Challenges in Proving Post-Quantum Key Exchanges Based on Key Encapsulation Mechanisms

Jacqueline Brendel[1], Marc Fischlin[1], Felix Günther[2], Christian Janson[1], and Douglas Stebila[3]

[1] *Technische Universität Darmstadt, Darmstadt, Germany*
[2] *Department of Computer Science, ETH Zürich, Zürich, Switzerland*
[3] *University of Waterloo, Waterloo, Canada*

November 26, 2019

Abstract

Modern key exchange protocols are usually based on the Diffie–Hellman (DH) primitive. The beauty of this primitive, among other things, is its potential reusage of key shares: DH shares can be either used once as an ephemeral key or used in multiple runs as a (semi-)static key. Since DH-based protocols are insecure against quantum adversaries, alternative solutions have to be found when moving to the post-quantum setting. However, most post-quantum candidates, including schemes based on lattices and even supersingular isogeny DH, are not known to be secure under key reuse. In particular, this means that they cannot be necessarily deployed as an immediate DH substitute in protocols.

In this paper, we introduce the notion of a *split key encapsulation mechanism* (split KEM) to translate the desired properties of a DH-based protocol, namely contributiveness and key-reusability, to a KEM-based protocol flow. We provide the relevant security notions of split KEMs and show that the formalism lends itself to lift Signal's X3DH to the post-quantum KEM setting. While the proposed framework conceptually solves the raised issues, we did not succeed in providing a strongly-secure, post-quantum instantiation of a split KEM yet. The intention of this paper hence is to raise further awareness of the challenges arising when moving to KEM-based key exchange protocols with contributiveness and key-resusability, and to enable others to start investigating potential solutions.

Contents

1 Introduction

The core Diffie–Hellman protocol [13]—Alice sends g^x, Bob sends g^y, and both compute g^{xy} as shared secret—is a beautiful and versatile cryptographic primitive, and plays a central role in modern key exchange protocols and appears in many facets. For example, a key share (x, g^x) can be ephemeral (used once) or static (reused multiple times), and the same share can be used in role-symmetric ways, i.e., as both initiator and responder of key exchange sessions. Furthermore, the same message flow can give rise to different AKE protocols (e.g., HMQV [35], CMQV [56], NAXOS [39]). The security of DH-based constructions can in turn be based on many cryptographic assumptions over the group, ranging from simple passive assumptions like CDH or DDH, to interactive assumptions like GapDH [45], oracle DH (ODH) [1] or PRF-ODH [32, 6].

Indeed, modern real-world cryptographic protocols employ the Diffie–Hellman key exchange protocol in ways that often rely on many aspects of this versatility. Table 1 shows key exchange patterns from various Internet protocols that employ a "signed ephemeral Diffie–Hellman" approach. In TLS 1.2 [12], the server sends the initial ephemeral public key, and the client responds, while in TLS 1.3 [52], the roles are reversed to reduce the number of round trips: the client sends the initial ephemeral public key, and the server responds. In both cases the security proofs [32, 22] rely on interactive DH assumptions (variants of PRF-ODH) because of how the session key is used in the protocol prior to the session being fully authenticated.

In implicitly authenticated key exchange, static key pairs are used to derive shared secrets that can only be computed by the intended parties; having a peer who successfully computes the shared secret implicitly authenticates that peer, in contrast to the explicit authentication provided by checking a signature computed by one's peer. Implicitly authenticated key exchange protocols have long been of academic interest [35, 9, 58], and have recently started to be used in real-world protocols, such as the original handshake design of Google's QUIC protocol [51, 41] or the Signal protocol [54, 7], as well as OPTLS [37] which is the conceptual foundation of the TLS 1.3 handshake (cf. Table 2).

Protocol	**Core message flow**	**Session key**	**Security**
SSHv2 signed ephemeral DH	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{\texttt{hello}}$ $\xrightarrow{epk_A}$ $\xleftarrow{epk_B, lpk_B, \texttt{sig}}$	$\mathsf{DH}(epk_A, epk_B)$	DDH [4]
TLS 1.2 signed ephemeral DH	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{epk_B, \texttt{cert}(lpk_B), \texttt{sig}}$ $\xrightarrow{epk_A}$	$\mathsf{DH}(epk_A, epk_B)$	snPRF-ODH [32]
TLS 1.3 signed ephemeral DH	$\xrightarrow{\texttt{hello}, epk_A}$ $\xleftarrow{epk_B, \texttt{cert}(lpk_B), \texttt{sig}}$	$\mathsf{DH}(epk_A, epk_B)$	snPRF-ODH [22]

Table 1: Signed Diffie–Hellman key exchange patterns of selected Internet protocols. With epk_X and lpk_X we denote the ephemeral resp. long-term key of a party, `hello` is the protocol's initiator message, `sig` a signature under the long-term key, and $\texttt{cert}(lpk_X)$ the long-term key and certificate of party X.

Protocol	Core message flow	Session key	Security
TLS 1.2 [12] (implicitly-auth static Diffie–Hellman + explicit-auth MAC)	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{\texttt{cert}[lpk_B],\texttt{mac}}$ $\xrightarrow{epk_A,\texttt{mac}}$	$\mathsf{DH}(epk_A, lpk_B)$	mnPRF-ODH [36]
OPTLS [37] (TLS 1.3–style, implicitly-auth Diffie–Hellman + explicit-auth MAC)	$\xrightarrow{\texttt{hello},epk_A}$ $\xleftarrow{epk_B,\texttt{cert}[lpk_B],\texttt{mac}}$	$\mathsf{DH}(epk_A, epk_B)$ $\|$ $\mathsf{DH}(epk_A, lpk_B)$	GapDH, DDH [37] (random oracle model)
Signal [54] X3DH triple handshake [+ optional ephemeral-ephemeral]	$\xrightarrow{\texttt{hello}}$ $\xleftarrow{lpk_B, sspk_B, [epk_B]}$ $\xrightarrow{lpk_A, epk_A}$	$\mathsf{DH}(lpk_A, sspk_B)$ $\|$ $\mathsf{DH}(epk_A, lpk_B)$ $\|$ $\mathsf{DH}(epk_A, sspk_B)$ $\|$ $[\mathsf{DH}(epk_A, epk_B)]$	mmPRF-ODH, smPRF-ODH, smPRF-ODH, [snPRF-ODH] [7]
QUIC original handshake [41]	$\xrightarrow{\texttt{hello},epk_A}$ $\xleftarrow{sspk_B}$	$\mathsf{DH}(epk_A, lpk_B)$ $\|$ $\mathsf{DH}(epk_A, sspk_B)$	GapDH [25] (random oracle model)

Table 2: Implicitly authenticated Diffie–Hellman key exchange patterns of selected Internet protocols. With epk_X and lpk_X we denote the ephemeral resp. long-term key of a party, `hello` is the protocol's initiator message, `mac` a message authentication code under a derived key, and $\texttt{cert}(lpk_X)$ the long-term key and certificate of party X.

Moving to post-quantum solutions. Unfortunately, DH-based protocols are not secure against quantum adversaries and thus, also key exchange protocols need to undergo the transition to post-quantum–secure designs. The NIST Post Quantum Cryptography Standardization process [44] is currently in the second round for identifying quantum-resistant primitives. Furthermore, experimental deployment of post-quantum algorithms in key exchange protocols has already taken place, e.g., by Google, Cloudflare, and the Open Quantum Safe project [40, 38, 55, 10].

While key exchange protocols are a crucial building block for many applications, the NIST standardization process did not explicitly ask for key exchange, but for the conceptually simpler notions of key encapsulation mechanisms (KEMs) [53, 8]. A KEM allows encapsulation of a symmetric key within a ciphertext under some public key, such that the symmetric key can be decapsulated again (only when) knowing the corresponding secret key. Security-wise, the ciphertext hides the encapsulated symmetric key indistinguishably from a random string. The proposals to the NIST standardization process mostly follow the approach to first provide a (weakly secure) public-key encryption scheme and then use well-known transforms such as the Fujisaki–Okamoto transform [27, 28, 31] to achieve a strongly-secure KEM.

In principle, KEMs can directly be used to build and analyze key exchange protocols, and allow to capture (implicitly authenticated) Diffie–Hellman flows (e.g., in the static Diffie–Hellman handshake of TLS 1.2 [36]). The naive approach hence would be to simply replace every DH combination in a key exchange protocol with a KEM combination. However, whereas DH shares can be freely reused by both parties, particularly allowing static-static combinations (as used, e.g., in Signal [54]), the KEM concept restricts reuse to one side—the decpasulator. This in turn limits the possible message flows and contributions of ephemeral randomness that standard KEMs can capture, hindering a direct translation of DH-type protocols to KEM-based designs, and leading to the question:

Can we capture post-quantum KEM designs in a way that enable flexible key reuse and support efficient message flows similar to the widely-used Diffie–Hellman-based designs?

1.1 Our Contribution

In this note we introduce the notion of *split KEMs* which provide the contributiveness and key-reusability features of Diffie–Hellman-based key exchanges in the KEM setting,thus enabling optimized message flows.

We transfer the relevant security notions for KEMs (indistinguishability and one-wayness) to the split KEM setting and show how split KEMs enable the smooth transfer of popular DH-based key exchanges such as Signal's X3DH to the (post-quantum) KEM setting. Especially in the case of the Signal protocol, the complex initial key agreement (via X3DH) has been abstracted away as an idealized primitive in works on secure messaging with and without post-quantum security considerations (cf., e.g., the recent work amenable to post-quantum security by Alwen, Coretti, and Dodis [2]).

As for realizing the split KEM notion, we show that LWE-based KEMs based on reconciliation do non-trivially match the split KEM *structure*. However, key re-use attacks against LWE-based KEMs mean that such an instantiation would not be secure, and we have been unable to successfully develop a provably secure instantiation from a post-quantum assumption. We identify this as an important challenge for future work.

1.2 Related Work

Related work for our split KEM notion includes works both specifically on the post-quantum security of concrete protocols, as well as foundational extensions to the definitional framework of KEMs.

1.2.1 Post-quantum Signal

Alwen, Coretti, and Dodis [2] gave a first variant of the double-ratchet of Signal that is amenable to take post-quantum secure KEMs as building blocks. However, their work omits the crucial initial key agreement between the participants and thus their solution does not result in a full-fledged post-quantum Signal protocol. Our focus in this note is the transferring the initial DH-based key agreement to the KEM setting.

In [24], it is explored how Signal can be transitioned to the post-quantum setting. While [24] correctly identified the main challenges when moving from DH-based key exchanges to KEMs, they claim that X3DH can be made post-quantum by replacing the DH operations with ones based on Supersingular Isogeny DH (SIDH) [33, 11]. However, as we will elaborate in the next paragraph, SIDH is not secure under key reuse and can thus unfortunately not be used as a drop-in replacement for DH in X3DH.

1.2.2 Key reuse with LWE and SIDH

There are a number of attacks on lattice-based key exchange schemes when keys are reused [26, 14, 15, 19, 42, 17, 50, 3, 18]. There have been proposals to enable secure key reuse in LWE-based schemes [30, 16]. However, these proposals only seem to at most guard against specific attacks at a time, while still being vulnerable to other attacks.

Similarly, for key exchange based on SIDH [33] there are attacks when keys are reused [29]. A recent proposal to enable reuse of keys [34] was unfortunately found to be still insecure [20, 21] such that, at this point, there is no supersingular-isogeny-based solution (apart from FO-transformed KEMs such as SIKE [11]) that supports key reuse.

1.2.3 Double-key KEMs

Xue et al. [57] introduced the notion of double-key KEMs, where both encapsulation and decapsulation take two public resp. secret keys as input. This notion aims to capture implicitly authenticated key exchange constructions based on KEMs, where typically encapsulations under the static secret of a party ensure authentication, while encapsulations under the party's ephemeral key ensure fresh contributions to the key agreement. The notion of double-key KEMs transfers this idea into a single encapsulation and decapsulation operation. The two input keys belong to the *same* party, i.e., encapsulation takes two public keys and decapsulation takes two secret keys of the same party. In the split KEM setting however, encapsulation and decapsulation take *inputs from both parties*—a public key input of the peer and a secret key input of the party executing the respective algorithm.

1.2.4 Merged KEMs

Drucker and Gueron [23] recently introduced the notion of *merged* KEMs (MKEMs) as a primitive for continuous key agreement (CKA) (a.k.a. public-key ratchets) in double ratchets as employed for example in the Signal protocol. In KEM-based CKA [49, 2], the encapsulator encapsulates a shared key and then sends the corresponding ciphertext along with an updated public key to the decapsulator. In merged KEMs this public key update and encapsulation under the "old" public key are merged to save on bandwidth. When decapsulating a ciphertext, the output is the shared secret as well as the updated public key. Our work on split KEMs applies to the stage before symmetric ratcheting or CKA come into play; split KEMs are concerned with the initial key agreement with contributions from both parties.

2 Preliminaries

We begin by briefly introducing the notation we require throughout this paper. Since our main concept builds upon the notion of KEMs, we subsequently review their basic syntax and security notions.

2.1 Notation

For an algorithm A we write $y \leftarrow \mathsf{A}(\cdot)$, resp. $y \xleftarrow{\$} \mathsf{A}(\cdot)$, for determinisitically, resp. probabilistically, running A on given inputs and assigning the output to y. PPT denotes probabilistic polynomial-time and λ the security parameter. By $\mathcal{A}^{\mathcal{O}}$ we express that the adversary denoted by $\mathcal{A}$ is given access to oracle $\mathcal{O}$. Finally, we use $\perp$ as a special symbol to denote rejection or an error, and we assume that $\perp \notin \{0,1\}^*$.

2.2 Key Encapsulation Mechanisms

Definition 1. A *key encapsulation mechanism* KEM with associated public key space $\mathcal{PK}$, secret key space $\mathcal{SK}$, ciphertext space $\mathcal{C}$, and key space $\mathcal{K}$ is a tuple of algorithms $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ defined as follows.

Key generation KGen**:** This probabilistic algorithm takes as input the security parameter λ and outputs a public-private key pair in $\mathcal{PK} \times \mathcal{SK}$, i.e., $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$.

Encapsulation Encaps**:** This probabilistic algorithm takes as input a public key pk and outputs a ciphertext $c \in \mathcal{C}$ and the therein encapsulated key $K \in \mathcal{K}$, i.e., $(c, K) \xleftarrow{\$} \mathsf{Encaps}(pk)$.

Decapsulation Decaps**:** This algorithm takes as input a ciphertext c and secret key sk and outputs $K' \in \mathcal{K} \cup \{\perp\}$, where $\perp$ indicates a decapsulation error, i.e., $K' \leftarrow \mathsf{Decaps}(sk, c)$.

$\underline{\mathcal{G}^{\mathsf{indcpa}}_{\mathsf{KEM},\mathcal{A}}(\lambda)}$:
1 $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $(c^\star, K_0^\star) \xleftarrow{\$} \mathsf{Encaps}(pk)$
3 $K_1^\star \xleftarrow{\$} \mathcal{K}$
4 $b \xleftarrow{\$} \{0,1\}$
5 $b' \xleftarrow{\$} \mathcal{A}(pk, c^\star, K_b^\star)$
6 **return** $[\![b' = b]\!]$

$\underline{\mathcal{G}^{\mathsf{indcca}}_{\mathsf{KEM},\mathcal{A}}(\lambda)}$:
1 $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $(c^\star, K_0^\star) \xleftarrow{\$} \mathsf{Encaps}(pk)$
3 $K_1^\star \xleftarrow{\$} \mathcal{K}$
4 $b \xleftarrow{\$} \{0,1\}$
5 $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{dec}}}(pk, c^\star, K_b^\star)$
6 **return** $[\![b' = b]\!]$

$\underline{\mathcal{O}_{\mathsf{dec}}(c)}$:
7 **if** $c = c^\star$
8 **return** $\perp$
9 **else**
10 **return** $\mathsf{Decaps}(sk, c)$

Figure 1: IND-CPA and IND-CCA security games for $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ with key space $\mathcal{K}$.

$\underline{\mathcal{G}^{\mathsf{owcpa}}_{\mathsf{KEM},\mathcal{A}}(\lambda)}$:
1 $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $(c^\star, K^\star) \xleftarrow{\$} \mathsf{Encaps}(pk)$
3 $K' \xleftarrow{\$} \mathcal{A}(pk, c^\star)$
4 **return** $[\![K' = K^\star]\!]$

$\underline{\mathcal{G}^{\mathsf{owcca}}_{\mathsf{KEM},\mathcal{A}}(\lambda)}$:
1 $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 $(c^\star, K^\star) \xleftarrow{\$} \mathsf{Encaps}(pk)$
3 $K' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{dec}}}(pk, c^\star)$
4 **return** $[\![K' = K^\star]\!]$

$\underline{\mathcal{O}_{\mathsf{dec}}(c)}$:
5 **if** $c = c^\star$
6 **return** $\perp$
7 **else**
8 **return** $\mathsf{Decaps}(sk, c)$

Figure 2: OW-CPA and OW-CCA security games for $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ with key space $\mathcal{K}$.

We say that a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ is ϵ-*correct* if

$$\Pr\left[K' \neq K : (pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda), (c, K) \xleftarrow{\$} \mathsf{Encaps}(pk), K' \leftarrow \mathsf{Decaps}(sk, c)\right] \leq \epsilon.$$

We call KEM *(perfectly) correct* if $\epsilon = 0$.

KEM Security

The security of key encapsulation mechanisms can be formulated in terms of an indistinguishability formulation as well as in terms of one-wayness. We first recap *indistinguishability* (of encapsulated keys), defined under either passive (*chosen-plaintext*, IND-CPA) or active (*chosen-ciphertext*, IND-CCA) attacks.

Definition 2. Let KEM be a key encapsulation mechanism with key space $\mathcal{K}$. We say that KEM is IND-CPA-*secure*, resp. IND-CCA-secure, if for every PPT adversary $\mathcal{A}$ the advantage function (for $\mathsf{atk} = \mathsf{cpa}$, resp. $\mathsf{atk} = \mathsf{cca}$) for winning the game $\mathcal{G}^{\mathsf{indatk}}_{\mathsf{KEM},\mathcal{A}}$ from Figure 1, defined as

$$\mathsf{Adv}^{\mathsf{indatk}}_{\mathsf{KEM},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathcal{G}^{\mathsf{indatk}}_{\mathsf{KEM},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|,$$

is negligible in the security parameter λ.

Secondly, *one-wayness* aims at (non-)recoverability of the encapsulated key; again defined under either passive (*chosen-plaintext*, OW-CPA) or active (*chosen-ciphertext*, OW-CCA) attacks.

Definition 3. Let KEM be a key encapsulation mechanism with key space $\mathcal{K}$. We say that KEM is OW-CPA-*secure*, resp. OW-CCA-secure, if for every PPT adversary $\mathcal{A}$ the advantage function (for $\mathsf{atk} = \mathsf{cpa}$, resp. $\mathsf{atk} = \mathsf{cca}$) for winning the game $\mathcal{G}^{\mathsf{owatk}}_{\mathsf{KEM},\mathcal{A}}$ from Figure 2, defined as

$$\mathsf{Adv}^{\mathsf{owatk}}_{\mathsf{KEM},\mathcal{A}}(\lambda) := \Pr\left[\mathcal{G}^{\mathsf{owatk}}_{\mathsf{KEM},\mathcal{A}}(\lambda) = 1\right],$$

is negligible in the security parameter λ.

3 A KEM-based Instantiation of Signal's X3DH Key Exchange

In this section, we illustrate the challenges arising when translating DH-based key exchange protocols to the KEM setting following the example of Signal's X3DH[1] initial key exchange design [54] for secure messaging. We will see that KEMs are not equally contributive as DH-based key agreement: simply replacing DH operations with KEM encapsulations results in additional message flows and altered ephemeral/static share combinations for deriving keying material of the involved parties.

While purely ephemeral DH-based key exchanges generally map well to KEMs, many protocol designs further include DH-based combinations of static or semi-static keys with (semi-)static or ephemeral keys, most importantly for implicit authentication (as in Signal [54] or the Noise framework [47]). However, KEMs allow only for restricted key reuse (namely only on the decapsulator's side) and are hence limited in their support of static key share combinations.

3.1 X3DH: The Initial Key Agreement in Signal

X3DH [43] is part of the Signal secure messaging protocol [48] and establishes the initial keys. We limit the following discussion to this initial key exchange. For further information on the remaining cryptographic building blocks of the Signal protocol, especially on the ratcheting stages following X3DH, we refer the interested reader to, e.g., the analyses of Cohn-Gordon et al. [7] and Alwen et al. [2].

In Figure 3 we give an illustration of X3DH, where Alice wishes to establish a shared key with Bob. The session setup in Signal involves three parties, namely the communicating parties Alice and Bob, plus a central server S. This is due to the fact that Signal aims to provide secure messaging in an *asynchronous* setting, i.e., chats can be initiated and encrypted messages can be exchanged even if not *all* communication partners are online. For this, all users need to register their identity key and further cryptographic key material with the central server S. In more detail, every user U provides the server S with the public keys of the following key pairs:

- a long-lived static identity key pair (lpk_U, lsk_U),
- a medium-lived semi-static (signed) prekey pair $(sspk_U, sssk_U)$, and
- n ephemeral prekey pairs $(eppk_U^1, epsk_U^1), \ldots, (eppk_U^n, epsk_U^n)$.

When Alice wants to initiate a chat with Bob, she simply requests the necessary information and cryptographic key material of Bob from the central server S. From this she then derives an initial shared secret that secures her first message(s) to Bob. Once Bob comes online again and receives the first message from Alice (via the server), he requests Alice's cryptographic key material from the central server S to be able to derive the same initial key to decrypt Alice's message.

More formally, Alice initiates a session with Bob by first pinging the server S and requesting Bob's public key material: the static identity key lpk_B, the semi-static prekey $sspk_B$, as well as (optionally, if available) a single ephemeral prekey $eppk_B$. Alice then generates an ephemeral key pair (epk_A, esk_A) of her own and derives the master secret ms as

$$\mathsf{ms} \leftarrow sspk_B^{lsk_A} || lpk_B^{esk_A} || sspk_B^{esk_A} || eppk_B^{esk_A},$$

where the last DH value $eppk_B^{esk_A}$ is only present if Alice has received one of Bob's ephemeral prekeys $eppk_B$ from the server. More on this below in Remark 1.

Alice then derives the initial key K from the master secret via a pseudorandom function F keyed with ms and can then use this key to encrypt her first message to Bob. Finally, Alice sends her ephemeral

[1] Short for "Extended Triple Diffie-Hellman".

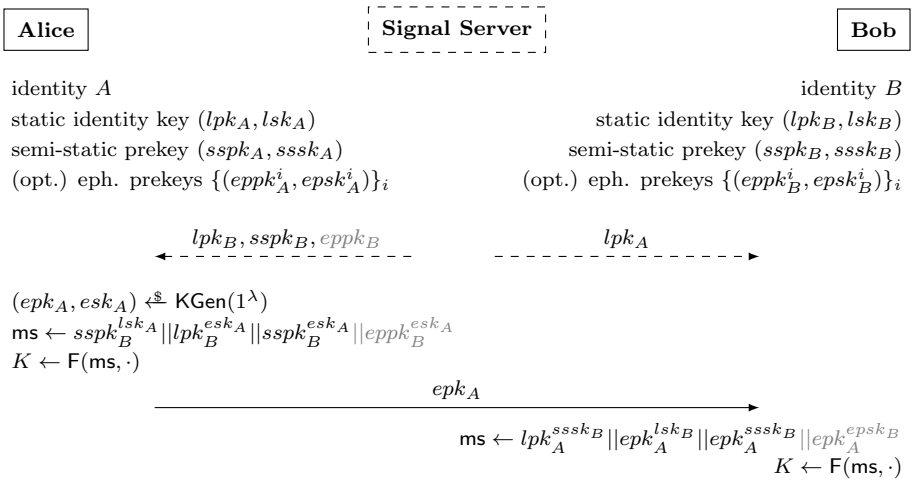

Figure 3: Signal's X3DH key exchange. Interaction with the Signal server is dashed, the optional ephemeral prekey (combination) is depicted in gray.

public key epk_A to Bob (alongside identifiers for, e.g., Bob's semi-static and ephemeral prekeys that she received from the server). Once Bob comes online he will receive this message and can then request Alice's static identity key lpk_A from the server. Analogously to Alice he can then compute the master secret ms and thus the final initial key K that decrypts Alice's message.

Remark 1 (Exhaustion of ephemeral prekeys). Note that each of the n stored ephemeral prekeys is only handed out once by the server, i.e., in case Charlie wishes to also initiate a session with Bob, he will receive an ephemeral prekey of Bob that is different from the one Alice received. However, in case many users initiate a session with Bob while he is offline, it may be the case that the stored ephemeral prekeys on the server are exhausted. In this case, the initial shared secret is only derived from the static identity key lpk_B and the semi-static prekey $sspk_B$.

3.2 A KEM-based X3DH Variant

Considering preparations for a post-quantum–secure messaging design, one may ask if any candidate of NIST's post-quantum cryptography process can be used smoothly in the above setting. Unfortunately this is not the case. As mentioned before, replacing the Diffie–Hellman operations in Signal's X3DH protocol with KEMs causes difficulties, as we illustrate in Figure 4 and discuss in the following.

As before, when Alice initiates a session with Bob, she requests and receives Bob's static identity key lpk_B, his semi-static prekey $sspk_B$, as well as a single ephemeral prekey $eppk_B$ (if available) from the server. Alice then separately encapsulates key material under each of these keys and sends the resulting ciphertexts to Bob, establishing three shared keys K_1, K_2, and K_3 (if available).

Yet, in order to fully transfer X3DH to the KEM setting, these three keys are not enough: they constitute, in order, the KEM analogues of the DH secrets $\mathsf{DH}(lpk_B, epk_A)$, $\mathsf{DH}(sspk_B, epk_A)$, and $\mathsf{DH}(eppk_B, epk_A)$, where the ephemeral contribution from Alice via epk_A is replaced by (differing) randomness inputs on Alice's side to the encapsulation algorithm. What is missing to complete the master secret computation—and thus key derivation—in the same fashion as in X3DH is the analogue of the DH combination of Alice's static identity key and Bob's semi-static key, i.e., $\mathsf{DH}(sspk_B, lpk_A)$.

Key encapsulation mechanisms, however, do not provide for a *non-ephemeral* contribution of the encapsulating party to the Encaps algorithm. In the KEM-based X3DH variant, Bob can thus at most

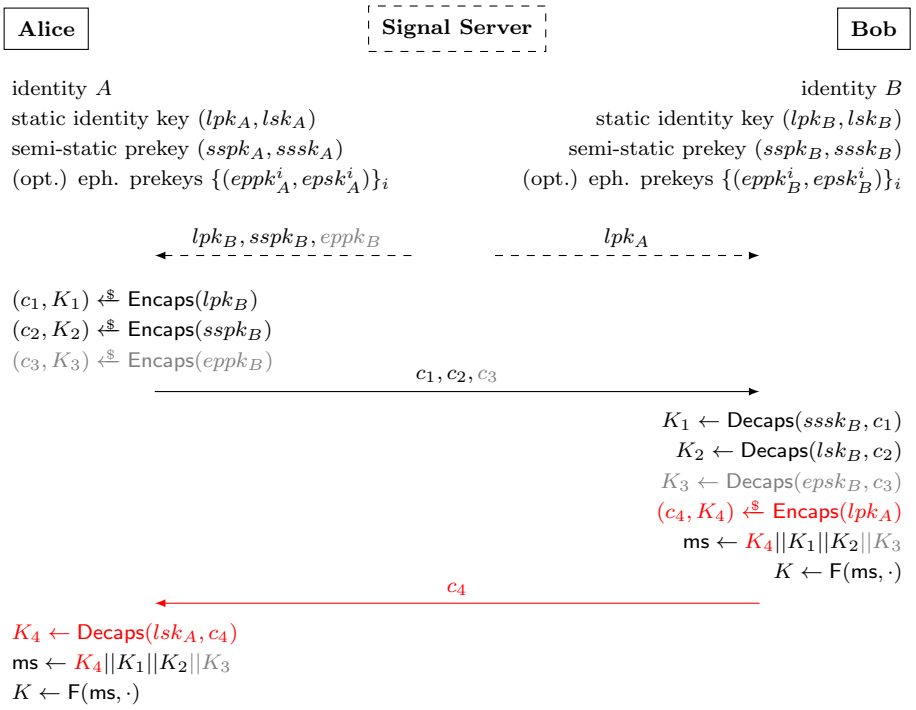

Figure 4: Signal's X3DH key exchange with KEMs replacing the DH operations. Interaction with the Signal server is dashed, the optional ephemeral prekey (combination) is depicted in gray. The last flow (in red), necessary for the key share combination involving Alice's long-term key, breaks the asynchronicity of X3DH.

encapsulate under Alice's static identity key lpk_A, which introduces an additional message flow (depicted in red in Figure 4). This however eradicates a key feature of instant messaging: asynchronicity, i.e., the ability to send encrypted messages even if the receiving party is offline.[2]

4 Split Key Encapsulation Mechanisms

To tackle the above mentioned issues of KEMs in a DH-based protocol, we introduce a new primitive called *split key encapsulation mechanism*, or split KEM, for short. Split KEMs enable a more fine-grained notion of key encapsulation mechanisms, where the encapsulation procedure is divided up into key generation and a subsequent shared key computation step. As it turns out, many proposals for key encapsulation mechanisms submitted to the NIST Post-Quantum Cryptography Standardization process [44], especially those based on lattices, are of the split KEM format (in their passively-secure version), i.e., their encapsulation procedure can be split into a key generation and a key agreement part.

4.1 Split KEM Definition

Intuitively, a split KEM is a KEM in which *both* parties can contribute to the encapsulation, with either ephemeral or (semi-)static keys. The key generation on the encapsulator's side (that does implicitly take place in many KEMs) is decoupled from the encapsulation algorithm, thus allowing key reuse and contributiveness similar to the DH setting.

Notation. Let enc denote the encapsulating party in a key encapsulation mechanism, in the following referred to as the *encapsulator* and similarly, dec denotes the decapsulating party, or *decapsulator*. Let $\mathcal{PK}_{\mathsf{enc}}$ and $\mathcal{SK}_{\mathsf{enc}}$ be the public and secret key space of the encapsulator, and $\mathcal{PK}_{\mathsf{dec}}$ and $\mathcal{SK}_{\mathsf{dec}}$ analogously for the decapsulator (if irrelevant or clear from the context, we will in the following omit the mention of these explicit key spaces). Let $\mathcal{C}$ be the ciphertext space and $\mathcal{K}$ the key space.

Definition 4. A *split KEM* sKEM consists of four algorithms $\mathsf{KGen}_{\mathsf{dec}}, \mathsf{KGen}_{\mathsf{enc}}, \mathsf{sEncaps}$, and $\mathsf{sDecaps}$, where $\mathsf{KGen}_{\mathsf{enc}}$ and $\mathsf{sEncaps}$ are executed by the encapsulator, and $\mathsf{KGen}_{\mathsf{dec}}$ and $\mathsf{sDecaps}$ by the decapsulator.

- **split KEM key generation** for decapsulator and encapsulator, respectively: $(D, d) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{dec}}(1^\lambda)$ and $(E, e) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{enc}}(1^\lambda)$ are probabilistic algorithms that output a key pair, consisting of a public key (denoted by capital letters) and a private key (denoted by lowercase letters) in $\mathcal{PK}_{\mathsf{dec}} \times \mathcal{SK}_{\mathsf{dec}}$ and $\mathcal{PK}_{\mathsf{enc}} \times \mathcal{SK}_{\mathsf{enc}}$, respectively.
- **split KEM encapsulation**: $(c, K) \xleftarrow{\$} \mathsf{sEncaps}(e, D)$ is a probabilistic algorithm executed by the encapsulator enc. It takes as input $e \in \mathcal{SK}_{\mathsf{enc}}$, the private key of the encapsulator, and $D \in \mathcal{PK}_{\mathsf{dec}}$, the public key of the decapsulator. Algorithm $\mathsf{sEncaps}$ then outputs the shared secret $K \in \mathcal{K}$ along with its *encapsulation* $c \in \mathcal{C}$. It is common to simply refer to the encapsulation c of K as *ciphertext*.
- **split KEM decapsulation**: $K/\perp \leftarrow \mathsf{sDecaps}(d, E, c)$ is a deterministic algorithm executed by the decapsulator dec. From a ciphertext c, the decapsulator's secret key d, and encapsulator's public key E, it outputs either the decapsulation K of c or $\perp$, if the operation fails.

We say that a split KEM $\mathsf{sKEM} = (\mathsf{KGen}_{\mathsf{dec}}, \mathsf{KGen}_{\mathsf{enc}}, \mathsf{sEncaps}, \mathsf{sDecaps})$ is ϵ-*correct* if

$$\Pr[K' \neq K : (D, d) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{dec}}(1^\lambda), (E, e) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{enc}}(1^\lambda), (c, K) \xleftarrow{\$} \mathsf{sEncaps}(e, D), K' \leftarrow \mathsf{sDecaps}(d, E, c)] \leq \epsilon.$$

We call sKEM *(perfectly) correct* if $\epsilon = 0$.

[2] Note that it is not possible for Bob to precompute and store ciphertext(s) on the server alongside his public keys to avoid the additional message flow since Bob does not know in advance which user wishes to establish a secure chat with him.

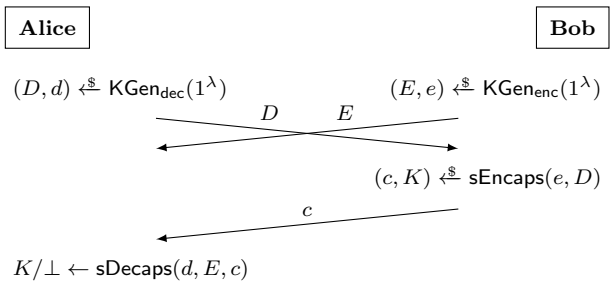

Figure 5: Communication flow for use of a split KEM.

Figure 5 shows the communication flow for use of a split KEM to establish a shared secret.

Symmetric Split KEMs. Supersingular-isogeny-based KEMs are an example where the specification of the key generation algorithm depends on the role of the generating party (cf., e.g., the NIST Round 2 candidate SIKE [11]). In supersingular-isogeny-based key exchanges, Alice and Bob generate public points in different subgroups of the curve. However, there are also many natural examples (e.g., DH- or LWE-based KEMs), where the key generation algorithms for the encapsulator and the decapsulator do not differ. This allows that generated key pairs can be used as input for both encapsulation and decapsulation. In order to capture these special types of split KEMs, we introduce the notion of a *symmetric* split KEM.

Definition 5 (Symmetric Split KEM)**.** We call a split KEM $\mathsf{sKEM} = (\mathsf{KGen}_{\mathsf{dec}}, \mathsf{KGen}_{\mathsf{enc}}, \mathsf{sEncaps}, \mathsf{sDecaps})$ *symmetric* if $\mathsf{KGen}_{\mathsf{dec}} = \mathsf{KGen}_{\mathsf{enc}}$ and the same key pair of a party is reused in both roles. In particular, this means that $\mathcal{PK}_{\mathsf{dec}} = \mathcal{PK}_{\mathsf{enc}}$ and $\mathcal{SK}_{\mathsf{dec}} = \mathcal{SK}_{\mathsf{enc}}$. For sake of simplicity, in this case we will often simply refer to the key generation algorithm as KGen instead of $\mathsf{KGen}_{\mathsf{dec}}$ and $\mathsf{KGen}_{\mathsf{enc}}$, respectively.
We say that a symmetric split KEM $\mathsf{sKEM} = (\mathsf{KGen}, \mathsf{sEncaps}, \mathsf{sDecaps})$ is ϵ-*correct* if both

$$\Pr\left[K' \neq K \mid (D, d) \xleftarrow{\$} \mathsf{KGen}(1^\lambda), (E, e) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)(c, K) \xleftarrow{\$} \mathsf{sEncaps}(e, D), K' \leftarrow \mathsf{sDecaps}(d, E, c)\right] \leq \epsilon$$

and

$$\Pr\left[K' \neq K \mid \xleftarrow{\$} \mathsf{KGen}(1^\lambda), (E, e) \xleftarrow{\$} (c, K) \xleftarrow{\$} \mathsf{sEncaps}(d, E), K' \leftarrow \mathsf{sDecaps}(e, D, c)\right] \leq \epsilon.$$

Again, as before, a symmetric split KEM is called *(perfectly) correct* if $\epsilon = 0$.

We note that it is not necessary to move to the symmetric split KEM setting if the key generation algorithms are the same for the encapsulator and decapsulator but a resulting key pair is only ever reused for a fixed role.

However, the symmetric split KEM setting is predestined for protocols like Signal's X3DH handshake with key encapsulation mechanisms. There, the long-term identity keys are used in both roles, either as the initiating party of the key exchange (the encapsulator) or the receiver (the decapsulator).

Plain (R)LWE-based key exchanges are split KEMs. In Figure 6 we illustrate that passively-secure key exchanges based on LWE naturally follow the split KEM flow. Encapsulation on Bob's side can be split into the generation of Bob's key pair as well as the final encapsulation of the shared key via the computation of an approximate shared secret and so-called *reconciliation information*, which constitutes the ciphertext.

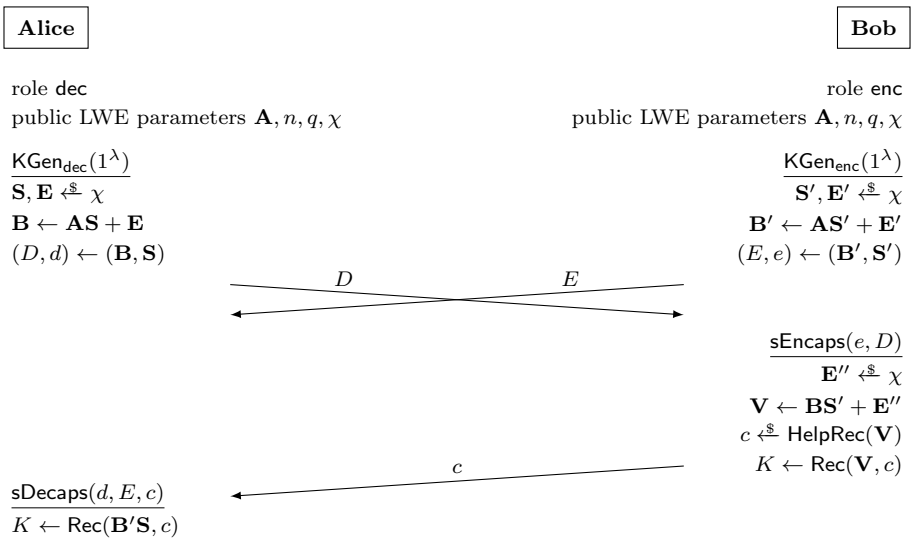

Figure 6: Example instantiation of split KEM flow with plain LWE as, e.g., in [5] with LWE parameters n, q, χ and fixed, public $\mathbf{A}$. The functions $\mathsf{HelpRec}$ and Rec aid computation of the shared secret from the approximately shared secret and differ among (R)LWE-based schemes.

$\underline{\mathcal{G}^{\mathsf{indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)}$:
1 $(D, d) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{dec}}(1^\lambda)$
2 $(E, e) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{enc}}(1^\lambda)$
3 $(c^\star, K_0^\star) \xleftarrow{\$} \mathsf{sEncaps}(e, D)$
4 $K_1^\star \xleftarrow{\$} \mathcal{K}$
5 $b \xleftarrow{\$} \{0, 1\}$
6 $b' \xleftarrow{\$} \mathcal{A}^{[\mathcal{O}_{\mathsf{sEncaps}}(\cdot)]_{\mathsf{atk} \in \{\mathsf{cpa},\mathsf{cca}\}}, [\mathcal{O}_{\mathsf{sDecaps}}(\cdot,\cdot)]_{\mathsf{atk}=\mathsf{cca}}}(D, E, c^\star, K_b^\star)$
7 **return** $[\![b' = b]\!]$

$\underline{\mathcal{O}_{\mathsf{sEncaps}}(D')}$:
8 $(c, K) \xleftarrow{\$} \mathsf{sEncaps}(e, D')$
9 **if** $(D', c) = (D, c^\star)$
10 **return** $\perp$
11 **return** (c, K)

$\underline{\mathcal{O}_{\mathsf{sDecaps}}(E', c)}$:
12 **if** $(E', c) = (E, c^\star)$
13 **return** $\perp$
14 **else**
15 **return** $\mathsf{sDecaps}(d, E', c)$

Figure 7: $\mathsf{IND\text{-}ATK}$ security for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$ of split KEMs. Brackets $[\mathcal{O}]_c$ indicate that the adversary has access to oracle $\mathcal{O}$ only if condition c is satisfied.

4.2 Security of Split KEMs

When translating the security definitions from the KEM setting (cf. Section 2.2) to the one of split KEMs, we need to address that encapsulation now contains a secret-key input e. This new scenario leads to a situation where the adversary cannot generate ciphertexts on its own when given only the public key D.

We begin with the usual indistinguishability-based security definitions $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CCA}$ which are formally depicted in Figure 7. In more detail:

- To formalize $\mathsf{IND\text{-}CPA}$ security we need to provide the adversary with an encapsulation oracle $\mathcal{O}_{\mathsf{sEncaps}}$ that internally has access to the secret key e of the encapsulator.
- The stronger notion of $\mathsf{IND\text{-}CCA}$ security is then as usual defined by also providing access to decapsulation oracle.

$\underline{\mathcal{G}^{\mathsf{owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)}$:

$(D, d) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{dec}}(1^\lambda)$

$(E, e) \xleftarrow{\$} \mathsf{KGen}_{\mathsf{enc}}(1^\lambda)$

$(c^\star, K^\star) \xleftarrow{\$} \mathsf{sEncaps}(e, D)$

$\widetilde{K} \xleftarrow{\$} \mathcal{A}^{[\mathcal{O}_{\mathsf{sEncaps}}(\cdot)]_{\mathsf{atk}\in\{\mathsf{cpa},\mathsf{pca},\mathsf{cca}\}},[\mathcal{O}_{\mathsf{Dist}}(\cdot,\cdot,\cdot)]_{\mathsf{atk}=\mathsf{pca}},[\mathcal{O}_{\mathsf{sDecaps}}(\cdot)]_{\mathsf{atk}=\mathsf{cca}}}(D, E, c^\star)$

return $[\![\widetilde{K} = K^\star]\!]$

$\underline{\mathcal{O}_{\mathsf{sEncaps}}(D')}$:

$(c, K) \xleftarrow{\$} \mathsf{sEncaps}(e, D')$

if $(D', c) = (D, c^\star)$

  return $\perp$

return (c, K)

$\underline{\mathcal{O}_{\mathsf{Dist}}(E', c, K)}$:

$K' \leftarrow \mathsf{sDecaps}(d, E', c)$

return $[\![K = K']\!]$

$\underline{\mathcal{O}_{\mathsf{sDecaps}}(E', c)}$:

if $(E', c) = (E, c^\star)$

  return $\perp$

else

  $K \leftarrow \mathsf{sDecaps}(d, E', c)$

  return K

Figure 8: OW-ATK security for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{PCA}, \mathsf{CCA}\}$ of split KEMs. Brackets $[\mathcal{O}]_c$ indicate that the adversary has access to oracle $\mathcal{O}$ only if condition c is satisfied.

Definition 6. Let $\mathsf{sKEM} = (\mathsf{KGen}_{\mathsf{dec}}, \mathsf{KGen}_{\mathsf{enc}}, \mathsf{sEncaps}, \mathsf{sDecaps})$ be a split KEM with key space $\mathcal{K}$. We say sKEM provides indistinguishability under *chosen-plaintext attacks* (CPA), or *chosen-ciphertext attacks* (CCA), in short sKEM is IND-ATK-secure for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$, if for every PPT adversary $\mathcal{A}$ the advantage function $\mathsf{Adv}^{\mathsf{indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)$ in winning the game $\mathcal{G}^{\mathsf{indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)$ as depicted in Figure 7 defined as

$$\mathsf{Adv}^{\mathsf{indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{G}^{\mathsf{indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} \right|$$

is negligible in the security parameter λ.

Similar to the standard KEM setting (cf. Section 2.2), we can also provide a security notion in terms of *one-wayness* for split KEMs, capturing an adversary's ability to actually *recover* the encapsulated key as opposed to merely distinguishing it from random.

In more detail, we provide a OW-ATK notion of one-wayness capturing different attack types ATK:

- For a *chosen-plaintext attack* (OW-CPA), an adversary may additionally query an encapsulation oracle $\mathcal{O}_{\mathsf{sEncaps}}$ to obtain further encapsulations.
- In a *plaintext-checking attack* [46] (OW-PCA), an adversary may in addition to the encapsulation oracle also query a distinguishing oracle $\mathcal{O}_{\mathsf{Dist}}$ yielding whether a given ciphertext decapsulates to a given key.
- Finally, a *chosen-ciphertext attack* (OW-CCA) considers an adversary that has access to both encapsulation and decapsulation oracles.

Definition 7. Let $\mathsf{sKEM} = (\mathsf{KGen}_{\mathsf{dec}}, \mathsf{KGen}_{\mathsf{enc}}, \mathsf{sEncaps}, \mathsf{sDecaps})$ be a split KEM with key space $\mathcal{K}$. We say sKEM provides one-wayness under *chosen-plaintext attacks* (CPA), *plaintext-checking attacks* (PCA), resp. *chosen-ciphertext attacks* (CCA), in short sKEM is OW-ATK-secure for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{PCA}, \mathsf{CCA}\}$, if for every PPT adversary $\mathcal{A}$ the advantage function $\mathsf{Adv}^{\mathsf{owatk}}_{\mathsf{sKEM},\mathcal{A}}$ in winning the game $\mathcal{G}^{\mathsf{owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)$ as depicted in Figure 8 defined as

$$\mathsf{Adv}^{\mathsf{owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) := \Pr\left[\mathcal{G}^{\mathsf{owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) = 1\right]$$

is negligible in the security parameter λ.

The definitions and description of the respective security notions for symmetric split KEMs can be found in Appendices A.1 and A.2.

4.3 X3DH with split KEMs

We now show that using the split KEM formalism solves the aforementioned problems when switching from the DH to the KEM setting. Figure 9 illustrates the flow between Alice and Bob using only split KEMs. On the one hand, the formalization of split KEMs now allow both parties to reuse key pairs and have them contribute equally to the key agreement within the encapsulation procedure(s). Furthermore, regarding the issue of having to encapsulate without knowing the corresponding public key, the split KEM formalism gets rid of the additional message flow from Bob to Alice, thereby effectively regaining the asynchronicity of the secure messaging application.

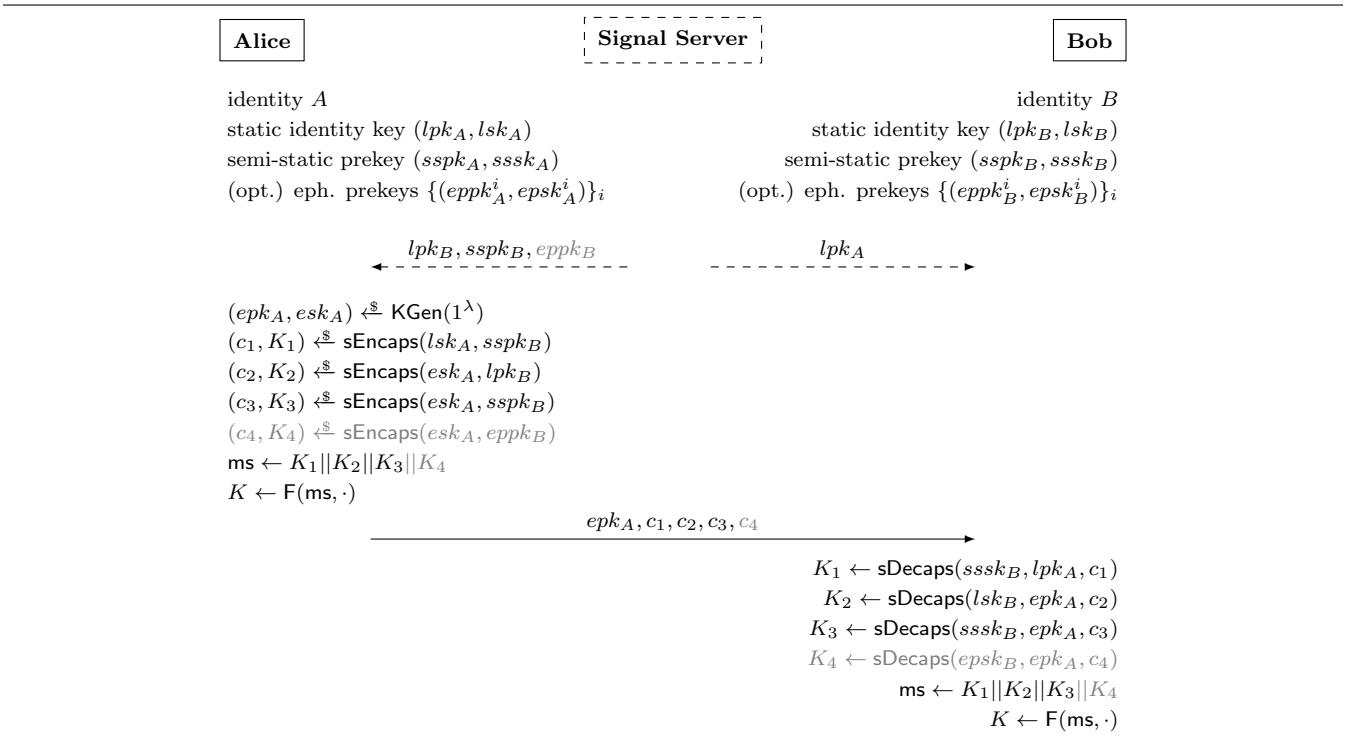

Figure 9: Split KEM flow for the KEM-based version of Signal's X3DH handshake. Interaction with the Signal server is dashed, the optional ephemeral prekey (combination) is depicted in gray.

5 Challenges

We have seen that split KEMs are an effective means to capture DH-style key exchange flows and contributiveness in KEMs. The starting point for this discussion and the need for a split-KEM-like notion stemmed from the fact that key exchange protocols based on DH must eventually be transitioned to post-quantum secure alternatives. These are given in the form of key encapsulation mechanisms and thus do not readily model the DH-flow. For "simple" protocols, that only combine two ephemeral DH key pairs at a time, this should not pose too much of an issue. For specialized usages, such as 0-RTT modes based on DH or

intricate patterns with many different DH combinations as in the initial key agreement of Signal, involving static keys, we have seen that standard KEMs are often inadequate.

We thus introduced the notion of split key encapsulation mechanisms. However, a major challenge remains, when it comes to showing that known KEMs fulfill the split KEM notion: While many passively-secure lattice-based KEMs are a prime example of the structure of split KEMs since their encapsulation can be divided up into key generation and key agreement on the encapsulator's side, we know that these are not secure when keys are reused [26, 14, 15, 19, 42, 17, 50, 3, 18] and thus we **were not able to instantiate strongly-secure split KEMs** from those.

Strongly-secure lattice-based KEMs can be achieved by taking the underlying PKE scheme and transforming it via a generic transform, usually the Fujisaki-Okamoto (FO) transform [27, 28, 31]. However, FO-transformed KEMs are no longer compatible with the split KEM flow. The FO transform provides active security by essentially having the encapsulator send to the decapsulator the randomness it used during encapsulation, which the decapsulator uses this to reconstruct the ciphertext to ensure it was well-formed. This means that the encapsulator's secret randomness cannot be reused (since it is disclosed to the decapsulator), and thus an FO-transformed KEM only supports static-ephemeral combinations that involve the decapsulator's static key. Furthermore, each FO ciphertext encapsulated by Alice has a different ephemeral contribution which also contradicts the idea of Signal's X3DH to pair the *same* ephemeral key of Alice with various keys of Bob.

Thus, it remains an open question to develop post-quantum constructions support static-static key exchange, or that can accommodate reversed message flows; in other words, it is an open question to develop post-quantum constructions that have the same flexibility as Diffie–Hellman-based primitives.

Acknowledgements

Jacqueline Brendel and Marc Fischlin have been funded by the DFG as part of project S4 within the CRC 1119 CROSSING and as part of project D.2 within the RTG 2050 "Privacy and Trust for Mobile Users." Felix Günther has been supported in part by Research Fellowship grant GU 1859/1-1 of the German Research Foundation (DFG) and National Science Foundation (NSF) grants CNS-1526801 and CNS-1717640. Christian Janson has been co-funded by the DFG as part of project P2 within the CRC 1119 CROSSING. Douglas Stebila has been supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada Discovery grant RGPIN-2016-05146 and a NSERC Discovery Accelerator Supplement.

References

[1] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (Apr 2001)

[2] Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 129–158. Springer, Heidelberg (May 2019)

[3] Bauer, A., Gilbert, H., Renault, G., Rossi, M.: Assessment of the key-reuse resilience of NewHope. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 272–292. Springer, Heidelberg (Mar 2019)

[4] Bergsma, F., Dowling, B., Kohlar, F., Schwenk, J., Stebila, D.: Multi-ciphersuite security of the secure shell (SSH) protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 2014. pp. 369–381. ACM Press (Nov 2014)

[5] Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1006–1018. ACM Press (Oct 2016)

[6] Brendel, J., Fischlin, M., Günther, F., Janson, C.: PRF-ODH: Relations, instantiations, and impossibility results. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 651–681. Springer, Heidelberg (Aug 2017)

[7] Cohn-Gordon, K., Cremers, C.J.F., Dowling, B., Garratt, L., Stebila, D.: A Formal Security Analysis of the Signal Messaging Protocol. In: IEEE European Symposium on Security and Privacy, EuroS&P 2017. pp. 451–466 (2017)

[8] Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. 33(1), 167–226 (Jan 2004)

[9] Cremers, C.J.F., Feltz, M.: Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 734–751. Springer, Heidelberg (Sep 2012)

[10] Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. In: NIST 2nd Post-Quantum Cryptography Standardization Conference 2019 (August 2019)

[11] David Jao, R.A., Campagna, M., Costello, C., Feo, L.D., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation (April 2019), https://sike.org/

[12] Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (Aug 2008), https://www.rfc-editor.org/rfc/rfc5246.txt

[13] Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)

[14] Ding, J., Alsayigh, S., RV, S., Fluhrer, S., Lin, X.: Leakage of signal function with reused keys in RLWE key exchange. Cryptology ePrint Archive, Report 2016/1176 (2016), http://eprint.iacr.org/2016/1176

[15] Ding, J., Alsayigh, S., Saraswathy, R.V., Fluhrer, S.R., Lin, X.: Leakage of signal function with reused keys in RLWE key exchange. In: IEEE International Conference on Communications, ICC 2017. pp. 1–6 (2017)

[16] Ding, J., Branco, P., Schmitt, K.: Key exchange and authenticated key exchange with reusable keys based on RLWE assumption. Cryptology ePrint Archive, Report 2019/665 (2019), https://eprint.iacr.org/2019/665

[17] Ding, J., Cheng, C., Qin, Y.: A simple key reuse attack on LWE and ring LWE encryption schemes as key encapsulation mechanisms (KEMs). Cryptology ePrint Archive, Report 2019/271 (2019), https://eprint.iacr.org/2019/271

[18] Ding, J., Deaton, J., Zhang, Z., Schmidt, K., Vishakha: A simple key reuse attack on NTRU cryptosystem. Cryptology ePrint Archive, Report 2019/1022 (2019), https://eprint.iacr.org/2019/1022

[19] Ding, J., Fluhrer, S.R., RV, S.: Complete attack on RLWE key exchange with reused keys, without signal leakage. In: ACISP 2018. LNCS, vol. 10946, pp. 467–486. Springer, Heidelberg (2018)

[20] Dobson, S., Galbraith, S.D., LeGrow, J., Ti, Y.B., Zobernig, L.: An adaptive attack on 2-SIDH. Cryptology ePrint Archive, Report 2019/890 (2019), https://eprint.iacr.org/2019/890

[21] Dobson, S., Li, T., Zobernig, L.: A note on a static SIDH protocol. Cryptology ePrint Archive, Report 2019/1244 (2019), https://eprint.iacr.org/2019/1244

[22] Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1197–1210. ACM Press (Oct 2015)

[23] Drucker, N., Gueron, S.: Continuous key agreement with reduced bandwidth. Cryptology ePrint Archive, Report 2019/088 (2019), https://eprint.iacr.org/2019/088

[24] Duits, I.: The Post-Quantum Signal Protocol: Secure Chat in a Quantum World. Master's thesis, University of Twente (February 2019), http://essay.utwente.nl/77239/

[25] Fischlin, M., Günther, F.: Multi-stage key exchange and the case of Google's QUIC protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 2014. pp. 1193–1204. ACM Press (Nov 2014)

[26] Fluhrer, S.: Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016), http://eprint.iacr.org/2016/085

[27] Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999)

[28] Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology 26(1), 80–101 (Jan 2013)

[29] Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 63–91. Springer, Heidelberg (Dec 2016)

[30] Gao, X., Ding, J., Li, L., Liu, J.: Practical randomized RLWE-based key exchange against signal leakage attack. IEEE Transactions on Computers 67(11), 1584–1593 (Nov 2018)

[31] Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Heidelberg (Nov 2017)

[32] Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012)

[33] Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. pp. 19–34. Springer, Heidelberg (Nov / Dec 2011)

[34] Kayacan, S.: A note on the static-static key agreement protocol from supersingular isogenies. Cryptology ePrint Archive, Report 2019/815 (2019), https://eprint.iacr.org/2019/815

[35] Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (Aug 2005)

[36] Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer, Heidelberg (Aug 2013)

[37] Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. In: 2016 IEEE European Symposium on Security and Privacy, EuroS&P 2016. pp. 81–96. IEEE, Saarbrücken, Germany (Mar 21–24, 2016)

[38] Kwiatkowski, K., Valenta, L.: The TLS Post-Quantum Experiment. The Cloudflare Blog, https://blog.cloudflare.com/the-tls-post-quantum-experiment/ (October 2019)

[39] LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (Nov 2007)

[40] Langley, A.: CECPQ1 results. Imperial Violet, Blog, https://www.imperialviolet.org/2016/11/28/cecpq1.html (Nov 2016)

[41] Langley, A., Chang, W.T.: QUIC Crypto. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/ (Dec 2016), revision 20161206

[42] Liu, C., Zheng, Z., Zou, G.: Key reuse attack on NewHope key exchange protocol. In: Lee, K. (ed.) ICISC 18. LNCS, vol. 11396, pp. 163–176. Springer, Heidelberg (Nov 2019)

[43] Marlinspike, M., Perrin, T.: The X3DH key agreement protocol (November 2016), https://signal.org/docs/specifications/x3dh/

[44] National Institute of Standards and Technology (NIST): Post-quantum cryptography. https://csrc.nist.gov/projects/post-quantum-cryptography (Aug 19, 2015)

[45] Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (Feb 2001)

[46] Okamoto, T., Pointcheval, D.: REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 159–175. Springer, Heidelberg (Apr 2001)

[47] Perrin, T.: The Noise protocol framework (July 2018, Revision 34), http://www.noiseprotocol.org/noise.pdf

[48] Perrin, T., Marlinspike, M.: The double ratchet algorithm (November 2016), https://signal.org/docs/specifications/doubleratchet/

[49] Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 3–32. Springer, Heidelberg (Aug 2018)

[50] Qin, Y., Cheng, C., Ding, J.: A complete and optimized key mismatch attack on NIST candidate NewHope. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019, Part II. LNCS, vol. 11736, pp. 504–520. Springer, Heidelberg (Sep 2019)

[51] QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic

[52] Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard) (Aug 2018), https://www.rfc-editor.org/rfc/rfc8446.txt

[53] Shoup, V.: A Proposal for an ISO Standard for Public Key Encryption (version 2.1) (Dec 2001), https://www.shoup.net/papers/iso-2_1.pdf

[54] Signal protocol: Technical documentation. https://whispersystems.org/docs/

[55] Stebila, D., Mosca, M.: Post-quantum key exchange for the internet and the open quantum safe project. In: Avanzi, R., Heys, H.M. (eds.) SAC 2016. LNCS, vol. 10532, pp. 14–37. Springer, Heidelberg (Aug 2016)

[56] Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Designs, Codes and Cryptography 46(3), 329–342 (Mar 2008)

[57] Xue, H., Lu, X., Li, B., Liang, B., He, J.: Understanding and constructing AKE via double-key key encapsulation mechanism. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 158–189. Springer, Heidelberg (Dec 2018)

[58] Yao, A.C.C., Zhao, Y.: OAKE: a new family of implicitly authenticated Diffie-Hellman protocols. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 1113–1128. ACM Press (Nov 2013)

A Security Notions for Symmetric Split KEMs

In the following, we simply provide the analogous security formalizations as given in Section 4.2 and adapt them accordingly to the symmetric case.

A.1 IND-ATK Security

Definition 8. Let $\mathsf{sKEM} = (\mathsf{KGen}, \mathsf{sEncaps}, \mathsf{sDecaps})$ be a symmetric split KEM with key space $\mathcal{K}$. We say sKEM provides symmetric indistinguishability under *chosen-plaintext attacks (*CPA*)*, resp. *chosen-ciphertext attacks (*CCA*)*, in short sKEM is $\mathsf{sym\text{-}IND\text{-}ATK}$-secure for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$, if for every PPT adversary $\mathcal{A}$ the advantage function $\mathsf{Adv}^{\mathsf{sym\text{-}indatk}}_{\mathsf{sKEM},\mathcal{A}}$ in winning the game $\mathcal{G}^{\mathsf{sym\text{-}indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)$ as depicted in Figure 10 is negligible in the security parameter λ, where

$$\mathsf{Adv}^{\mathsf{sym\text{-}indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) := \Big| \Pr\Big[\mathcal{G}^{\mathsf{sym\text{-}indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) = 1\Big] - \frac{1}{2}\Big|.$$

A.2 OW-ATK Security

Definition 9. Let $\mathsf{sKEM} = (\mathsf{KGen}, \mathsf{sEncaps}, \mathsf{sDecaps})$ be a symmetric split KEM. We say sKEM provides symmetric one-wayness under *chosen-plaintext attacks (*CPA*)*, *plaintext-checking attacks (*PCA*)*, resp. *chosen-ciphertext attacks (*CCA*)*, in short we say sKEM is $\mathsf{sym\text{-}OW\text{-}ATK}$-secure for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{PCA},$

$\underline{\mathcal{G}^{\mathsf{sym\text{-}indatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)}$:

$(D, d) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
$(E, e) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
$(c^\star, K_0^\star) \xleftarrow{\$} \mathsf{sEncaps}(e, D)$
$K_1^\star \xleftarrow{\$} \mathcal{K}$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\left([\mathcal{O}^{sk}_{\mathsf{sEncaps}}(\cdot)]_{\mathsf{atk}\in\{\mathsf{cpa},\mathsf{cca}\}}, [\mathcal{O}^{sk}_{\mathsf{sDecaps}}(\cdot,\cdot)]_{\mathsf{atk}=\mathsf{cca}}\right)_{sk\in\{d,e\}}}(D, E, c^\star, K_b^\star)$
return $[\![b' = b]\!]$

$\underline{\mathcal{O}^{sk}_{\mathsf{sEncaps}}(pk)}$:

$(c, K) \xleftarrow{\$} \mathsf{sEncaps}(sk, pk)$
if $(pk, c) = (D, c^\star)$
  return $\perp$
return (c, K)

$\underline{\mathcal{O}^{sk}_{\mathsf{sDecaps}}(pk, c)}$:

if $sk = d$ and $(pk, c) = (E, c^\star)$
or **if** $sk = e$ and $(pk, c) = (D, c^\star)$
  return $\perp$
else
  return $\mathsf{sDecaps}(sk, pk, c)$

Figure 10: sym-IND-ATK security for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$ of symmetric split KEMs. Brackets $[\mathcal{O}]_c$ indicate that the adversary has access to oracle $\mathcal{O}$ only if condition c is satisfied.

$\mathsf{CCA}\}$, if for every PPT adversary $\mathcal{A}$ the advantage function $\mathsf{Adv}^{\mathsf{sym\text{-}owatk}}_{\mathsf{sKEM},\mathcal{A}}$ in winning the game $\mathcal{G}^{\mathsf{sym\text{-}owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)$ as depicted in Figure 11 is negligible in the security parameter λ, where

$$\mathsf{Adv}^{\mathsf{sym\text{-}owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) := \Pr\left[\mathcal{G}^{\mathsf{sym\text{-}owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda) = 1\right].$$

$\underline{\mathcal{G}^{\mathsf{sym\text{-}owatk}}_{\mathsf{sKEM},\mathcal{A}}(\lambda)}$:

$(D, d) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$

$(E, e) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$

$(c^\star, K^\star) \xleftarrow{\$} \mathsf{sEncaps}(e, D)$

$\widetilde{K} \xleftarrow{\$} \mathcal{A}^{\left([\mathcal{O}^{sk}_{\mathsf{sEncaps}}(\cdot)]_{\mathsf{atk}\in\{\mathsf{cpa},\mathsf{pca},\mathsf{cca}\}},[\mathcal{O}^{sk}_{\mathsf{Dist}}(\cdot,\cdot,\cdot)]_{\mathsf{atk}=\mathsf{pca}},[\mathcal{O}_{\mathsf{sDecaps}}(\cdot)]_{\mathsf{atk}=\mathsf{cca}}\right)_{sk\in\{d,e\}}}(D, E, c^\star)$

return $[\![\widetilde{K} = K^\star]\!]$

$\underline{\mathcal{O}^{sk}_{\mathsf{sEncaps}}(pk)}$:

$(c, K) \xleftarrow{\$} \mathsf{sEncaps}(sk, pk)$

if $(pk, c) = (D, c^\star)$

  return $\perp$

return (c, K)

$\underline{\mathcal{O}^{sk}_{\mathsf{Dist}}(pk, c, K)}$:

$K' \leftarrow \mathsf{sDecaps}(sk, pk, c)$

return $[\![K = K']\!]$

$\underline{\mathcal{O}^{sk}_{\mathsf{sDecaps}}(pk, c)}$:

if $sk = d$ and $(pk, c) = (E, c^\star)$

or **if** $sk = e$ and $(pk, c) = (D, c^\star)$

  return $\perp$

else

  $K \leftarrow \mathsf{sDecaps}(sk, pk, c)$

  return K

Figure 11: sym-OW-ATK security for $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{PCA}, \mathsf{CCA}\}$ of split KEMs. Brackets $[\mathcal{O}]_c$ indicate that the adversary has access to oracle $\mathcal{O}$ only if condition c is satisfied.