On using the same key pair for Ed25519 and an X25519 based KEM

Erik Thormarker*
Ericsson

Abstract

Haber and Pinkas [12] discussed the principle of when it is secure to reuse key material between public-key cryptosystems. They showed that this can be secure for multiple combinations of systems, including Schnorr signatures. Degabriele, Lehmann, Paterson, Smart and Strefler [10] proved the security of sharing a key pair between a generic elliptic curve Schnorr signature scheme and an elliptic curve Diffie-Hellman based KEM in the random oracle model (ROM). They essentially ran the original security proofs in parallel by leveraging domain separation for the random oracle (RO) usage between the signature scheme and the specific KDF of the KEM. We make two contributions. First, we extend the result in [10] by proving the joint security in the ROM of an X25519 based KEM with an HKDF-Extract-like KDF construction and Ed25519. Second, we make no assumptions about domain separation of RO usage between the two systems while making minimal assumptions about the format of the RO usage in Ed25519. Our result is applicable to Ed448 and a corresponding KEM based on X448 as well.

1 Introduction

Haber and Pinkas [12] discuss the principle of when it is secure to share a key pair between two public-key cryptosystems. They point out that not sharing key material between two systems should always be the default choice, and sharing key material should only be considered if there is a proof that doing so is secure. They go on to discuss that for example differences in life cycle management may make sharing key material between different types of systems impossible or unsuitable. However, they also discuss how there may be suitable real-world use cases where sharing key material makes sense. Part of our motivation here is ongoing work in an IETF draft ietf-core-oscore-groupcomm [21] where an endpoint, in one mode of operation, needs to sign its messages delivered to a group of endpoints using e.g. Ed25519 and, in another pairwise mode of operation, authenticate messages exchanged between itself and another endpoint efficiently using e.g. keys derived from a X25519 shared secret. All endpoints already have access to each others Ed25519 public keys and it is suitable to reuse those for the pairwise authentication. This results in savings in key distribution and storage, which may be especially beneficial in constrained applications. Furthermore, in some constrained IoT scenarios where using two keys is not possible to begin with, reusing a single key enables replacing signatures with MAC tags (e.g. as in the pairwise mode of operation described above), which reduces overhead, or prevents applications from resorting to relying on symmetric keys and weaker security models.

Degabriele, Lehmann, Paterson, Smart and Strefler [10] proved the joint security – that is, that the key pair can be shared – of a generic elliptic curve Schnorr signature scheme and an elliptic curve Diffie-Hellman based KEM in the random oracle model (ROM). They essentially ran the

*erik.thormarker@ericsson.com

original proofs in parallel by leveraging domain separation for the RO usage between the signature scheme and the specific KDF of the KEM. There are a few reasons for us to extend this work here. While it may be clear to some that this result carries over to the setting of X25519 and Ed25519, sharing key material is – for good reasons – such a delicate matter that an explicit reference for this important case is warranted. Another reason for this work is that applications may rather want to use the common HKDF-Extract (that is, a salted HMAC) as KDF than the KDF specified in [10]. For applications that may support a range of hash functions, and maybe not want to lock in on a specific one of the Ed25519 variants in for example RFC 8032 [13] it can be tedious to assert that the multiple underlying hash function invocations in, say, HKDF are separable from those of any relevant Ed25519 variant – if the systems happen to use the same hash function in a configuration. Our result here also complement that in [10] in the generic group model for joint security of an ECDH based KEM and ECDSA, which does not require domain separation for the hash function usage. Finally, our result is also of independent theoretical interest because it shows that joint security in the ROM for these two systems is not necessarily dependent on domain separation for the random oracle (RO) usage.

At a glance, a fairly recent work by Patton and Shrimpton [18] appears to not be relevant to our aim here. They discuss and prove results for a framework for general key reuse through context-separable interfaces while our work here aims to provide key reuse between two concrete cryptosystems without leveraging any context separation. However, their [18, Theorem 6] says that any of the EdDSA variants of RFC 8032 [13] – including those that do not use context – can be composed with an interface $\mathcal{I}$ for a (in their terminology) *simple* group $\mathbb{G}$, without hurting security of $\mathcal{I}$'s intended application. This is similar to our Theorem 1 (though our result is restricted to a specific KEM). However, their simple groups are by definition cyclic and implicitly enforce point validation. Our KEM operates on X25519 public keys rather than a (single) group.

Following the definitions in [10], to show joint security we need to show the security of the KEM in the presence of a signing oracle and the security of the signature scheme in the presence of a decapsulation oracle.

At the core of showing our KEM secure in the presence of a signing oracle – without relying on domain separation – is the observation that the unpredictability the commitment in a signature, which in the case of Schnorr signatures is added when hashing the message, means that we will not have any collisions with inputs that have been hashed as up until that point in the KDF of the KEM. This observation is the same as that used in proofs of security for Schnorr signatures to argue that the RO can be redefined at suitable inputs while keeping the RO consistent in the view of the adversary, see e.g. [19, Lemma 4]. If formatting allows and the adversary wishes so, it can potentially later send crafted decapsulation queries so that an RO input previously used in signing also appears somewhere in the RO invocations of the KEM's KDF. In our security proofs, we just need to make sure that our (simulation of the) decapsulation oracle is consistent with regard to the information about the RO that has leaked through the signing oracle. See Theorem 1 for details.

To show that Ed25519 is secure in the presence of a decapsulation oracle for our KEM we tap into recent work by Brendel, Cremers, Jackson and Zhao [7] which, among other things, adapts a generic Schnorr signature security proof to the specific setting of Ed25519. In [7, Theorem 3] a reduction algorithm $\mathcal{B}$ leverages an adversary $\mathcal{A}$ against the EUF-CMA security of Ed25519 to impersonate the prover in a related identification protocol. To do this, $\mathcal{B}$ needs to guess the index of an RO query which is relevant to a forged signature output by $\mathcal{A}$ in the EUF-CMA game. In our setting we want to make minimal assumptions about the format of the hash input used in signature verification, so we also need to account for the fact that $\mathcal{A}$ could learn the relevant information through a decapsulation query. In addition to this, we have the consistency issues from above that we need to deal with. See Theorem 2 for details.

Theorems 1 and 2 show the joint security of our KEM and Ed25519. Theorems 3 and 4 do the corrsponding for a corresponding KEM for X448 and Ed448.

1.1 Notation

We write $s \leftarrow_\$ S$ if an element s is chosen uniformly at random in a set S. For a set S, $|S|$ is its size. We will work with bytestrings rather than e.g. bitstrings and all functions we mention work on bytestrings. For a bytestring s, $|s|$ is its length in bytes. In the specific case of of the random oracle H below, we write $\mathrm{H}(s_1, ..., s_n)$ instead of $\mathrm{H}(s_1||...||s_n)$ for readability, where $s_1||...||s_n$ is the concatenation of the bytestrings s_1, ..., s_n. For bytestrings s_1, ..., s_n, the n-tuple $(s_1, ..., s_n)$ is the concatenation $s_1||...||s_n$. We sometimes implicitly interpret bytestrings as integers, in those cases the bytestrings are interpreted as being little-endian encoded integers as in [13, 5.1.2]. We write $s \leftarrow_\$ \{0, 1, ..., 255\}^n$ to indicate that a bytestring of length n is chosen uniformly at random. We let *blocklen* be the block length in bytes of the hash function H that instantiates our RO in practice. We let *hashlen* be the length in bytes of the output of H. For a byte b, b^n is the bytestring consisting of b repeated n times. We will sometimes refer to an elliptic curve basepoint G, depending on context this will either be the base point of Ed25519 in [13] or the encoding of the base point of X25519 in [17]. In the context of Ed25519 we let ℓ be the order of G. We often let the encoding of points on edwards25519 (the curve of Ed25519) be handled implicitly for readability.

2 Specification

Suppose that we want to use a single key pair for both Ed25519 and a KEM based on X25519. If different hash functions are used in the signature scheme and the KEM, then the original standalone proofs for the respective systems can be run in parallel to prove joint security. The proof in [10, Theorem 2] already covers this case since that proof assumes domain separation for the hash function usage between the two systems. It is also clear that our proofs for the specific variants we deal with here can be adapted to this simpler case. For the rest of this paper we consider the less trivial case where same hash function H is used for both systems. We model H as an RO in our security proofs.

Ed25519 was introduced by Bernstein, Duif, Lange, Schwabe and Yang in [6], and standardized in RFC 8032 [13]. We follow the scheme as defined in [13] and refer there for the full definition. We highlight the following

- Our result does not cover pre-hashing variants of Ed25519 such as Ed25519ph from RFC 8032 [13] which "SHOULD NOT be used" according to [13, 8.5]. We are not aware of any fundamental problems with extending our result to include Ed25519ph or other pre-hashing variants.
- We avoid depending on the specifics of the hash input for signature verification of a message m. To have something concrete we will write $\mathrm{H}(R, X, m)$ (where R is the commitment and X is the public key), but our theorems only depend on R being a substring of the hash input.
- Our theorems are valid for (but not necessarily limited to) any of the Ed25519 variants listed in [7, Table 2], including the Ed25519 variants Ed25519 and Ed25519ctx in RFC 8032 [13].
- In Ed25519, the commitment $R = rG$ for a message m is deterministically generated through $r = \mathrm{H}(k', m)$, where k' are the rightmost *hashlen*/2 bytes of $\mathrm{H}(k)$ and k is the root secret key (that is the "private key" in [13, 5.1.2]).

- We write an (honestly generated) Ed25519 signature for a message m as (R, s) with $R = rG$ and $s = r + h \cdot \mathrm{clamp}(x) \bmod \ell$, where R and r are as above, $h = \mathrm{H}(R, X, m)$ is as above, clamp is as defined in Section 3, and x is the private signing key bytestring (which is also deterministically generated from the root secret key).

X25519 was introduced by Bernstein in [5] and standardized in RFC 7748 [17]. Let a and x be randomly chosen bytestrings as in [17], and let G be the endoding of the basepoint for X25519 in [17]. We define the following KEM.

- $\mathrm{KeyGen}() := (x, X) = (x, \mathrm{X25519}(x, G))$
- $\mathrm{Encap}() := (A, Z) = (\mathrm{X25519}(a, G), \mathrm{KDF_H}(A, \mathrm{X25519}(a, X))$
- $\mathrm{Decap}(A) := \mathrm{KDF_H}(A, \mathrm{X25519}(x, A))$

Note that the encapsulation A can also be produced differently. For example, in ietf-core-oscore-groupcomm [21], the encapsulation is the sender's (static) Ed25519 public key. In this case the recipient translates the public key from edwards25519 to an X25519 public key using the map from [17]. To avoid potential malleability and preserve IND-CCA security, the original untranslated encapsulation A should be put into the KDF during decapsulation (and encapsulation).

In the following, *salt* is a bytestring that is at most *blocklen* bytes long (it serves the same purpose as the salt in HKDF [15], see discussion below). The key derivation function is then defined as (note Item (iii) with a condition on salt)

- $\mathrm{KDF_H}(A, Z) := \mathrm{HMAC_H}(salt, A||Z) = \mathrm{H}(\mathrm{z}(salt) \oplus out, \mathrm{H}(\mathrm{z}(salt) \oplus in, A, Z))$, where
  - (i) $\mathrm{z}(salt) = salt||0\mathrm{x}00^{(blocklen-|salt|)}$
  - (ii) *in* and *out* are HMAC constants from [14]
  - (iii) *salt* is such that
    $$\mathrm{z}(salt) \oplus out \neq \mathrm{z}(salt') \oplus in$$
    for any other $salt'$ used with the same key pair[1]. Two possible ways that this can be achieved are by letting *salt* be
    - fixed to a bytestring of *blocklen* many 0x00 bytes, or
    - $|salt| \leq |blocklen| - 1$

    Note that the second item above says that salt is 1 byte (not bit) shorter than the block length of the hash function[2].

Our KDF is similar to a one-step KDF based on HMAC in NIST SP 800-56C [8], when we take only hashlen output. Our security proofs depend on

- being able to parse A and Z from the input of the inner hash invocation, and
- that the input to the inner hash invocation of HMAC is distinct for any distinct $A \neq A'$.

[1]This simplifies our security proofs by providing domain separation between the inner and outer hash invocations between invocations with distinct *salt*.

[2]Also note that salts longer than blocklen which are first hashed (as in the HMAC specification [14]) are fine, assuming that $hashlen < blocklen - 1$.

Other than that, applications are free to use other formatting or insert other information into the message field of the HMAC application[3]. Our KDF also agrees with HKDF-Extract [15] when A is appended to the typical Input Key Material (IKM) Z. The output from our KDF can be plugged into a PRF like HKDF-Expand. We note that the computational min-entropy of Z [16, Definition 4] is already conditioned on the public information A. Krawczyk discusses various randomness extraction results for NMAC and HMAC and keying material of required min-entropy in [16][4], referencing work by Dodis, Gennaro, Håstad, Krawczyk and Rabin [11].

How salts are derived or authenticated is not in scope in this paper. To achieve generality in our security proofs we work in a worst-case scenario where the attacker chooses the salt at all times (conditioned on the constraint we set on salts in the KDF definition). See for example the discussion by Krawczyk and Eronen in [15] for a starting point about choosing salts. Salts longer than *blocklen* are hashed first in HMAC. We do not treat this explicitly since the attacker is free to run a salt through the RO before supplying it to us in the security proofs.

It appears to be necessary for us to give a security proof for the KEM from scratch since we want to argue that there is no harmful interaction between the hash invocations in HMAC and the hash invocation in the signature scheme. It is otherwise natural to consider the whole KDF as a single RO, e.g. as in [9, Theorem 9]. For the security proof of our KEM we follow the proof in [9, Theorem 9] by Cramer and Shoup for their KEM [9, Figure 8] with some tweaks:

- there is some extra bookkeeping due to the key derivation being implemented through iterated invocations of H, and
- differently from the KEM in [9, Figure 8], we do not perform any public key validation during decapsulation.

The second difference above is natural because underlying X25519 implementations may skip validating public keys since the curve and public key format was chosen Bernstein in [5] to remain secure without such validation [4]. As a consequence of this difference we will rely on the Strong Diffie-Hellman (SDH) assumption described by Abdalla, Bellare and Rogaway in [1, Definition 9], instead of the Gap-DH assumption as used in [9, Theorem 9]. In both the SDH problem and the Gap-DH problem the task is to solve the Computational Diffie-Hellman (CDH) problem with the help of an oracle. That is, to compute xaP when given xP and aP for a known basepoint P. Our version of the CDH problem will be: Given $\mathrm{X25519}(x, G)$ and $\mathrm{X25519}(a, G)$, where x and a are randomly chosen bytestrings as in RFC 7748 [17], compute $\mathrm{X25519}(x, \mathrm{X25519}(a, G))$. In the Gap-DH problem, while solving CDH, we are given access to a Decisional DH (DDH) oracle that recognizes DH triples (bP, cP, bcP) for public keys bP, cP, bcP. In the SDH problem applied to our setting we instead have an oracle O_x that takes X25519 public keys as input and is defined as

$$\mathrm{O}_x(A, Z) := (\mathrm{X25519}(x, A) == Z)$$

where x is one of the private keys from the CDH problem. We assume that we can apply the oracle to any X25519 public keys, and thus we can check scalar multiplication by x outside the main prime subgroup used by X25519, and in particular on the twist[5]. A traditional DDH oracle does not appear to be sufficient to simulate decapsulations in our security proof since there is no apparent way to use it to ensure consistency between decapsulation queries on the twist and RO

[3]For example, if symmetry is desirable, then $A \oplus X||Z$ instead of $A||Z$ would agree with our security proofs.

[4]Note that applications that want to rely on the particular result of [16, Lemma 6] may want to e.g. make sure that block length of the inner hash invocation after padding (including the salt block) is two blocks.

[5]Note that this differs from the oracle in [1, Definition 9], which is defined for, by their terminology, *represented* groups [1, 2.1].

queries related to those decapsulation queries. The SDH assumption is that CDH is sufficiently hard, when given access to O_x. This assumption matches typical static key X25519 usage in the real-world since O_x can be simulated by an active attacker that interacts through DH with the holder of a target static X25519 key pair. Checking whether $\mathrm{X25519}(x, A) = Z$, for an attacker crafted public key A and some guess for the shared secret Z, can typically be done by for example trying to decrypt some ciphertext that was encrypted by the holder of the target key using a key derived from the shared secret. Finally, note that our oracle O_x appears to be similar to a decision oracle DH in a recent work by Alwen, Blanchet, Hauck, Kiltz, Lipp and Riepel [2, Definition 5] in that $\mathrm{O}_x(A, Z) = \mathrm{DH}(X, A, Z)$ when their *nominal group* is curve25519. They do use this oracle in the security proof for a KEM, but the KDF is modeled as a single RO.

The following lemma will be used implicitly throughout theorems 2 and 3.

Lemma 1. *Let q be any fixed bytestring. Let p_1 be the probability that an honestly generated X25519 public key equals q. Independently, let p_2 be the probability that q equals rG, where*

- *G, of order ℓ, is the basepoint from Ed25519,*
- *$r \leftarrow_\$ \{0, 1, ..., \ell - 1\}$, and*
- *(the scalar multiplication of G by r) rG is encoded using the point to bytestring encoding of [13].*

Then we have $p_1, p_2 \leq 2^{-250}$.

Proof. See [2, Lemma 1] for p_1. For p_2, the encoded rG determines the scalar r and $\ell > 2^{250}$. □

For the rest of the paper we set the variable *minentropy* := 250.

3 Public key translation and the SDH problem

As stated in Section 2 our SDH problem is: Given $\mathrm{X25519}(x, G)$, $\mathrm{X25519}(a, G)$ and access to O_x, where $x, a \leftarrow_\$ \{0, 1, ..., 255\}^{32}$, compute $\mathrm{X25519}(x, \mathrm{X25519}(a, G))$. To argue about joint security we need a related public key for Ed25519 on the curve edwards25519 from [17]. Now, $\mathrm{X25519}(x, G)$ is an encoding of the u-coordinate of xG [17], where G is the basepoint on curve25519 and the arithmetic is on that curve as well. We will retrieve the two candidates for the v-coordinate from the curve equation and choose one of them uniformly at random. We then change coordinates to edwards25519 using the map in [17, 4.1]. This change of coordinates preserves addition on the curve and the basepoint of curve25519 is mapped to the basepoint on edwards25519[6]. Furthermore, the same clamping (and decoding) of private key bytestrings is used in Ed25519 [13, 5.1.5] and X25519 [17, 5]. If we guessed the correct v-coordinate, the obtained point is thus equal to

$$\mathrm{clamp}(x)G$$

where G is the basepoint on edwards25519 and clamp is the (decoding and) clamping in [13, 5.1.5]. We define a new SDH problem which is identical to the original one, but where we are additionally given $\mathrm{clamp}(x)G$ on edwards25519. As described above, if we can solve the new SDH problem with advantage ϵ, then we can solve the original one with advantage $\epsilon/2$. We accept this slight loss and assume below that we are given an SDH problem of the new form.

[6]The small number of possible values for u for which the mapping is not defined has at most a negligible probability of occurring by Lemma 1. We omit dealing with them. A more detailed or less elementary analysis might eliminate the possibility of them occurring completely for our application here.

4 Security of our KEM in the presence of an Ed25519 signing oracle

We assume that salts have always already been zero-padded to length *blocklen* in this section. Recall the IND-CCA game for a KEM with an adversary $\mathcal{A}$ from [9, 7.1.2]:

1. We generate a target key pair (x, X') using KeyGen.
2. The adversary $\mathcal{A}$ is given the public key X' and access to a decapsulation oracle Decap.
3. At some point $\mathcal{A}$ requests a challenge encapsulation (and supplies a $salt^*$ in our setting). We then generate $(K^*, A^*) = \text{Encap}(salt^*)$. After that we sample a secret bit b^* uniformly at random. If $b^* = 0$, then we update $K^* \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$. Either way, we return (K^*, A^*) to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ continues to have oracle access with the restriction it can not query $\text{Decap}(A^*, salt^*)$.
5. The adversary $\mathcal{A}$ outputs a bit b and wins if $b = b^*$.

A KEM is IND-CCA secure if $|\Pr(\mathcal{A} \text{ wins}) - 1/2|$ is sufficiently small, for all efficient probabilistic adversaries $\mathcal{A}$. Following the definition from [10] we need to show that our KEM is IND-CCA secure even when $\mathcal{A}$ is given additional access to a signing oracle that is using the public key $X = \text{clamp}(x)G$ where the arithmetic is on edwards25519 and clamp is as in Section 3.

Theorem 2. *Our KEM is IND-CCA secure in the presence of an Ed25519 signing oracle in the ROM, under the SDH assumption*[7]

Proof. We will follow the proof for [9, Theorem 9] with some changes due to our setting. Let G_0 be the original IND-CCA game for the KEM and suppose that $\mathcal{A}$ is the adversary in G_0. We let Q_{H}, Q_{Decap} and Q_{Sign} be the number of RO queries, decapsulation queries and signing queries, respectively, that $\mathcal{A}$ makes during the game. We will define a series of games G_i similar to those in [9, Theorem 9]. We also let T_i be the event that $b = b^*$ and we let F_i be the event that an abort event introduced in game G_i occurs. We replace the output bit b by a bit chosen uniformly at random if a game aborts, so in this case $\mathcal{A}$ wins with probability 1/2. We assume, without loss of generality, that all signing queries performed by $\mathcal{A}$ are distinct. We let $Z^* := \text{X25519}(x, A^*)$ and $h^* := \text{H}(salt^* \oplus in, A^*, Z^*)$.

We use the following terminology to discuss the state of the random oracle H and $\mathcal{A}$'s view of it.

- We say that a string q is *revealed* if $\mathcal{A}$ issues an RO query q or if a signature (R, s) for a message m is issued with $q = (R, X, m)$. We define Q_{Reveal} as the number of strings that are revealed during the game. We have

$$Q_{\text{Reveal}} \leq Q_{\text{H}} + Q_{\text{Sign}}$$

- We say that a string q is *evaluated* if H is applied to q within the decapsulation oracle, within the signing oracle, during the generation of the challenge encapsulation, or if q is revealed. We assume that the oracles and the encapsulation challenge generation follow the specification in Section 2. We define Q_{Eval} as the number of strings that are evaluated during the game. We have

$$Q_{\text{Eval}} \leq 2(Q_{\text{Decap}} + 1) + Q_{\text{H}} + (2Q_{\text{Sign}} + 1)$$

[7]The SDH assumption is in the X25519 setting as in Section 3.

Let k be the root secret key in Ed25519 (that is the "private key" in [13, 5.1.2]) in the following. Now, let G_1 be as G_0 but where

(i) We abort if

- k is ever evaluated outside of the deterministic commitment generation in Ed25519 [13, 5.1.6 steps 1 and 2], or if
- a bytestring starting with k' is ever evaluated outside of the deterministic commitment generation, where k' are the rightmost $hashlen/2$ bytes of $\mathrm{H}(k)$.

The probability of this occurring during the game is at most

$$2Q_{\text{Eval}} \cdot 2^{-4hashlen}$$

(ii) Let h be the $hashlen/2$ rightmost bytes of $\mathrm{H}(k)$. We abort if the signing oracle is queried at a message m such that

$$\mathrm{H}(h, m) \geq 2^{8hashlen} - 1 - (2^{8hashlen} - 1 \bmod \ell)$$

The probability of this occurring during the game is at most approximately

$$\ell Q_{\text{Sign}} \cdot 2^{-8hashlen}$$

Note that $hashlen$ is 64 (the output length SHA-512) and $\ell \approx 2^{252}$ in Ed25519. The hash output $\mathrm{H}(h, m)$ is interpreted as the scalar r in the commitment $R = rG$ for the message m. Unless abort events (i) or (ii) here occur, $\mathcal{A}$ can not distinguish our signing oracle simulation described later from the real thing.

(iii) The challenge encapsulation A^* is generated immediately at the start of the game and we abort the game if $\mathcal{A}$ happens to query the decapsulation oracle with A^* (with any associated $salt$) before it is given as the challenge. The probability of this occurring during the game is at most

$$Q_{\text{Decap}} \cdot 2^{-minentropy}$$

(iv) We abort if $q = (salt^* \oplus out, h^*)$ is evaluated – outside the challenge encapsulation generation – before $(salt^* \oplus in, A^*, Z^*)$ has been revealed. The probability of this occurring during the game is at most

$$\frac{Q_{\text{Eval}}}{2^{8hashlen} - Q_{\text{Eval}}}$$

since there are at least $2^{8hashlen} - Q_{\text{Eval}}$ choices of h^* that are consistent with the game up to the point q is evaluated. To see this, consider the moment just before q is evaluated. First, note that $(salt^* \oplus in, A^*, Z^*)$ has actually not been evaluated – outside the challenge encapsulation generation – either, since

- It is not evaluated in the deterministic commitment generation in Ed25519 [13, 5.1.6 steps 1 and 2] by abort condition (i).
- It is not the input to the inner hash invocation of the KDF for a decapsulation query by abort condition (iii).
- It is not the input to the outer hash invocation of the KDF for a decapsulation query by our condition (iii) on salt in the KDF definition.

- It has by assumption not been revealed yet.

  Thus, just before q is evaluated, we can replace h^* – that is, the output $\mathrm{H}(\mathit{salt}^* \oplus \mathit{in}, A^*, Z^*)$ – with any h such that $(\mathit{salt}^* \oplus \mathit{out}, h)$ has not been evaluated[8], and the game would still be consistent up to this point.

Let G_2 be as G_1 but where we abort the game if $(\mathit{salt}^* \oplus \mathit{in}, A^*, Z^*)$ is revealed. Thus, unless we abort, $(\mathit{salt}^* \oplus \mathit{out}, h^*)$ is never evaluated outside the challenge encapsulation generation either by abort condition (iv) in G_1. This means that

$$\Pr(T_2) = \frac{1}{2}$$

Now, for $i = 1, 2$ we have

$$|\Pr(T_{i-1}) - \Pr(T_i)| \leq \Pr(F_i)$$

by conditioning on F_i occurring. By our analysis above, F_1 is sufficiently small. So we are done if we can show that $\Pr(F_2)$ is sufficiently small. Similarly to [9, Theorem 9], we will describe an adversary $\mathcal{B}$ that solves the SDH problem by using $\mathcal{A}$, with probability sufficiently close to $\Pr(F_2)$.

As described in Section 3, in the SDH problem the adversary $\mathcal{B}$ is given two honestly generated X25519 public keys $\mathrm{X25519}(x, G)$ and $\mathrm{X25519}(a, G)$, and is asked to compute the shared secret

$$\mathrm{X25519}(x, \mathrm{X25519}(a, G))$$

Additionally, $\mathcal{B}$ is given access to an O_x oracle and an Ed25519 public key X such that $X = \mathrm{clamp}(x)G$ on edwards25519. To use $\mathcal{A}$, $\mathcal{B}$ sets the public key of the KEM in G_2 to $\mathrm{X25519}(x, G)$, the challenge encapsulation $A^* := \mathrm{X25519}(a, G)$ and $K^* \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$. Also, $\mathcal{B}$ sets the public key of the signing oracle to X. We need to show that $\mathcal{B}$ can simulate G_2 sufficiently well to $\mathcal{A}$. We essentially follow the construction in [9, Theorem 9] with some changes due to our different KDF and the additional signing oracle that $\mathcal{A}$ expects access to. To simulate the required oracles, $\mathcal{B}$ will use two lookup tables

- H which is $\mathcal{B}$'s simulation of the RO in G_2.
- $\mathcal{L}_{\mathrm{Decap}}$ for which $\mathcal{L}_{\mathrm{Decap}}(A, \mathit{salt}) = h$, if we should have $h = \mathrm{H}(\mathit{salt} \oplus \mathit{in}, A, \mathrm{X25519}(x, A))$

We write $\mathcal{L}_{\mathrm{Decap}}(A, \mathit{salt}) = \perp$, when $\mathcal{L}_{\mathrm{Decap}}$ has not been set at input (A, salt). We do the same for H. We assume that $\mathcal{B}$ does not solve the SDH problem if it aborts the simulation.
On a decapsulation query (A, salt), $\mathcal{B}$ does

1. If $A = A^*$, then it aborts the simulation. This corresponds to abort event (iii) in G_1.
2. If $\mathcal{L}_{\mathrm{Decap}}(A, \mathit{salt}) = \perp$, then it lets $\mathcal{L}_{\mathrm{Decap}}(A, \mathit{salt}) := h$ for $h \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$.
3. It then processes $q = (\mathit{salt} \oplus \mathit{out}, \mathcal{L}_{\mathrm{Decap}}(A, \mathit{salt}))$ as it would for an RO query q below.

On an RO query q, $\mathcal{B}$ does

1. If $q = (q', Z)$ for a X25519 public key Z such that $O_x(A^*, Z) = 1$, then it outputs the correct answer Z to the SDH problem.

[8]We would also switch the output values $\mathrm{H}(\mathit{salt}^* \oplus \mathit{out}, h^*)$ and $\mathrm{H}(\mathit{salt}^* \oplus \mathit{out}, h)$, and consider $(\mathit{salt}^* \oplus \mathit{out}, h^*)$ as not evaluated.

2. If $\mathrm{H}(q) \neq \perp$, then it returns $\mathrm{H}(q)$. Otherwise, it lets $h \leftarrow_{\$} \{0, 1, ..., 255\}^{hashlen}$ and continues below.

3. If $q = (salt \oplus in, A, Z)$ with $|salt \oplus in| = blocklen$ and X25519 public keys A and Z such that $\mathrm{O}_x(A, Z) = 1$, then it does

   (a) If $\mathcal{L}_{\mathrm{Decap}}(A, salt) = \perp$, then it lets $\mathcal{L}_{\mathrm{Decap}}(A, salt) := h$.

   (b) It lets $\mathrm{H}(q) := \mathcal{L}_{\mathrm{Decap}}(A, salt)$ and returns $\mathrm{H}(q)$.

4. Otherwise, it lets $\mathrm{H}(q) := h$ and returns h.

On a signing query m, $\mathcal{B}$ does

1. It lets $s \leftarrow_{\$} \{0, 1, ..., \ell - 1\}$ and $h \leftarrow_{\$} \{0, 1, ..., 255\}^{hashlen}$. Then it defines R as

$$sG - hX$$

   where the arithmetic is on the curve used by Ed25519. The distribution of R (uniform in $\langle G \rangle$) is as expected by abort event (ii) in G_1. This is the common way to simulate Schnorr signatures in the ROM introduced by Pointcheval and Stern [19].

2. If $\mathrm{H}(R, X, m) \neq \perp$, then it aborts the simulation. The probability of this occurring in the simulation is less than

$$Q_{\mathrm{Sign}} Q_{\mathrm{Eval}} \cdot 2^{-minentropy}$$

3. Otherwise, it runs through the steps above as if $\mathcal{A}$ would have queried the RO at $q = (R, X, m)$. Only, instead of choosing h randomly in Step 2 there, it uses h from here and returns the signature (R, s) instead of h in the end. If it ends up executing Step 3(a) of the RO query handling and $\mathcal{L}_{\mathrm{Decap}}(A, salt) \neq \perp$, then it aborts the simulation. The probability of this occurring during the simulation is less than

$$Q_{\mathrm{Sign}} Q_{\mathrm{Eval}} \cdot 2^{-minentropy}$$

If neither of the abort events during signing query handling described above occurs, then our simulation and G_2 are indistinguishable up until any of the abort events in G_2 occur. Also, the event F_2 must occur before any of the other abort events in G_2. We therefore have

$$\Pr(\mathcal{B} \text{ solves } SDH) \geq \Pr(F_2) - 2 Q_{\mathrm{Sign}} Q_{\mathrm{Eval}} \cdot 2^{-minentropy}$$

and $\Pr(\mathcal{B} \text{ solves } SDH)$ is sufficiently small under the SDH assumption. □

5 Security of Ed25519 in the presence of a decapsulation oracle for our KEM

Recall that in the EUF-CMA security game the adversary $\mathcal{A}$ is given access to a public key X and a signing oracle for X, and attempts to output a (valid) forged signature σ for a message m such that m was not queried to the signing oracle. If the probability that any probabilistic efficient $\mathcal{A}$ succeeds (wins) is sufficiently small, then we say that the signature scheme is EUF-CMA secure. See e.g. [7, Definition 2] for details. Following the definition in [10, 4.1], we additionally give $\mathcal{A}$ access to a decapsulation oracle for our KEM with public key $\mathrm{X25519}(x, G)$, where x is the private key bytestring of the signing oracle.

Theorem 3. *Ed25519 is EUF-CMA secure*[9] *in the presence of a decapsulation oracle for our KEM, under the SDH assumption*[10]

Proof. We let $\mathcal{A}$ be the adversary in the EUF-CMA game and we call the EUF-CMA game G_0. As in Theorem 1, we will assume that salts have always already been zero-padded to length *blocklen* before being supplied by $\mathcal{A}$. We assume that Q_{H}, Q_{Decap} and Q_{Sign} is the maximum number of RO queries, decapsulation queries and signing queries, respectively, that $\mathcal{A}$ makes during the game. The notation Q_{Reveal} and Q_{Eval}, and the terms *revealed* and *evaluated* are defined as in Theorem 1[11]. We assume without loss of generality that

- all RO queries issued by $\mathcal{A}$ are distinct,
- all decapsulation queries issued by $\mathcal{A}$ are distinct, and
- all signing queries issued by $\mathcal{A}$ are distinct.

Now, let G_1 be as G_0 but where

(i) We abort if the correponding events to those in abort events (i)-(ii) from G_1 in the proof of Theorem 1 occur.

(ii) We abort if a string $q = (\mathit{salt} \oplus \mathit{out}, h)$ is evaluated such that

  - there was previously a decapsulation query (A, salt) such that $h = \mathrm{H}(\mathit{salt} \oplus \mathit{in}, A, \mathrm{X25519}(x, A))$,
  - q had not been evaluated previously to the decapsulation query, and
  - $(\mathit{salt} \oplus \mathit{in}, A, \mathrm{X25519}(x, A))$ has not been revealed.

  The probability that this abort event occurs during the game is at most

$$\frac{Q_{\mathrm{Eval}} Q_{\mathrm{Decap}}}{2^{8\mathit{hashlen}} - Q_{\mathrm{Eval}}}$$

  since there are at least $2^{8\mathit{hashlen}} - Q_{\mathrm{Eval}}$ choices of h that are consistent with the game up to the point q is evaluated. To see this, consider the moment just before q is evaluated. First, note that $(\mathit{salt} \oplus \mathit{in}, A, \mathrm{X25519}(x, A))$ has actually not been evaluated – except once during the assumed decapsulation query – either, since

  - It is not evaluated in the deterministic commitment generation in Ed25519 [13, 5.1.6 steps 1 and 2] by abort condition (i).
  - It is not the input to the inner hash invocation of the KDF for any other decapsulation query since we assume that all decapsulation queries are distinct.
  - It is not the input to the outer hash invocation of the KDF for a decapsulation query by our condition (iii) on salt in the KDF definition.
  - It has by assumption not been revealed yet.

[9]Through a non-tight reduction, see the end of the proof for references.
[10]The SDH assumption is in the X25519 setting as in Section 3.
[11]Without the part about the challenge encapsulation generation which is not relevant here.

Thus, just before q is evaluated, we can replace h – that is, the output $\mathrm{H}(salt \oplus in, A, Z)$ – with any h' such that $(salt \oplus out, h')$ has not been evaluated[12], and the game would still be consistent up to this point.

(iii) If there is a decapsulation query $(A, salt)$ such that

- a string $(salt \oplus out, h)$ has been previously evaluated with $h = \mathrm{H}(salt \oplus in, A, \mathrm{X25519}(x, A))$, and
- $(salt \oplus in, A, \mathrm{X25519}(x, A))$ has not been revealed,

then we abort immediately after processing it. The probability that this abort event occurs during the game is at most

$$\frac{Q_{\mathrm{Decap}} Q_{\mathrm{Eval}}}{2^{8hashlen}}$$

To see this, note that $(salt \oplus in, A, \mathrm{X25519}(x, A))$ had actually not been evaluated prior to the decapsulation query, since

- We would have aborted during decapsulation query processing if it had been previously evaluated in the deterministic commitment generation in Ed25519 by abort condition (i).
- It was not the input to the inner hash invocation of the KDF for any other decapsulation query since we assume that all decapsulation queries are distinct.
- It was not the input to the outer hash invocation of the KDF for a decapsulation query by our condition (iii) on salt in the KDF definition.
- It had by assumption not been revealed yet.

Thus, $\mathrm{H}(salt \oplus in, A, \mathrm{X25519}(x, A))$ could just as well have taken any other value than h.

(iv) Let 2^c be the edwards25519 cofactor. Note that, by the verification equation [13, 5.1.7], a valid forgery signature (R^*, s^*) for a message m^* output by $\mathcal{A}$ determines $\mathrm{H}(R^*, X, m^*)$ modulo ℓ as

$$log_{2^c X}(s^* 2^c G - 2^c R^*)$$

If $\mathcal{A}$ outputs a valid forgery such that (R^*, X, m^*) has not been evaluated during the game, then we abort (before $\mathcal{A}$ wins). This happens with probability at most approximately ℓ^{-1}.

(v) If $\mathcal{A}$ outputs a valid forgery (R^*, s^*) for a message m^* such that $q = (R^*, X, m^*)$ has only been evaluated as the input to the inner hash invocation of the KDF for a decapsulation query and nowhere else, then we abort (before $\mathcal{A}$ wins). This happens with probability at most approximately ℓ^{-1}. To see this, note that abort conditions (iii) and (ii) imply that the input to the outer hash invocation in the KDF of the decapsulation query has not been evaluated – except for that one time in the decapsulation oracle – either. We can thus argue as for abort condition (ii) that we can replace the output value $\mathrm{H}(q)$ with any $h \leftarrow_{\$} \{0, 1, ..., 255\}^{hashlen}$ such that $(salt \oplus out, h)$ has not been evaluated.

(vi) If $\mathcal{A}$ outputs a valid forgery (R^*, s^*) for a message m^* such that (R^*, X, m^*) has only been evaluated inside the deterministic commitment generation in Ed25519 [13, 5.1.6 steps 1 and 2], then we abort (before $\mathcal{A}$ wins). This happens with probability at most $2^{1-4hashlen}$.

[12]We would also switch the output values $\mathrm{H}(salt \oplus out, h)$ and $\mathrm{H}(salt \oplus out, h')$, and consider $(salt \oplus out, h)$ as not evaluated.

Let T_i for $i = 1, 2$ be the event that $\mathcal{A}$ wins in G_i. Letting F_1 be the event that any abort event in G_1 occurs, we have

$$\Pr(T_1) \geq \Pr(T_0) - \Pr(F_1)$$

As in [7, Theorem 3], we will now use $\mathcal{A}$ to build an adversary $\mathcal{B}$ against the IMP-KOA security [7, Figure 2][13] of the underlying identification scheme. In [7, Theorem 3], $\mathcal{B}$ guesses the index i of the RO query that equals (R^*, X, m^*) for the forged signature (R^*, s^*) for m^* that $\mathcal{A}$ eventually outputs. What $\mathcal{B}$ does is that it takes the commitment R^* and outputs it to its own challenger in the IMP-KOA game [7, Figure 2]. After receiving a challenge $h \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$ in response from its challenger, $\mathcal{B}$ lets $\mathrm{H}(R^*, X, m^*) := h$. If (R^*, s^*) for the message m^* is a valid forgery, then $\mathcal{B}$ wins in the IMP-KOA game by just outputting s^* after $\mathcal{A}$ outputs the forgery in the end of the game. Letting ϵ' be the probability that $\mathcal{A}$ outputs a valid forgery, the main term in the success probability of $\mathcal{B}$ in its own game in [7, Theorem 3] is $\frac{\epsilon'}{Q_H}$ due to the need to guess the index i. We follow the same idea of index guessing, but to keep our proof as general as possible we account for the scenario that (R^*, X, m^*) can also be equal to one of the inputs to the RO in the KDF for a decapsulation query.

- To account for the scenario that (R^*, X, m^*) is equal to the input in the outer hash invocation of one of the KDF invocations we do the following. Instead of guessing an index i for the ith RO query, $\mathcal{B}$ will guess an index i for the ith RO/decapsulation query, where i ranges from 1 to $Q_H + Q_{\text{Decap}}$. This means that the main term in the success probability for $\mathcal{B}$ becomes $\frac{\epsilon'}{Q_H + Q_{\text{Decap}}}$, instead of $\frac{\epsilon'}{Q_H}$ as in [7, Theorem 3].
- The scenario that it is equal to the input q in the inner hash invocation of one of the KDF invocations is already covered. In that case there must additionally be an RO query for q at some other point in time by abort conditions (v)-(vi) above.

Note that, if $\mathcal{A}$ wins in G_1, then (R^*, X, m^*) must be evaluated during the game by abort condition (iv).

As in Theorem 1, we will now describe how $\mathcal{B}$ simulates the oracles in G_1 to $\mathcal{A}$. We will assume that $\mathcal{B}$ has access to an O_x oracle and an X25519 public key $\mathrm{X25519}(x, G)$, where $X = \mathrm{clamp}(x)G$ on edwards25519 for the Ed25519 public key X that $\mathcal{B}$ is given in [7, Figure 2][14]. The adversary $\mathcal{B}$ lets X be the public key of the signing oracle in G_1 and lets $\mathrm{X25519}(x, G)$ be the public key of the decapsulation oracle. The lookup tables H and $\mathcal{L}_{\text{Decap}}$ are defined as in the simulation in Theorem 1. Note that with regard to the processing related to the guessing of the ith decapsulation/RO query below we will only discuss how the processing is done when we have guessed the correct index, since we assume that $\mathcal{B}$ fails in the IMP-KOA game otherwise.

On a decapsulation query (A, salt), $\mathcal{B}$ does

1. If $\mathcal{L}_{\text{Decap}}(A, \mathit{salt}) = \perp$, then it lets $\mathcal{L}_{\text{Decap}}(A, \mathit{salt}) = h$ for $h \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$.
2. It then processes $q = (\mathit{salt} \oplus \mathit{out}, \mathcal{L}_{\text{Decap}}(A, \mathit{salt}))$ as it would for an RO query q below.

On an RO query q, $\mathcal{B}$ does

1. If this is the ith RO/decapsulation query, then

[13]Actually, $\mathcal{B}$ is an IMP-PA adversary in [7, Theorem 3], but we do not use $\mathrm{O}_{\text{Trans}}$ from [7, Figure 2].

[14]We can assume that $\mathcal{B}$ has access to what is specified here since the algorithm $\mathcal{C}$ that will use $\mathcal{B}$ to solve a discrete logarithm problem later in the proof has access this.

   (a) If $\mathrm{H}(q) \neq \perp$, then $\mathcal{B}$ aborts[15].
   (b) it parses the commitment R^* from q, sends R^* to its challenger and lets h be the received challenge.

   Otherwise, $\mathcal{B}$ lets $h \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$. Either way, it then continues below.

2. If $\mathrm{H}(q) \neq \perp$, then it returns $\mathrm{H}(q)$.
3. If $q = (salt \oplus in, A, Z)$ with $|salt \oplus in| = blocklen$ and X25519 public keys A and Z such that $\mathrm{O}_x(A, Z) = 1$, then it does
   (a) If this is the ith RO/decapsulation query[16] and if $\mathcal{L}_{\text{Decap}}(A, salt) \neq \perp$, then $\mathcal{B}$ lets
      (i) $h' := \mathcal{L}_{\text{Decap}}(A, salt)$
      (ii) $\mathcal{L}_{\text{Decap}}(A, salt) := h$
      (iii) $\mathrm{H}(salt \oplus out, h) := \mathrm{H}(salt \oplus out, h')$
      (iv) $\mathrm{H}(salt \oplus out, h') := \perp$

      and continues below. The update that we perform here is possible due to abort events (ii)-(iii) above, which imply that $(salt \oplus out, h')$ can not have been evaluated besides the single application in the decapsulation oracle, since q had not been revealed previously to this query[17].
   (b) If $\mathcal{L}_{\text{Decap}}(A, salt) = \perp$, then it lets $\mathcal{L}_{\text{Decap}}(A, salt) := h$.
   (c) It lets $\mathrm{H}(q) := \mathcal{L}_{\text{Decap}}(A, salt)$ and returns $\mathrm{H}(q)$.
4. Otherwise, it lets $\mathrm{H}(q) := h$ and returns h.

On a signing query m, $\mathcal{B}$ does

1. It lets $s \leftarrow_\$ \{0, 1, ..., \ell - 1\}$ and $h \leftarrow_\$ \{0, 1, ..., 255\}^{hashlen}$. Then it defines R as

$$sG - hX$$

   where the arithmetic is on edwards25519.

2. If $\mathrm{H}(R, X, m) \neq \perp$, then it aborts the simulation. The probability of this occurring during the simulation is less than

$$Q_{\text{Sign}} Q_{\text{Eval}} \cdot 2^{-minentropy}$$

3. Otherwise, it runs through the steps above as if $\mathcal{A}$ would have queried the RO at $q = (R, X, m)$. Only,
   - This specific RO query processing is excluded from the index guessing of the ith RO/decapsulation query that we have described above.

[15] Since we are interested in the case when $q = (R^*, X, m^*)$ here, q can not have been revealed by the signing oracle. Since we also assume that all RO queries are distinct, and that all decapsulation queries are distinct, we can only fail here if $\mathrm{H}(q)$ was previously set due to an RO query and we are now processing a decapsulation query, or vice versa. Either way, we could then initially instead have guessed the index i' of that RO/decapsulation query and succeeded. This means that the main term in the success probability for $\mathcal{B}$ remains $\frac{\epsilon'}{Q_H + Q_{\text{Decap}}}$.

[16] In this case we must be here due to an RO query (and not a decapsulation query) by condition (iii) on salt in the definition of the KDF in the KEM.

[17] The probability that $\mathrm{H}(salt \oplus out, h) \neq \perp$ prior to this processing is less than $(2^{8hashlen} - Q_{\text{Eval}})^{-1}$ and we omit dealing with it.

- Instead of choosing h randomly in Step 2 there, it uses h from here and returns the signature (R, s) instead of h in the end.

If it ends up executing Step 3(b) of the RO query handling and $\mathcal{L}_{\text{Decap}}(A, salt) \neq \perp$, then it aborts the simulation. The probability of this occurring during the simulation is less than

$$Q_{\text{Sign}} Q_{\text{Eval}} \cdot 2^{-minentropy}$$

If neither of the abort events during signing query handling described above occurs, then our simulation and G_1 are indistinguishable up until any of the abort events in G_1 occur. If $\mathcal{A}$ outputs a forgery in the end of the simulation, then $\mathcal{B}$ outputs the (response part of the) forgery as its own response, if $\mathcal{B}$ had also guessed the right decapsulation/RO query index. In conclusion, $\mathcal{B}$ succeeds with at least probability

$$\frac{\epsilon'}{Q_{\text{H}} + Q_{\text{Decap}}}$$

minus the sufficiently small sum of abort probabilities discussed above, where ϵ' is the probability that $\mathcal{A}$ outputs a valid forgery in G_0. As shown in [7, B1] an adversary $\mathcal{C}$ against our SDH problem can now use $\mathcal{B}$ and [3, Reset Lemma] to obtain the discrete logarithm clamp(x) of X through a non-tight reduction, and thus also solve its SDH instance. □

6 X448 and Ed448

One can verify that proofs corresponding to ours hold for a corresponding X448 [17] based KEM and Ed448 as defined in [13]. As in the case of Ed25519, our result does not cover any pre-hashing variant of Ed448 such as Ed448ph. Note that, when defining a corresponding SDH problem as in Section 3, the map from curve448 to edwards448 in [17] maps the basepoint of curve448 to $4G$ where G is the edwards448 basepoint. So we should multiply by 4^{-1} (where inversion is modulo the prime order of G) after mapping to edwards448. Replacing v^2 using the curve448 equation and using also [2, Lemma 1], the small number of points for which the mapping is not defined can be dealt with as in Footnote 3.

Theorem 4. *Our X448-KEM is IND-CCA secure in the presence of an Ed448 signing oracle, under the SDH assumption*

Theorem 5. *Ed448 is EUF-CMA secure in the presence of a decapsulation oracle for our X448-KEM, under the SDH assumption*

7 Conclusion and observations

- If we remove the encapsulation A from the inner hash invocation in the KDF, then we lose our IND-CCA security and instead apparently obtain a KEM with known malleability. We are not aware of any other security issues with this construction, but see for example Shoup's discussion [20, 15.6.1]. The ephemeral public key A could instead for example be added the info string in a potential later HKDF-Expand operation. We have not explicitly used the presence of A in relation to any interaction with the signature scheme in our proofs. One could therefore ask if there are maybe no "bad interactions" with the signature scheme when sharing the key pair and also not including A in the inner hash invocation. We have not investigated this setting further however. Note that for one thing, it appears that for the decapsulation oracle simulation in a proof corresponding to Theorem 1, the running time would grow like $Q_{\mathrm{Reveal}} \cdot Q_{\mathrm{Decap}}$ as discussed by Shoup [20, 15.6.1].

- Returning to the IETF draft ietf-core-oscore-groupcomm [21], we have shown that if [21] uses our KEM construction, then [21] can securely share the key pair between EdDSA and the KEM. The only difference between our KEM construction and the current one in [21] is that we input the encapsulation to key derivation, so with this small change [21] can use our result to ensure joint security for the systems in question. As discussed earlier, in the specific case of [21], the encapsulation is static and corresponds to the EdDSA public key of the sender. To leverage our result for the sender as well, [21] should add both the sender's and the recipient's EdDSA public keys to the KDF. This way, both endpoints of the pairwise mode of operation in [21] can consider themselves playing the recipient role in the KEM and leverage our result to ensure joint security for their respective key pairs. Furthermore, [21] can use the KDF from our KEM in the KEM from [10, Theorem 3] to ensure the same joint security for ECDSA and that KEM applied to any of the curves also used for ECDSA[18]. Like our result, [10, Theorem 3] does not require any domain separation for the hash function usage and this way [21] can use a single KDF construction regardless of which hash function is used for a KEM.

Acknowledgments

I want to thank John Mattsson and Göran Selander for giving comments and suggestions for a draft version of this paper.

Work on this document has in part been supported by the H2020 project SIFIS-Home (grant agreement 952652).

References

[1] M. Abdalla, M. Bellare, and P. Rogaway. Dhies: An encryption scheme based on the diffie-hellman problem. 2001.

[2] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp, and D. Riepel. Analysing the hpke standard. Cryptology ePrint Archive, Report 2020/1499, 2020. `https://eprint.iacr.org/2020/1499`.

[3] M. Bellare and A. Palacio. Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In *Annual International Cryptology Conference*, pages 162–177. Springer, 2002.

[4] D. J. Bernstein. A state-of-the-art diffie-hellman function: How do i validate curve25519 public keys? `https://cr.yp.to/ecdh.html#validate`.

[5] D. J. Bernstein. Curve25519: New diffie-hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[6] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.

[18]The proof of [10, Theorem 3] does not depend on how the KDF is implemented in terms of the underlying hash function, as long as the KDF has the required properties.

[7] J. Brendel, C. Cremers, D. Jackson, and M. Zhao. The provable security of ed25519: Theory and practice. Cryptology ePrint Archive, Report 2020/823, 2020. `https://eprint.iacr.org/2020/823`.

[8] L. Chen. Nist special publication 800-56c, recommendation for key derivation through extraction-then-expansion. *Computer Security Division, Information Technology Laboratory, Computer Security*, 2010.

[9] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive, Report 2001/108, 2001. `https://eprint.iacr.org/2001/108`.

[10] J. P. Degabriele, A. Lehmann, K. G. Paterson, N. P. Smart, and M. Strefler. On the joint security of encryption and signature in emv. Cryptology ePrint Archive, Report 2011/615, 2011. `https://eprint.iacr.org/2011/615`.

[11] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the cbc, cascade and hmac modes. In *Annual International Cryptology Conference*, pages 494–510. Springer, 2004.

[12] S. Haber and B. Pinkas. Securely combining public-key cryptosystems. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, page 215–224, New York, NY, USA, 2001. Association for Computing Machinery.

[13] S. Josefsson and I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, Jan. 2017.

[14] D. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Feb. 1997.

[15] D. H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.

[16] H. Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. Cryptology ePrint Archive, Report 2010/264, 2010. `https://eprint.iacr.org/2010/264`.

[17] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748, Jan. 2016.

[18] C. Patton and T. Shrimpton. Security in the presence of key reuse: Context-separable interfaces and their applications. Cryptology ePrint Archive, Report 2019/519, 2019. `https://eprint.iacr.org/2019/519`.

[19] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[20] V. Shoup. A proposal for an iso standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. `https://eprint.iacr.org/2001/112`.

[21] M. Tiloca, G. Selander, F. Palombini, J. P. Mattsson, and J. Park. Group OSCORE - Secure Group Communication for CoAP. Internet-Draft draft-ietf-core-oscore-groupcomm-11, Internet Engineering Task Force, Feb. 2021. Work in Progress.