n$ for some integer n. We first recall the notion of extractable random-oracle simulator introduced by Don et al. [Don+22] and the corresponding properties as presented in [HV22], and a useful lemma. We refer the reader to the original paper for more details.

Definition A.1 (Extractable RO [Don+22])**.** *An extractable RO-simulator is a tuple $(\mathcal{S}, \mathsf{Ext})$, where $\mathcal{S}$ is a compressed RO efficiently simulatable and* Ext *is the extractor, such that the following properties hold.*

1. *If the extractor is never called, the simulator is indistinguishable from a (standard) RO.*
2. *Any two subsequent independent queries to $\mathcal{S}$ commute.*
3. *Any two subsequent independent queries to* Ext *commute.*
4. *Any two subsequent independent queries to* Ext *and $\mathcal{S}$ $8\sqrt{2/2^n}$-almost commute.*
5. *Querying classically the simulator $\mathcal{S}$ on the same value multiples times in a row has the same effect on the state of $\mathcal{S}$ as making* one *of these queries.*
6. *Let $x^{\mathsf{e}} \leftarrow \mathsf{Ext}(t)$ for some t, and $t' \leftarrow \mathcal{S}(x^{\mathsf{e}})$ be two subsequent classical queries. Then,*
$$\Pr[t \neq t' \wedge x^{\mathsf{e}} \neq \perp] \leq 2/2^n \ .$$
7. *Let $t \leftarrow \mathcal{S}(x)$ for some x and $x^{\mathsf{e}} \leftarrow \mathsf{Ext}(t)$ be two subsequent classical queries. Then,*
$$\Pr[\widehat{x} = \perp] \leq 2/2^n \ .$$
8. *Let* COLL *be the game defined in Figure 19. Then, for any $\mathcal{A}$ we have*
$$\Pr[\mathsf{COLL}(\mathcal{A}) \Rightarrow 1] \leq \frac{40e^2(q+m+1)^3+2}{2^n} \ ,$$
where q is the number of queries $\mathcal{A}$ makes to $\mathcal{S}$ and m is the number of tuples output by $\mathcal{A}$ in the game.

Lemma 12 (Early Extraction)**.** *Let Γ be a game where an adversary runs on some input, queries a quantum RO H, and outputs two values t and o. Then, the game applies some deterministic function on o to obtain a value x and queries $h \leftarrow H(x)$. The game returns 1 if $h = t$. Now let Γ' be the same as Γ, except the extractor is called on t right after it is returned by $\mathcal{A}$, and the game returns 1 if $h = t$* and *$x = x^\star$, where $x^\star$ is the value extracted. Then,*
$$\Pr[\Gamma \Rightarrow 1] - \Pr[\Gamma' \Rightarrow 1] \leq \frac{2}{2^n} + 8\sqrt{2/2^n} + \frac{40e^2(q_H+2)^3+2}{2^n}.$$

```
$\mathcal{S}_{i^*}^{H,\mathcal{A}}(\Theta)$
1 : $(j,b) \leftarrow\$ (\{0,\ldots q_H - 1\} \setminus \{i^*\} \times \{0,1\}) \cup \{(q_H, 0)\}$
2 : $q \leftarrow 1$
3 : $(x,z) \leftarrow\$ \mathcal{A}^{H'}$ and
4 : $x' \leftarrow$ measure $\mathcal{A}$'s $j+1$-th query input register
5 : return $(x,z)$
```

```
$H'(x)$
1 : if $q = i^*$ :
2 :   return $\Theta$
3 : if $q < j + b + 1$ :
4 :   return $H(x)$
5 : else
6 :   if $x = x'$ :
7 :     return $\Theta$
8 :   else :
9 :     return $H(x)$
```

Figure 20: Algorithm $\mathcal{S}$ for Lemma 14.

Proof. This follows from Corollary 4.7 in Don et al. and the fact that if $h = t$, where $h = H(x)$, then $\Pr[x^\star = \bot] \leq \frac{2}{2^n}$. ☐

We also recall the algorithmic one-way to hiding lemma [HHK17].

Lemma 13 (AOW2H [HHK17])**.** *Let $\mathcal{A}$ be a quantum adversary making at most q_H queries to the QRO $H : \{0,1\}^\ell \mapsto \{0,1\}^n$ and outputting 0 or 1. Then, for any algorithm F that does not use H*

$$\begin{aligned}&\big|\Pr[\mathcal{A}^H(inp) \Rightarrow 1 : \sigma^* \leftarrow\$ \{0,1\}^\ell; inp \leftarrow \mathsf{F}(\sigma^*, H(\sigma^*))]\\ &- \Pr[\mathcal{A}^H(inp) \Rightarrow 1 : (\sigma^*, K) \leftarrow\$ \{0,1\}^{\ell+n}; inp \leftarrow \mathsf{F}(\sigma^*, K)]\big|\\ &\leq 2q_H \sqrt{\Pr\left[\sigma^* \leftarrow \mathsf{E}^{\mathcal{A},H}(inp) : \begin{array}{c}(\sigma^*, K) \leftarrow\$ \{0,1\}^{\ell+n};\\ inp \leftarrow \mathsf{F}(\sigma^*, K)\end{array}\right]}\ .\end{aligned}$$

where E is an algorithm that runs $\mathcal{A}$, measures the input register of a random query made to H, and outputs the result.

Finally, we recall the measure-and-reprogram lemma of Jiang et al. [JMZ23].

Lemma 14 (Lemma 3.1 [JMZ23])**.** *Let $H : \{0,1\}^m \mapsto \{0,1\}^n$ be a quantum random oracle and $\mathcal{A}^H$ be a quantum algorithm that makes q quantum queries to H and outputs (x,z), where x and z are classical. Furthermore, we assume the i^*-th query of $\mathcal{A}$ to H is classical and equal to x, for some $i^* \in [q_H]$. In addition, let $V(x,y,z)$ be some predicate s.t. $V(x,y,z) = 1$ implies that y was output on $\mathcal{A}$'s i^*-th query to H.*
Then, there exists an algorithm $\mathcal{S}_{i^*}$ (see Figure 20), that takes some $\Theta \in \{0,1\}^n$ as input and is such that

$$\Pr\left[V(x,H(x),z) = 1 : (x,z) \leftarrow \mathcal{A}^H\right] \leq 2(2q_H+1)^2 \Pr\left[V(x,\Theta,z) = 1 : (x,z) \leftarrow \mathcal{S}_{i^*}^{\mathcal{A}}(\Theta)\right] + \frac{8q_H^2}{2^n}$$

where the probabilities are taken over the randomness of the algorithms, the random oracle H and the sampling of Θ at random.

Informally, the previous lemma states that if some adversary $\mathcal{A}^H$ can satisfy a predicate with probability p, one can build another algorithm $\mathcal{S}^{\mathcal{A}}$ that does not query H on the i^*-th query (but uses its input instead) but that can satisfy with probability $\approx \frac{p}{q_H^2}$.

A.2 Proof in the QROM

We proceed with a sequence of games that is detailed in Figure 21. The proof uses the extractable RO-simulator of Don et al. [Don+22] (see Definition A.1).

Game Γ_0: This is the UNF-1KCA game with $\mathsf{sKEM} := \mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}(\mathsf{sKEM}_0)$ written explicitly. In addition, the RO used to compute the tag corresponding to ct (i.e. $t = H_1(\mathsf{pk}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}, K_\mathsf{B})$) is different from the one used to compute the tag for ct' (i.e. $t_c = H_2(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}', \mathcal{K}_\mathsf{A})$). Note that since $(\mathsf{pk}, \mathsf{ct}) \neq (\mathsf{pk}_\mathsf{A}, \mathsf{ct}')$ for the adversary to win, both oracles can be separated in this way.

Game Γ_1: The game is the same as the previous one, except we use the simulated RO for H_2, and we use the extractor on t' (the tag output by the adversary) at the end. Note that this does not change anything to the probability of success of the game.

Game Γ_2: Now the game outputs 0 if the values extracted are different than $(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}', K_\mathsf{A})$. For the game to return 1, t_c must be equal to t', so let's assume it is the case. Hence, Γ^2 and Γ^1 differ only if $\mathsf{S.Ext}(t_c) \neq (\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}', K_\mathsf{A})$ and $H_2(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}', K_\mathsf{A}) = \mathsf{t}_c$. By Lemma 12, this happens with probability at most $\epsilon := \frac{2}{2^n} + 8\sqrt{2/2^n} + \frac{40e^2(q_{H'}+2)^3+2}{2^n}$. Hence, we have

$$\Pr[\Gamma_1] - \Pr[\Gamma_2] \leq \epsilon \ .$$

Game Υ_1: We see that if an adversary $\mathcal{A}$ wins Γ_2, one can build an adversary $\mathcal{B}$ that wins the game Υ_1 defined in Figure 21. The reduction works simply by $\mathcal{B}$ running $\mathcal{A}$, simulating H_2 with the simulated RO, and running the extractor on t' at the end. Therefore, we have

$$\Pr[\Gamma_2] \leq \Pr[\Upsilon_1] \ .$$

In addition, note that one can consider oracles H and H_1 as *one* oracle $H^* := H_1 \otimes H$ with images in $\{0,1\}^{2n}$ that can be accessed $q_H + q_{H'}$ times by the adversary.

Game Υ_2: We change the game such that (t, K) are picked at random and the oracle used is now $\hat{H}$ instead of $H^* := H_1 \otimes H$. Now, let's consider a game $\mathcal{C}$ that runs Γ^1 and outputs $(x = (\mathsf{pk}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}, K_\mathsf{B}), z = ((t, K), K_\mathsf{A}, K'_\mathsf{A})$. In addition, let $V(x, y, z) := 1_{z_1 = y} \wedge 1_{z_2 = z_3}$. Clearly, we have that

$$\Pr[\Upsilon_1] \leq \Pr[V(x, H^*(x), z) : (x, z) \leftarrow\!\!\$\ \mathcal{C}^{H^*}]$$

as V is satisfied iff $K_\mathsf{A} = K'_\mathsf{A}$. Also, note that the condition $z_1 = y$ in the predicate is always satisfied by the definition of z_1 itself. Therefore, one can apply Lemma 14 with i^* equal to the query to H^* made by the game (i.e. $(t, K) \leftarrow H^*(\mathsf{pk}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}, K_\mathsf{B})$) and we get

$$\Pr[\Upsilon_1] \leq 2(2(q_H + q_{H'}) + 1)^2 \Pr\left[V(x, (t, K), z) = 1 : (x, z) \leftarrow \mathcal{S}_{i^*}^{\mathcal{A}}((t, K)\right] + \frac{8(q_H + q_{H'})^2}{2^{2n}}$$

where (t, K) is sampled at random and $\mathcal{S}_{i^*}$ is the algorithm shown in Figure 20. By inspection, one can see that if the output of $\mathcal{S}_{i^*}$ satisfies the predicate V then Υ_2 would output 1. Therefore, we have

$$\Pr[\Upsilon_1] \leq 2(2(q_H + q_{H'}) + 1)^2 \Pr\left[V(x, (t, K), z) = 1 : (x, z) \leftarrow \mathcal{S}_{i^*}^{\mathcal{A}}((t, K)\right] + \frac{8(q_H + q_{H'})^2}{2^{2n}} \leq \Pr[\Upsilon_2].$$

Finally, one can see that if $\mathcal{A}$ wins Υ_2, one can build an adversary $\mathcal{B}$ s.t. $\mathcal{B}$ wins the decaps-OW-CPA game against sKEM_0. That is, the first phase of $\mathcal{B}$ runs the first phase of $\mathcal{A}$ and outputs the same public key pk. Then, in the second phase, $\mathcal{B}$ runs $\mathcal{A}^{\hat{H}}$ with its own input $(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{ct})$ and random tag and key (t, K). In addition, note that $\mathcal{B}$ can perfectly simulate $\hat{H}$.

```
$\Gamma_0(\mathcal{A})$
$\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\$ \mathsf{KeyGen_A}(1^\lambda)$
$\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\$ \mathsf{KeyGen_B}(1^\lambda)$
$\mathsf{pk} \leftarrow\$ \mathcal{A}^{H,H_1,H_2}(\mathsf{pk_A}, \mathsf{pk_B})$
$(K_\mathsf{B}, \mathsf{ct}) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
$t \leftarrow H_1(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$K \leftarrow H(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$(\mathsf{ct}', t') \leftarrow\$ \mathcal{A}^{H,H_1,H_2}(\mathsf{pk_A},$
            $\mathsf{pk_B}, (\mathsf{ct}, t), K)$
$\mathbf{if}\ (\mathsf{pk_A}, \mathsf{ct}') = (\mathsf{pk}, \mathsf{ct}) :\ \mathbf{return}\ 0$
$K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
$t_c \leftarrow H_2(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A})$
$\mathbf{if}\ t_c \neq t' :\ \mathbf{return}\ 0$
$\mathbf{return}\ 1$
```

```
$\Gamma_1(\mathcal{A})$
$\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\$ \mathsf{KeyGen_A}(1^\lambda)$
$\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\$ \mathsf{KeyGen_B}(1^\lambda)$
$\mathsf{pk} \leftarrow\$ \mathcal{A}^{H,H_1,H_2}(\mathsf{pk_A}, \mathsf{pk_B})$
$(K_\mathsf{B}, \mathsf{ct}) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
$t \leftarrow H_1(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$K \leftarrow H(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$(\mathsf{ct}', t') \leftarrow\$ \mathcal{A}^{H,H_1,H_2}(\mathsf{pk_A},$
            $\mathsf{pk_B}, (\mathsf{ct}, t), K)$
$\mathbf{if}\ (\mathsf{pk_A}, \mathsf{ct}') = (\mathsf{pk}, \mathsf{ct}) :\ \mathbf{return}\ 0$
$K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
$t_c \leftarrow H_2(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A})$
$\mathbf{if}\ t_c \neq t' :\ \mathbf{return}\ 0$
$(\mathsf{pk}_1^\star, \mathsf{pk}_2^\star, \mathsf{ct}^\star, K_\mathsf{A}^\star) \leftarrow \mathsf{S.Ext}(t')$
$\mathbf{return}\ 1$
```

```
$\Gamma_2(\mathcal{A})$
$\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\$ \mathsf{KeyGen_A}(1^\lambda)$
$\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\$ \mathsf{KeyGen_B}(1^\lambda)$
$\mathsf{pk} \leftarrow\$ \mathcal{A}^{H,H_1,H_2}(\mathsf{pk_A}, \mathsf{pk_B})$
$(K_\mathsf{B}, \mathsf{ct}) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
$t \leftarrow H_1(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$K \leftarrow H(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$(\mathsf{ct}', t') \leftarrow\$ \mathcal{A}^{H,H_1,H_2}(\mathsf{pk_A},$
            $\mathsf{pk_B}, (\mathsf{ct}, t), K)$
$(\mathsf{pk}_0^\star, \mathsf{pk}_1^\star, \mathsf{ct}^\star, K_\mathsf{A}^\star) \leftarrow \mathsf{S.Ext}(t')$
$\mathbf{if}\ (\mathsf{pk_A}, \mathsf{ct}') = (\mathsf{pk}, \mathsf{ct}) :\ \mathbf{return}\ 0$
$K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
$t_c \leftarrow H_2(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A})$
$\mathbf{if}\ t_c \neq t' :\ \mathbf{return}\ 0$
$\mathbf{if}\ (\mathsf{pk}_1^\star, \mathsf{pk}_2^\star, \mathsf{ct}^\star, K_\mathsf{A}^\star) \neq (\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A}) :$
  $\mathbf{return}\ 0$
$\mathbf{return}\ 1$
```

```
$\Upsilon_1(\mathcal{A})$
$\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\$ \mathsf{KeyGen_A}(1^\lambda)$
$\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\$ \mathsf{KeyGen_B}(1^\lambda)$
$\mathsf{pk} \leftarrow\$ \mathcal{A}^{H,H_1}(\mathsf{pk_A}, \mathsf{pk_B})$
$(K_\mathsf{B}, \mathsf{ct}) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
$(t, K) \leftarrow H^*(\mathsf{pk}, \mathsf{pk_B}, \mathsf{ct}, K_\mathsf{B})$
$in \leftarrow (\mathsf{pk_A}, \mathsf{pk_B}, (\mathsf{ct}, t), K)$
$K'_\mathsf{A}, \mathsf{ct}' \leftarrow\$ \mathcal{A}^{H,H_1}(in)$
$K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
$\mathbf{if}\ (\mathsf{pk_A}, \mathsf{ct}') = (\mathsf{pk}, \mathsf{ct})\ \mathbf{or}\ K_\mathsf{A} \neq K'_\mathsf{A} :$
  $\mathbf{return}\ 0$
$\mathbf{return}\ 1$
```

```
$\Upsilon_2(\mathcal{A})$
$(j, b) \leftarrow\$ (\{0, \ldots q_H - 1\} \times \{0, 1\}) \cup \{(q_H, 0)\}$
$x' \leftarrow$ measure $\mathcal{A}$'s $j + 1$-th query input register
$q \leftarrow 0$
$\mathsf{pk_A}, \mathsf{sk_A} \leftarrow\$ \mathsf{KeyGen_A}(1^\lambda)$
$\mathsf{pk_B}, \mathsf{sk_B} \leftarrow\$ \mathsf{KeyGen_B}(1^\lambda)$
$(K_\mathsf{B}, \mathsf{ct}) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk_A}, \mathsf{sk_B})$
$\mathsf{pk} \leftarrow\$ \mathcal{A}^{\hat{H}}(\mathsf{pk_A}, \mathsf{pk_B})$
$(K_\mathsf{B}, \mathsf{ct}) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}, \mathsf{sk_B})$
$(t, K) \leftarrow\$ \{0, 1\}^{2n}$
$K'_\mathsf{A}, \mathsf{ct}' \leftarrow\$ \mathcal{A}^{\hat{H}}(\mathsf{pk_A}, \mathsf{pk_B}, (\mathsf{ct}, t), K)$
$K_\mathsf{A} \leftarrow \mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}')$
$\mathbf{if}\ K_\mathsf{A} \neq K'_\mathsf{A} :$
  $\mathbf{return}\ 0$
$\mathbf{return}\ 1$
```

```
$\hat{H}(x)$
$q \leftarrow q + 1$
$\mathbf{if}\ q < j + b + 1 :$
  $\mathbf{return}\ H^*(x)$
$\mathbf{else}$
  $\mathbf{if}\ x = x' :$
    $\mathbf{return}\ (t, K)$
  $\mathbf{else}\ :$
    $\mathbf{return}\ H^*(x)$
```

Figure 21: Sequence of games for the proof of Theorem 5. H^* is defined as $H_1 \otimes H_1$.

Finally, $\mathcal{B}$ outputs the same as the adversary $\mathcal{A}$. If $K_\mathsf{A} = K'_\mathsf{A}$ then $\mathcal{B}$ wins the decaps-OW-CPA game. Hence, we have that

$$\Pr[\Upsilon_2] \leq \Pr[\text{decaps-OW-CPA}_{\mathsf{sKEM}_0}(\mathcal{B}) \Rightarrow 1] .$$

Collecting the probabilities concludes the proof. □

A.3 Proof in the ROM

The proof follows a similar idea as the one in the QROM.

Game Γ_0: This is the same as the UNF-1KCA game with $\mathsf{sKEM} = \mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}(\mathsf{sKEM}_0)$, except we assume there is no collision on H'. Thus, Γ^0 is the same as UNF-1KCA except with probability at most $\frac{q_{H'}^2}{2^n}$.

Game Γ_1: In this game, we return 0 if $\mathcal{A}$ did not query $H'(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A})$. As we can assume $(\mathsf{pk}, \mathsf{ct}) \neq (\mathsf{pk_A}, \mathsf{ct}')$, this changes the probability of $\mathcal{A}$ winning only if $\mathcal{A}$ outputs $t' = H'(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A})$ without having made the oracle query. Since the query was not made, one can actually lazy sample the value of $H'(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}', K_\mathsf{A})$ after $\mathcal{A}$ returns t', and the probability both values are equal is $\frac{1}{2^n}$. Hence,

$$\Pr[\Gamma_0] - \Pr[\Gamma_1] \leq \frac{1}{2^n} .$$

```
$\mathcal{O}(\mathcal{L}_{H'}, \{(\mathsf{pk}_i, (\mathsf{ct}_i, t_i))\}_{i=1}^{d})$
1 : for $i \in [d]$ :
2 :     $K'_i \leftarrow \mathsf{Decaps}(\mathsf{pk}_i, \mathsf{sk}_\mathsf{A}, \mathsf{ct}_i)$
3 :     if $((\mathsf{pk}_i, \mathsf{ct}_i, K'_i), t_i) \notin \mathcal{L}_{H'}$ : return 0
4 : return 1
```

Figure 22: Oracle $\mathcal{O}$ used in the proof of Theorem 6.

Game Υ_1: If Γ_1 outputs 1, it means $\mathcal{A}$ outputs (ct, t') s.t. $((\mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}', K_\mathsf{A}), t')$ is in the list of queries made by the $\mathcal{A}$. Hence, if that happens, one can find ct' and K_A s.t. $\mathsf{Decaps}(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{A}, \mathsf{ct}') = K_\mathsf{A}$ by running $(\mathsf{ct}', t') \leftarrow\!\!\$\ \mathcal{A}$ and looking for t' in the list of queries (note that we assume there is no collision). Therefore, it means one can build an adversary that wins the game Υ_1 in Fig. 21, and we have

$$\Pr[\Gamma_1] \leq \Pr[\Upsilon_1]\ .$$

Game Υ_2: We modify the game s.t. the tag t and the key given to the adversary are picked uniformly at random as shown in Figure 21. Both games are indistinguishable unless $\mathcal{A}$ queries $(\mathsf{pk}, \mathsf{pk}_\mathsf{B}, \mathsf{ct}, K_\mathsf{B})$ to H or H'. Then, an adversary $\mathcal{B}$ playing Υ_2 can perfectly simulate $\mathcal{A}$'s view in Υ_1 if it guesses correctly which query it is going to be and if such a query is going to happen. Overall, $\mathcal{B}$ can make a correct guess with probability $\frac{1}{q_{H'}+q_H+1}$. If that happens though, one can build an OW-CPA adversary $\mathcal{B}$ against sKEM_0 that runs $\mathcal{A}$ and picks a random query made by $\mathcal{A}$ to H or H'. Hence, we have

$$\Pr[\Upsilon_1] \leq (q_{H'} + q_H + 1)\Pr[\Upsilon_2]\ .$$

Finally, one can see that Υ_2 is the same as the decaps-OW-CPA for sKEM_0 if we omit the random values K and t and the more restrictive winning condition $(\mathsf{pk}_\mathsf{A}, \mathsf{ct}') \neq (\mathsf{pk}, \mathsf{ct})$. Hence, one can build an adversary $\mathcal{C}$ such that

$$\Pr[\Upsilon_2] \leq \Pr[\text{decaps-OW-CPA}_{\mathsf{sKEM}_0}(\mathcal{C}) \Rightarrow 1]\ .$$

□

B Proof of Theorem 6

B.1 Proof in the ROM

Proof. The idea of the proof is very similar to the IND-qCCA proof of the T_CH transform by Huguenin-Dumittan et al. [HV22] and is the following. Either all tags in the decapsulation query are valid and thus they are the form $H'(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_i, \mathsf{ct}_i, K'_i)$, or the oracle returns $\perp$. Then, if they are valid, with high probability the adversary queried $(\mathsf{pk}_\mathsf{A}, \mathsf{pk}_i, \mathsf{ct}_i, K'_i)$ to H' and thus K'_i can be recovered from the list of queries to the RO, i.e. the decapsulation oracle can be simulated without the knowledge of sk_A. In other words, the only information leaked by a query to the decapsulation oracle is whether *all* tags are valid or not, i.e. 1 bit of information, which is not sufficient to break the OW-CPA game. We prove this formally with a sequence of hybrid games.

Game Γ_0: This is the IND-1BatchCCA game with $\mathsf{sKEM} = \mathsf{T}_\mathsf{CH}^\mathsf{skem}(\mathsf{sKEM}_0)$.

Game Γ_1: We modify the previous game s.t. we abort if the adversary finds any collision when querying H'. We have that

$$\Pr[\Gamma_0] - \Pr[\Gamma_1] \leq \frac{q_{H'}^2}{2^n}$$

where q'_H is the number of queries the adversary makes to H'.
Game Γ_2: We modify the game s.t. it aborts if $\text{BatchDec}(\{(\mathsf{pk}_i, (\mathsf{ct}_i, t_i))\}_{i=1}^{d})$ does not return $\perp$ but one of the tags t_i was not obtained through an adversary's query to H'. The probability that some tag t_i is valid but $H'(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, K'_i)$ (with $(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, t_i) \neq (\mathsf{pk_A}, \mathsf{pk}_B, \mathsf{ct}^*, t_i^*)$) was not queried by the adversary is $\frac{1}{2^n}$. Hence, overall we have

$$\Pr[\Gamma_1] - \Pr[\Gamma_2] \leq \frac{d}{2^n}$$

Game Γ_3: We now change the game as follows. We record all queries to H' of the form $(\mathsf{pk_A}, \cdot, \cdot, \cdot)$ made by the adversary in a list $\mathcal{L}_{H'} = \{((\mathsf{pk}_j, \mathsf{ct}_j, K_j), h_j)\}_{j=1}^{q'_H}$ s.t. $H'(\mathsf{pk_A}, \mathsf{pk}_j, \mathsf{ct}_j, K_j) = h_j$ for all $j \in [q_{H'}]$. Then, the BatchDec oracle is modified as follows. If some tag t_i is s.t. for all $K \in \mathcal{K}$ $((\mathsf{pk}_i, \mathsf{ct}_i, K), t_i) \notin \mathcal{L}_{H'}$ then $\perp$ is returned. Then, $\mathcal{O}(\mathcal{L}_{H'}, \{(\mathsf{pk}_i, (\mathsf{ct}_i, t_i))\}_{i=1}^{d}) \to r$ is queried, where $\mathcal{O}$ is defined in Figure 22. If $r = 0$ BatchDec outputs $\perp$, otherwise it outputs $H(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, K_i)$ for all $i \in [d]$, where K_i is s.t. $((\mathsf{pk}_i, \mathsf{ct}_i, K_i), t_i) \in \mathcal{L}_{H'}$. Note that all these modifications are only syntactical as $\mathcal{O}$ outputs 1 iff for all $i \in [d]$, K_i is (the unique) value in $\mathcal{L}_{H'}$ s.t. $H'(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, K_i := \mathsf{Decaps}(\mathsf{pk}_i, \mathsf{sk_A})) = t_i$. Hence, we have

$$\Pr[\Gamma_2] = \Pr[\Gamma_3] .$$

Game Γ_4: We replace the challenge tag t^* and the real key K_0 by random values. This change can only be noticed if the adversary or the BatchDec oracle queries $H(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}^*, K^*)$ or $H'(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{ct}^*, K^*)$ at some point in the game. Let QUERY be this event. We show that if QUERY occurs, then one can break the OW-CPA security of sKEM_0 with high probability. The reduction works as follows. The OW-CPA adversary $\mathcal{B}$ receives a challenge ciphertext ct^* and public keys $\mathsf{pk_A}, \mathsf{pk_B}$ from its own challenger. Next, it samples random values K, t^* and passes all these to the IND-1BatchCCA adversary $\mathcal{A}$. Then, $\mathcal{B}$ can simulate everything in BatchDec (except the oracle call to $\mathcal{O}$) by recording $\mathcal{A}$'s queries to H'. In order to simulate $\mathcal{O}$, $\mathcal{B}$ samples a bit r at random instead, which succeeds with probability $\frac{1}{2}$. Finally, it samples at random a query made by $\mathcal{A}$ to H or H' or a query made to H by itself, and it outputs the key K that was part of this query. Overall, the simulation is correct with probability $\frac{1}{2}$ and if QUERY occurs $\mathcal{B}$ recovers K^* with probability $\frac{1}{q_H + q_{H'} + d}$. Hence,

$$\Pr[\Gamma_3] - \Pr[\Gamma_4] \leq \Pr[\mathsf{QUERY}] \leq 2(q_H + q_{H'} + d)\mathsf{Adv}_{\mathsf{sKEM}}^{\text{ow-cpa}}(\mathcal{A}) .$$

Finally, we see that the adversary's view is independent of b in Γ_4, therefore $\Pr[\Gamma_4] = \frac{1}{2}$. This concludes the proof. □

B.2 Proof in the QROM

Proof. As in the proof of Theorem 5, we use the extractable RO-simulator by Don et al. [Don+22] and we proceed with a sequence of hybrid games. We note the proof is nearly identical to the QROM IND-qCCA proof of $\mathsf{T_{CH}}$ [HV22] and we refer the reader to the latter for a detailed explanation of the game transitions.

Game Γ_0: This is the IND-1BatchCCA game with $\mathsf{sKEM} = \mathsf{T}_{\mathsf{CH}}^{\mathsf{skem}}(\mathsf{sKEM}_0)$. We also assume that the adversary only makes queries of the form $(\mathsf{pk_A}, \cdot, \cdot, \cdot)$ to the oracles. This has no consequence on the winning probability of the adversary as other type of queries are independent of the key.

Game Γ_1: We modify the BatchDec oracle s.t. it returns $\perp$ whenever the list of $(\mathsf{pk}_i, \mathsf{ct}_i, t_i)$ in the query contains $(\mathsf{pk}_i, \mathsf{ct}_i) = (\mathsf{pk_B}, \mathsf{ct}^*)$ (and thus $t_i \neq t^*$). This change has no impact except if $\mathsf{Decaps}(\mathsf{pk_B}, \mathsf{sk_A}, \mathsf{ct}^*) \neq K_0$, where K_0 is the challenge real key. In turn, this would imply that ct^* is an incorrect ciphertext. Hence,

$$\Pr[\Gamma_0] - \Pr[\Gamma_1] \leq \delta \ .$$

<u>Game Γ_2</u>: Now, we split the random oracle H' into two oracles H'_0 and H'_1 s.t.

$$H'(\mathsf{pk_A}, \mathsf{pk}, \mathsf{ct}, K) := \begin{cases} H'_0(K), & \text{if } (\mathsf{pk}, \mathsf{ct}) = (\mathsf{pk_B}, \mathsf{ct}^*) \\ H'_1(\mathsf{pk}, \mathsf{ct}, K), & \text{otherwise} \end{cases}$$

and we give the adversary access to H'_0, H'_1 instead of H'. We also switch to the RO simulator instead of using H'_1. In addition, at the end of the game, the challenger calls the extractor on all tags t_i queried as part of the call to the BatchDec oracle to obtain extracted values $(\mathsf{pk}^\mathsf{e}_i, \mathsf{ct}^\mathsf{e}_i, K^\mathsf{e}_i)$, $i \in [d]$. Note that H'_0 is never called as part of a BatchDec query due to the modification in the previous game. These changes have no impact on the success of the game and thus

$$\Pr[\Gamma_1] = \Pr[\Gamma_2] \ .$$

<u>Game Γ_3</u>: We abort whenever the decapsulation oracle does not return $\perp$ but the extracted values $(\mathsf{pk}^\mathsf{e}_i, \mathsf{ct}^\mathsf{e}_i, K^\mathsf{e}_i)$ are not equal to $\perp$ or $(\mathsf{pk}_i, \mathsf{ct}_i, K'_i)$, where $K'_i = \mathsf{Decaps}(\mathsf{pk}_i, \mathsf{sk_A}, \mathsf{ct}_i)$. By Property 8 of the extractable oracle, we have

$$\Pr[\Gamma_2] - \Pr[\Gamma_3] \leq \frac{40e^2(q_{H'} + d + 1)^3 + 2}{2^n} \ .$$

<u>Game Γ_4</u>: We move the extraction to the BatchDec oracle, right after the corresponding tag verification. By Property 4 of the extractable oracle, we have

$$\Pr[\Gamma_3] - \Pr[\Gamma_4] \leq 8d(d + q_{H'})\sqrt{2/2^n} \ .$$

<u>Game Γ_5</u>: We modify the BatchDec oracle s.t. we abort if all tag checks pass, i.e. $H'(\mathsf{pk}_i, \mathsf{ct}_i, K'_i) = t_i, \forall i \in [d]$ but some extracted value is equal to $\perp$, i.e. $(\mathsf{pk}^\mathsf{e}_i, \mathsf{ct}^\mathsf{e}_i, K^\mathsf{e}_i) = \perp$ for some $i \in [d]$. By Property 7 of the extractable oracle we have

$$\Pr[\Gamma_4] - \Pr[\Gamma_5] \leq d\frac{2}{2^n} \ .$$

<u>Game Γ_6</u>: We modify the BatchDec oracle s.t. the queries to H' made for the tag verification are made after the corresponding extraction. By Property 8 of the extractable oracle we have

$$\Pr[\Gamma_5] - \Pr[\Gamma_6] \leq 8d\sqrt{2/2^n} \ .$$

<u>Game Γ_7</u>: We modify the previous game as follows. Let r be a bit set to 1 iff for all $i \in [d]$ $(\mathsf{pk}^\mathsf{e}_i, \mathsf{ct}^\mathsf{e}_i) = (\mathsf{pk}_i, \mathsf{ct}_i)$ and $\mathsf{Decaps}(\mathsf{pk}^\mathsf{e}_i, \mathsf{sk_A}, \mathsf{ct}^\mathsf{e}_i) = K^\mathsf{e}_i$. Then, we change BatchDec s.t. it returns $\perp$ if $r = 0$, otherwise it returns $H(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, K^\mathsf{e}_i)$ for all $i \in [d]$. In addition, the tag verification is now skipped. We argue this affects only negligibly the advantage of the adversary compared to the previous game:

- If BatchDec returns $H(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, K'_i)$, $i \in [d]$ in Γ_6, then by the previous changes we know that $(\mathsf{pk}^\mathsf{e}_i, \mathsf{ct}^\mathsf{e}_i, K^\mathsf{e}_i) = (\mathsf{pk}_i, \mathsf{ct}_i, K'_i)$ for all $i \in [d]$, therefore BatchDec returns $H(\mathsf{pk_A}, \mathsf{pk}_i, \mathsf{ct}_i, K'_i)$, $i \in [d]$ in Γ_7 as well.

- If BatchDec returns $H(\mathsf{pk}_{\mathsf{A}}, \mathsf{pk}_i, \mathsf{ct}_i, K_i^{\mathsf{e}})$ $, i \in [d]$ in Γ_7, we know that $(\mathsf{pk}_i^{\mathsf{e}}, \mathsf{ct}_i^{\mathsf{e}}, K_i^{\mathsf{e}}) = (\mathsf{pk}_i, \mathsf{ct}_i, K_i')$. In addition, for each $i \in [d]$, $t_i = H(\mathsf{pk}_i^{\mathsf{e}}, \mathsf{ct}_i^{\mathsf{e}}, K_i^{\mathsf{e}})$ with probability $1 - \frac{2}{2^n}$ by Property 6 of the extractable oracle. Therefore, the tag verification would pass in Γ_6 with high probability and BatchDec would return the same values in that game as well.

Overall, we have

$$\Pr[\Gamma_6] - \Pr[\Gamma_7] \leq d\frac{2}{2^n} .$$

<u>Game Γ_8</u>: Now we move all d queries to H' made in BatchDec to the end of the game. By Property 8 of the extractable oracle, we have

$$\Pr[\Gamma_7] - \Pr[\Gamma_8] \leq 8dq_{H'}\sqrt{2/2^n} .$$

Note that we can now forget about the queries to H' we just moved to the end of the game.

<u>Game Γ_9</u>: We replace the real key K_0 and the challenge tag t^* by random values. We have $\Pr[\Gamma_9] = \frac{1}{2}$. Applying the OW2H lemma on $H \otimes H_0'$, we get

$$\Pr[\Gamma_8] - \Pr[\Gamma_9] \leq 2(q_{H'} + q_H + d)\sqrt{\Pr[\Upsilon]}$$

where Υ is the same as Γ_9, except the challenger measures a random query made to $H \otimes H_0'$ and outputs 1 iff the query contains K^*, where K^* is the key encapsulated in ct^*. We can build an OW-CPA adversary $\mathcal{B}$ against sKEM_0 that wins with high probability when Υ outputs 1. The reduction works nearly as in the ROM proof: $\mathcal{B}$ receives a challenge ciphertext ct^* and two public keys $\mathsf{pk}_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}$, then it samples t^* and K^* at random and passes all these values to $\mathcal{A}$. Then, $\mathcal{B}$ can perfectly simulate BatchDec as in Γ_9, except for the bit r that it can guess correctly with probability $\frac{1}{2}$. Finally, $\mathcal{B}$ measures a random query that was made to H or H' in the execution and outputs the corresponding value K. Overall, we have

$$\Pr[\Upsilon] \leq 2\mathsf{Adv}_{\mathsf{sKEM}}^{\text{ow-cpa}}(\mathcal{A})$$

which concludes the proof. □

C Differences from the Proceedings Version

Apart from providing all relevant proofs and formal statements where relevant and including additional text, we have updated this work since submitting the camera-ready version to USENIX Security 2024 [Col+24]. In particular, our protocol was not proven secure in the presence of key-compromise impersonation (KCI) attacks because we disallowed long-term key exposure of a non-partnered receiver being tested in the KIND game, and now is. The following attack also was not therefore considered before: if a malicious party chooses a prekey bundle adversarially with a chosen split-KEM public key, then a sender who used that prekey could potentially learn some information about the sender's ephemeral split-KEM secret. In this revised document, we strengthened our decaps-OW-CPA and UNF-1KCA (previously UNF-1KMA) notions to allow the encapsulation call to be with respect to a chosen (potentially malicious) public key and capture these attacks. The corresponding proofs were changed accordingly. In particular, for the decaps-OW-CPA proof of our split-KEM, we now simply use a guessing technique that implies a security loss of 64 bits. This has the advantage of removing the need for rejection sampling without affecting our security claims. We also included a table comparing our (updated) protocol with the state-of-the-art in the introduction and made other minor changes including correcting some minor mistakes and typos.