X-Wing

The Hybrid KEM You've Been Looking For

Manuel Barbosa[1,2,3], Deirdre Connolly[6], João Diogo Duarte[1,2], Aaron Kaiser[3], Peter Schwabe[3,4], Karolin Varner[3,5] and Bas Westerbaan[7]

[1] University of Porto, Portugal
[2] INESC TEC, Portugal
[3] Max Planck Institute for Security and Privacy, Germany
[4] Radboud University, The Netherlands
[5] Rosenpass e.V., Germany
[6] SandboxAQ, USA
[7] Cloudflare, USA

Abstract. X-Wing is a hybrid key-encapsulation mechanism based on X25519 and ML-KEM-768. It is designed to be the sensible choice for most applications. The concrete choice of X25519 and ML-KEM-768 allows X-Wing to achieve improved efficiency compared to using a generic combiner. In this paper, we introduce the X-Wing construction and provide a proof of security. We show (1) that X-Wing is a classically IND-CCA secure KEM if the strong Diffie-Hellman assumption holds in the X25519 nominal group, and (2) that X-Wing is a post-quantum IND-CCA secure KEM if ML-KEM-768 is itself an IND-CCA secure KEM and SHA3-256 is secure when used as a pseudorandom function. The first result is proved in the ROM, whereas the second one holds in the standard model. Loosely speaking, this means X-Wing is secure if either X25519 or ML-KEM-768 are secure.

Keywords: Hybrid KEM · Post-Quantum Cryptography · Public-Key Cryptography

1 Introduction

To counter the potential threat of store-now/decrypt-later attacks using quantum computers, industry has started to deploy post-quantum key-encapsulation mechanisms (KEMs) for key agreement (cf. [O'B23, WR22]). The post-quantum cryptography that is now being considered for widespread adoption is relatively young compared to the public-key cryptography that has protected applications until now. RSA and (elliptic-curve) Diffie-Hellman might not be secure against quantum-computers, but these schemes have undergone decades of cryptographic analysis. For this reason, many opt to deploy a *hybrid* of traditional and post-quantum schemes: if the post-quantum component turns out to be weak, security falls back to the (non-post-quantum) security of traditional constructions.

As pointed out by Giacon, Heuer, and Poettering [GHP18], under the standard IND-CCA security definition for KEMs[1], creating a KEM combiner is subtler than one might

Warning. X-Wing depends on the NIST standard ML-KEM, which has not been finalised yet. Thus, X-Wing is not final yet.

E-mail: mbb@fc.up.pt (Manuel Barbosa), durumcrustulum@gmail.com (Deirdre Connolly), joao@dioguduarte.pt (João Diogo Duarte), aaron.kaiser@mpi-sp.org (Aaron Kaiser), peter@cryptojedi.org (Peter Schwabe), karo@cupdev.net (Karolin Varner), bas@westerbaan.name (Bas Westerbaan)

[1]Here, IND-CCA means the standard notion of ciphertext indistinguishability under adaptive chosen-ciphertext attacks for KEMs.

Date of this document: 2024-01-09.

expect. They show that the obvious combiner $\mathsf{KDF}(k_1\|k_2)$ is not IND-CCA secure in all cases, but that such a combiner can be made secure by mixing in the ciphertexts into the KDF input, e.g., as $\mathsf{KDF}(k_1\|k_2\|c_1\|c_2)$. As typical post-quantum KEMs have large ciphertexts, this second combiner comes with a noticeable performance penalty.

Despite this, in specific applications such as TLS 1.3, the simpler combiner is taken to be secure because the ciphertexts are mixed into the *transcript hash.* This has lead to the current situation where there are two different hybrid KEMs, both called X25519Kyber768Draft00: one for HPKE where the ciphertext is used in the key derivation input, and one for use in TLS where security relies on the *transcript hash* outside the combiner instead. This is an undesirable situation.

In this paper, we show that there is a class of KEMs where the best of both worlds can be achieved: For these KEMs, the simple, more efficient combiner yields an IND-CCA secure hybrid KEM despite not using the KEM ciphertext during key derivation. We call these KEMs *ciphertext collision resistant (CCR)* and show that ML-KEM provides ciphertext collision resistance. We use this observation to design an efficient, hybrid KEM suitable for a broad range of applications.

Besides the choice of combiner, the KEMs used in the combiner have to be decided upon as well, which involves several choices. To start the scheme. Then the paramters and security levels have to be fixed. There might be further details. For instance, for ECDH, designers need to decide upon how a KEM is to be created from the Diffie–Hellman operation like X25519 [Ber06, LHT16].

Although there is value in having a standardized recipe to create bespoke hybrid KEMs using a black-box combiner from existing KEMs [OWK23], for most use cases a single choice is beneficial for increased interoperability, and reduced engineering efforts. With X-Wing we are making concrete choices, and we provide a proof of security for these choices specifically. This simplifies our proof and yields a performance level unmet by previous constructions at the same security level.

X-Wing is a concrete KEM. Its simplicity and performance ensures it is a good choice in most use cases, including TLS and HPKE. X-Wing targets 128-bit security, and achieves this goal by combining X25519 and ML-KEM-768 using SHA3-256 as the key derivation function. ML-KEM-768 is chosen over ML-KEM-512, to hedge against advances in cryptanalysis, and to match X25519Kyber768Draft00 which already found real-world usage.

In this paper, we show that X-Wing achieves IND-CCA security based on either X25519 or ML-KEM. In the pre-quantum case, we model SHA3-256 as a random oracle and reduce the IND-CCA security of the scheme to the strong Diffie-Hellman problem in the X25519 nominal group. To achieve this result, we also show that ML-KEM-768 retains a ciphertext collision resistance, if its secret key is known to the attacker. In the post-quantum case, we give a standard model reduction from the IND-CCA security of X-Wing to IND-CCA security of ML-KEM-768 while assuming SHA3-256 to be a PRF. Here we closely follow the proof idea for KEM combiners given by Giacon, Heuer, and Poettering [GHP18], while extending it to KEMs that may allow for a small decryption error probability.

Structure of this paper: To start, in Section 2 we describe the design of X-Wing, and explain the choices made. Before getting into the details of our construction, in Section 3, we give an intuitive explanation for the security of our scheme as well as a sketch of the proof. Notation and definitions are introduced Section 4. The QSF framework is introduced in Section 5; our generic construction – termed quantum superiority fighter – constitutes a generalization of the X-Wing construction. In Section 6 we prove the security of QSFs in general, and finally, in Section 7 we show that X-Wing is a quantum superiority fighter.

2 Design

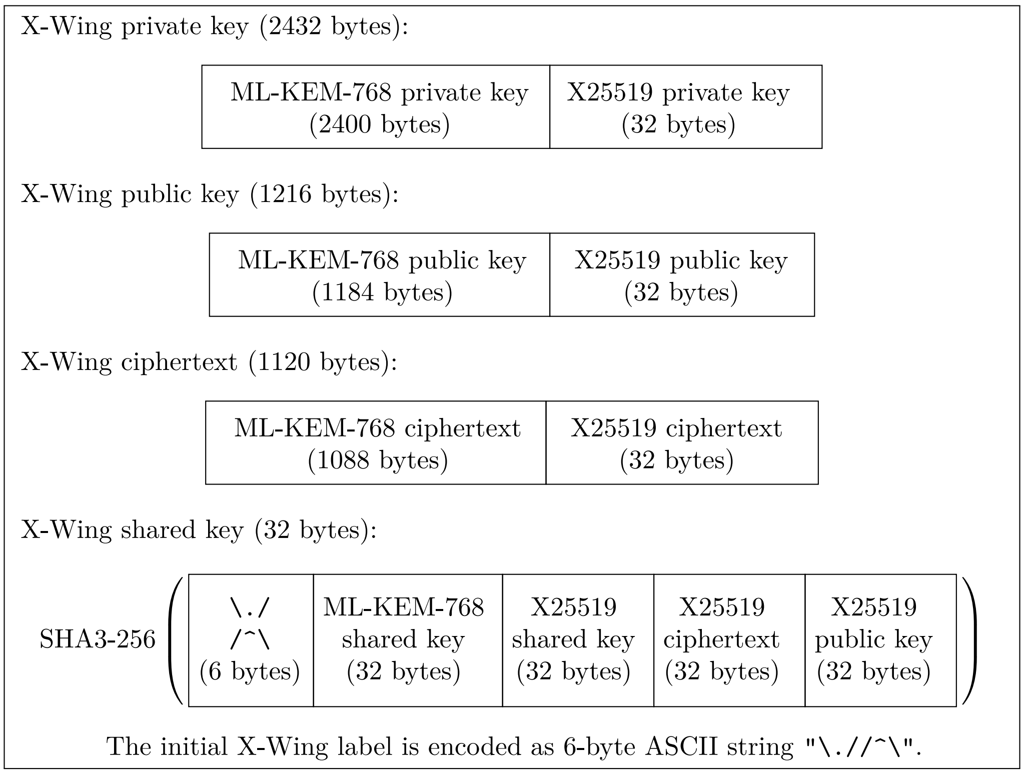
X-Wing private key (2432 bytes):
ML-KEM-768 private key (2400 bytes)
X25519 private key (32 bytes)
X-Wing public key (1216 bytes):
ML-KEM-768 public key (1184 bytes)
X25519 public key (32 bytes)
X-Wing ciphertext (1120 bytes):
ML-KEM-768 ciphertext (1088 bytes)
X25519 ciphertext (32 bytes)
X-Wing shared key (32 bytes):
SHA3-256
\./ /^\ (6 bytes)
ML-KEM-768 shared key (32 bytes)
X25519 shared key (32 bytes)
X25519 ciphertext (32 bytes)
X25519 public key (32 bytes)
The initial X-Wing label is encoded as 6-byte ASCII string "\.//^\".

Figure 1: The X-Wing KEM private key, public key, ciphertext, and shared key.

The primary goal of X-Wing is to be usable in most applications. There are many aspects to being 'usable', which can be in conflict.

First, there are the security guarantees and performance, which we already covered in the introduction: targeting IND-CCA at 128 bits with extra margin for ML-KEM, we have a solid security guarantee with comfortable margin, while retaining performance.

Secondly, there is implementation simplicity. We designed X-Wing so that it is straightforward to implement with X25519 and ML-KEM-768 as black boxes. In particular, we opted not to use the DHKEM(X25519) construction from HPKE [BBLW22, ABH+21] that turns X25519 into a KEM. Also, these considerations steered us away from utilising a standard combiner. Compared to a scheme based on DHKEM and the GHP-combiner [GHP18], we achieve the same security at a lower computational cost and implementation complexity. We stress that this optimization is possible, because of the concrete choices we made, and it may not apply in general.

We chose X25519 as it is currently the de-facto standard traditional key-agreement with excellent performance. ML-KEM [NIS23] is currently NIST's only choice for a post-quantum KEM. More importantly, these choices match closely with X25519Kyber768Draft00, which is used in the current wide-scale early deployment of post-quantum cryptography in TLS by Google and Cloudflare [O'B23, WR22]. A summary of the X-Wing design is depicted in Figure 1.

While we could further improve performance, by opening up ML-KEM and merging the key derivation phase of ML-KEM with that of X-Wing, we decided against this optimization since it would require the implementer to open up the ML-KEM implementation abstraction, which might not always be possible.

The final key-derivation includes the X25519 public key and ciphertext (i.e., both the DH long-term and ephemeral public keys). The first is added as a measure of security against multi-target attacks, similarly to what is done in the ML-KEM design.[2] Removing the X25519 ciphertext from the final key-derivation step is not possible, as X25519, if seen as a KEM, is not ciphertext collision resistant. Removing either or both tokens would not improve performance by much, since in the KDF call, we are already processing only a single SHA3-256 input block.

3 Security Intuition

Collision attacks against KEM-Combiners. Hashing public keys, shared keys, and ciphertexts are natural steps in creating a KEM combiner, but one might think that the key-derivation stage could be as uncomplicated as concatenating both shared keys before applying a key-derivation function. Let us call this the *pedestrian* combiner. Giacon, Heuer, and Poettering [GHP18] showed that such a combiner would not *robustly* provide IND-CCA security. Instead, the GHP-combiner [GHP18] that is proved secure, additionally mixes both ciphertexts into the key-derivation step.

The paper provides an in-depth explanation of the attack that can be performed if the ciphertexts are omitted. To briefly illustrate the attack, recall that a robust combiner is meant to combine two schemes so that security is preserved if either scheme (but not both) is replaced by an arbitrarily bad scheme. Consider a hypothetical bad scheme, which is broken in the following sense: given a challenge ciphertext c_1, it is easy to find another ciphertext $c_1' \neq c_1$ that decapsulates to the same shared key k_1. Then, an attacker could win the IND-CCA game against the KEM combiner as follows: Given (c_1, c_2) as the challenge ciphertext, where c_2 is the ciphertext for a secure KEM, the adversary knows this will decapsulate to $\mathsf{KDF}(k_1, k_2)$ for unknown k_2. However, the attacker can simply call its decapsulation oracle on (c_1', c_2), which is a legitimate query, and obtain the correct shared key.

Ciphertext Collision Resistant KEMs. The fact that the previous attack works for a degenerate insecure KEM does not mean that omitting a corresponding ciphertext from the KDF input always leads to an attack. Indeed in this paper we show that, for X-Wing, this is *not* the case: we can omit the (large) ML-KEM-768 ciphertext from the KDF input, and still prove that X-Wing is an IND-CCA secure KEM. Intuitively, this is because ML-KEM-768 has the following property: even if ML-KEM-768 is broken as a KEM, we show that its internal structure—based on the Fusijaki-Okamoto transform—guarantees that it is impossible to find colliding ciphertexts as described above. We call this notion *ciphertext collision-resistance* for KEMs.

Inlining the Diffie-Hellman based KEM. The X-Wing construction is not a generic KEM combiner because we do not treat the DH component as a black-box KEM. We instead take the direct route, reducing the X25519-based security of our construction to the hardness of breaking the Strong Diffie-Hellman [AP05] (SDH) problem in a nominal group. Intuitively, this means that we prove security based on a computational-Diffie-Hellman-like problem, but no assumption is made about the format of the generated group element. In particular, no assumption is made that the shared group element is indistinguishable from random bytes. Indistinguishability of the final shared secret from a random key is established by modelling the key-derivation function as a random oracle.

[2] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/C0D3W1KoINY/m/99kIvydoAwAJ

Proof-strategy. As is customary for hybrid protocols, the proof is split into two cases, one covering the possibility of quantum computers becoming a reality (and thus breaking X25519), and the other one covering the possibility of post-quantum cryptography, specifically ML-KEM-768, being classically broken.

Pre-Quantum Security: This the case case where X25519 is assumed to be secure but ML-KEM-768 may not offer any security. This proof is itself presented in two steps and, in both cases, the attacker is assumed to be classical. We first prove that X-Wing is IND-CCA secure when:

1. The SHA3-256 KDF is modelled as a Random Oracle;
2. X25519 is modelled as a nominal group in which the strong Diffie-Hellman assumption holds; and
3. ML-KEM-768 is modelled as a possibly broken KEM that provides no security beyond ciphertext collision resistance.

We then show that ML-KEM-768 satisfies the ciphertext collision resistance property, again in the ROM.

Post-Quantum Security: This is the case where ML-KEM-768 is assumed to be post-quantum secure, but X25519 offers no security. This proof is itself presented in two steps and, in both cases, the attacker is assumed to be able to perform quantum computations, but it interacts with X-Wing classically. We prove in the standard model that X-Wing is secure when:

1. The SHA3-256 KDF is modelled as a (post-quantum) PRF;
2. There is no assumption on the X25519 component; and
3. ML-KEM-768 is assumed to provide (post-quantum) IND-CCA security.

We present the above proofs in a modular way, by first justifying the X-Wing design as a generic construction, and then argue that SHA3-356, X25519 and ML-KEM-768 are good instantiations.

4 Preliminaries

4.1 Notation and conventions

For an integer n, we denote by $\mathbb{Z}_n$ the residual ring $\mathbb{Z}/n\mathbb{Z}$. $a \leftarrow\!\!\$\; A$ denotes sampling a uniformly at random from a non-empty finite set A. $\leftarrow$ denotes a deterministic assignment of a variable. $\{0,1\}^n$ is the set of all bitstrings of length n. (x, y) denotes a tuple of two elements x and y. $\mathbf{X}[y]$ denotes access into the table $\mathbf{X}$ at position y. Tables are denoted with bold uppercase variable names or $\sum$. An uninitialized position in a table is denoted with the bottom symbol $\perp$. $\mathbf{X}[\cdot] \leftarrow y$ sets all positions of table $\mathbf{X}$ to y. All algorithms are probabilistic polynomial time (PPT) unless stated otherwise. $o \leftarrow\!\!\$\; \mathcal{A}(I)$ denotes running the algorithm $\mathcal{A}$ with input I with uniform random coins and o describing its output. If $\mathcal{A}$ has additionally access to an oracle O, this is denoted as $o \leftarrow\!\!\$\; \mathcal{A}^{O(\cdot)}(I)$. A security game consists of a main procedure and optionally some oracle procedures. When a game is played, the main procedure is run and adversary $\mathcal{A}$ is given some inputs and access to the oracle procedures. Based on the output of the adversary $\mathcal{A}$ and its oracle calls, the main procedure outputs 1 or 0 depending on whether the adversary $\mathcal{A}$ won the game. If a game aborts at any time, it means that the adversary has no advantage in winning this game. In case of a decision game, this means that the game returns a random bit indicating whether

```
Game IND-CCA$^b_{\text{KEM},\mathcal{A}}$
$(sk, pk) \leftarrow\$ \text{KEM}.\mathsf{KeyGen}(\,)$
$(k_0, c^*) \leftarrow\$ \text{KEM}.\mathsf{Enc}(pk)$
$k_1 \leftarrow\$ \mathcal{K}$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k_b)$
return $b'$

Oracle $\mathsf{Dec}(c)$
if $c = c^*$ then
    return $\perp$
$k \leftarrow \text{KEM}.\mathsf{Dec}(c, sk)$
return $k$
```

Figure 2: IND-CCA security game for KEMs.

the adversary has won or not. Whenever an adversary algorithm executes "stop with x", it halts returning x to its challenger. Our security analyses are concrete, which means that we prove that the advantage of an attacker against a construction with fixed parameters is bounded by a real value that is argued to be small. Here, *small* means a summation of statistical terms and advantage terms, the former representing worst-case probabilities below 2^{-128} and the latter representing attacks on lower-level primitives that are assumed to require at least 2^{128} steps to break.

4.2 Key-Encapsulation Mechanisms

4.2.1 Syntax

Definition 1 (Key-Encapsulation Mechanism (KEM))**.** A key-encapsulation mechanism is a triple of algorithms KEM = $\{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$ with public keyspace $\mathcal{PK}$, private keyspace $\mathcal{SK}$, ciphertext space $\mathcal{C}$ and shared keyspace $\mathcal{K}$. The triple of algorithms is defined as:

- KEM.$\mathsf{KeyGen}(\,) \overset{\$}{\to} (sk, pk)$ Randomized algorithm that outputs a secret (private) key $sk \in \mathcal{SK}$ and a public key $pk \in \mathcal{PK}$.
- KEM.$\mathsf{Enc}(pk) \overset{\$}{\to} (k, c)$ Randomized algorithm that, given a public key $pk \in \mathcal{PK}$, outputs a shared key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$.
- KEM.$\mathsf{Dec}(c, sk) \to y \in \{k, \perp\}$ Deterministic algorithm that, given a secret key, $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$, returns the shared key $k \in \mathcal{K}$. In case of rejection, this algorithm returns $\perp$.

4.2.2 Correctness

The correctness of a KEM imposes that, except with small probability drawn over the coin space of KEM.KeyGen and KEM.Enc, we have that KEM.Dec correctly recovers the shared key produced by KEM.Enc. Formally, we say that a KEM is δ-correct if:

$$\Pr\left[\, k' = \perp \vee\ k \neq k' \ :\ (sk, pk) \leftarrow\$ \mathsf{KeyGen}(\,); (k, c) \leftarrow\$ \mathsf{Enc}(pk); k' \leftarrow \mathsf{Dec}(c, sk) \,\right] \leq +\delta.$$

Note that, for ML-KEM using implicit rejection, this condition simplifies to $k \neq k'$.

4.2.3 Security

The IND-CCA security game for KEMs is denoted as IND-CCA$^b_{\text{KEM},\mathcal{A}}$ and shown in Figure 2.

```
**Game** $\mathrm{PRF}^b_{\mathrm{PRF},\mathcal{A}}$
$\mathbf{T}[\cdot] \leftarrow \perp$
$k \leftarrow\!\!\$\ \mathcal{K}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Eval}(\cdot)}$
**return** $b'$

**Oracle** $\mathsf{Eval}_f(x)$
**if** $x \in \mathbf{T}$ **then**
    **return** $\mathbf{T}[x]$
$y_0 \leftarrow f(k, x);\ y_1 \leftarrow\!\!\$\ \mathcal{Y}$
$\mathbf{T}[x] \leftarrow y_b$
**return** $y_b$
```

Figure 3: PRF security games.

Definition 2 (IND-CCA advantage for KEMs)**.** The advantage of $\mathcal{A}$ in breaking the IND-CCA security of KEM is defined as

$$\mathrm{Adv}^{\mathrm{KEM}}_{\text{IND-CCA},\mathcal{A}} = \left|\Pr[\text{IND-CCA}^0_{\mathrm{KEM},\mathcal{A}} \Rightarrow 1] - \Pr[\text{IND-CCA}^1_{\mathrm{KEM},\mathcal{A}} \Rightarrow 1]\right|.$$

4.3 Pseudorandom function

A pseudorandom function (PRF) is a keyed deterministic function that cannot be distinguished from a truly random function.

Definition 3 (Pseudorandom Function (PRF))**.** For a finite keyspace $\mathcal{K}$, input space $\mathcal{X}$, finite output space $\mathcal{Y}$, and an efficiently computable function $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, we define the PRF advantage of adversary $\mathcal{A}$ based on the experiments in Figure 3 as

$$\mathrm{Adv}^{f}_{\mathrm{PRF},\mathcal{A}} = \left|\Pr[\mathrm{PRF}^0_{\mathrm{PRF},\mathcal{A}} \Rightarrow 1] - \Pr[\mathrm{PRF}^1_{\mathrm{PRF},\mathcal{A}} \Rightarrow 1]\right|.$$

4.4 Nominal Group

The construction that we introduce in this paper uses an elliptic curve. In our security proofs we abstract away the elliptic curve and use the notion of a nominal group. In this section, we recall the definition of a nominal group, as of [ABH+21].

Definition 4 (Nominal Group [ABH+21])**.** A nominal group $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$ consists of an efficiently recognisable finite set of elements $\mathcal{G}$ (also called “group elements”), a base element $g \in \mathcal{G}$, a prime p, a finite set of honest exponents $\varepsilon_h \subset \mathbb{Z}$, a finite set of exponents $\varepsilon_u \subset \mathbb{Z}\backslash p\mathbb{Z}$, and an efficiently computable exponentiation function $\mathsf{exp} : \mathcal{G} \times \mathbb{Z} \to \mathcal{G}$, where we write X^y for $\mathsf{exp}(X, y)$. The exponentiation function is required to have the following properties:

1. $(X^y)^z = X^{yz}$ for all $X \in \mathcal{G}, y, z \in \mathbb{Z}$.

2. The function ϕ defined by $\phi(x) = g^x$ is a bijection from ε_u to $\{g^x | x \in [1, p-1]\}$.

As done in [ABH+21] for a nominal group $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$, we define D_H to be the uniform distribution of honestly generated exponents ε_h and D_U to be the uniform distribution on ε_u. We also recall the definition for the statistical distance between these distributions: $\Delta_{\mathcal{N}} = \Delta[D_H, D_U] = \frac{1}{2}\sum\limits_{x\in\mathbb{Z}} |\underset{D_U}{Pr}(x) - \underset{D_H}{Pr}(x)|$ [ABH+21].

```
Algorithm KeyGen()
$sk_1, pk_1, \leftarrow$ KEM.KeyGen()
$sk_2 \leftarrow\!\!\$\ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
return $(sk, pk)$
```

```
Algorithm Enc(pk)
$(pk_1, pk_2) \leftarrow pk$
$k_1, c_1 \leftarrow$ KEM.Enc$(pk_1)$
$sk_e \leftarrow\!\!\$\ \varepsilon_h$
$c_2 \leftarrow \mathsf{exp}(g, sk_e)$
$k_2 \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k \leftarrow H(\text{label}\|k_1\|k_2\|c_2\|pk_2)$
$c \leftarrow (c_1, c_2)$
return $(k, c)$
```

```
Algorithm Dec(c, sk)
$(sk_1, sk_2) \leftarrow sk$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow$ KEM.Dec$(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ then
  return $\perp$
$k \leftarrow H(\text{label}\|k_1\|k_2\|c_2\|pk_2)$
return $k$
```

Figure 4: QSF Framework.

4.5 Strong Diffie-Hellman Problem

Definition 5 (Strong Diffie-Hellman (SDH) Problem)**.** We define the advantage function of an adversary $\mathcal{A}$ against the Strong Diffie-Hellman problem over the nominal group $\mathcal{N}$ as

$$\text{Adv}^{\text{SDH}}_{\mathcal{N},\mathcal{A}} = \Pr_{x,y\leftarrow\varepsilon_u}[Z = g^{xy} | Z \leftarrow \mathcal{A}^{DH(\cdot,\cdot)}(g^x, g^y)].$$

where DH is a decision oracle that on input (Y, Z) with $Y, Z \in \mathcal{G}$, returns 1 iff $Y^x = Z$ and 0 otherwise.

5 Introducing QSF

In this section, we introduce a general framework to build a robust KEM based on two building blocks, a nominal group and another KEM. We show that the resulting KEM is secure even if one of the building blocks loses its security properties. We call that framework QSF.

Definition 6 (QSF)**.** Let $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$ be a nominal group, let KEM be a key encapsulation mechanism and let H be a hash function. We define the QSF KEM as depicted in Fig. 4.

6 Security of QSF

In this section, we will analyse the security of the QSF framework. We will consider two cases. In the first case, we will analyse the security of QSF relative to the security of the nominal group, while in the second case, we analyse the security relative to the security of the underlying KEM.

6.1 Reduction to SDH and CCR in the ROM

As mentioned in the introduction, we introduce ciphertext collision resistance for KEMs (CCR) and show that this notion is sufficient, together with the SDH security of the nominal group, to make QSF an IND-CCA secure KEM.

Definition 7 (CCR)**.** We define the advantage function of an adversary $\mathcal{A}$ against KEM collision resistance as:

$$\mathrm{Adv}^{\mathrm{CCR}}_{\mathrm{KEM},\mathcal{A}} = \left| \Pr_{(sk,pk)\leftarrow\$\mathsf{KeyGen}(\,)} \left[\mathsf{Dec}(c_1, sk) = \mathsf{Dec}(c_2, sk) \neq \bot \wedge c_1 \neq c_2 \,|\, (c_1, c_2) \leftarrow \mathcal{A}(sk, pk) \right] \right|.$$

Note that the adversary also gets access to the secret key. This ensures that this notion will hold even if other security notions are broken, such as IND-CCA security. This allows us to prove the security of QSF relative to the strength of the nominal group for an arbitrarily bad KEM. With this definition, we can prove the IND-CCA security of QSF.

Note that this notion is related to the key-binding properties introduced by Cremers et al. in [CDM23]. Our CCR notion is strictly weaker than the M-BIND-K-CT notion introduced in that paper. We chose to introduce this additional notion because using the strictly stronger M-BIND-K-CT notion could prevent the instantiation of QSF with a KEM that is not M-BIND-K-CT, but would result in an overall secure construct.

Theorem 1. *Let $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$ be a nominal group and KEM be a CCR secure key encapsulation mechanism, let H be a random oracle with an output size of n and let $\mathcal{A}$ be an adversary against the IND-CCA security of QSF making at most q_h queries to the random oracles. Then there exist adversaries $\mathcal{B}$ and $\mathcal{C}$ such that,*

$$Adv^{IND\text{-}CCA}_{QSF,\mathcal{A}} \leq 2\Delta_{\mathcal{N}} + Adv^{SDH}_{\mathcal{N},\mathcal{B}} + Adv^{CCR}_{KEM,\mathcal{C}}.$$

The run-times of $\mathcal{B}$ and $\mathcal{C}$ are roughly the same as of $\mathcal{A}$. $\mathcal{B}$ performs at most $2q_h$ queries to its own DH oracle.

Proof. Let us consider the following games,

Game 0 Let G_0 be defined as in Figure 5. Clearly, G_0 is the IND-CCA^0 game instantiated with QSF. By definition,

$$\Pr[\text{IND-CCA}^0_{\mathrm{QSF},\mathcal{A}} \Rightarrow 1] = \Pr[G_0^{\mathcal{A}} \Rightarrow 1].$$

Game 1 For G_1 we choose the secret keys sk_2 and sk_e from the set of exponents ε_u instead of the set of honest exponents ε_h. This allows us to use the SDH problem to bound the probability of the next game hop.

Claim 1

$$|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]| \leq 2\Delta_{\mathcal{N}}.$$

Proof With this game hop, we change the distribution from which these values are chosen. $\Delta_{\mathcal{N}}$ defines the bound on the probability of an adversary to distinguish the original distribution from the new one for one element. As we are replacing the distribution of two elements, we get $2\Delta_{\mathcal{N}}$ as an upper bound for $\mathcal{A}$ to detect this change.

Game 2 With the changes in G_2 we prevent the adversary from querying the random oracle containing the value k_2^* that was used to generate the challenge key k^*. This prevents $\mathcal{A}$ entirely from obtaining the shared challenge key k^* from the random oracle.

```
Game $G_0/G_1/G_2/G_3/G_4$
$\sum[\cdot] \leftarrow \perp$
$\boxed{\mathbf{K}[\cdot] \leftarrow \perp \quad \triangleright G_3 - G_4}$
$(sk_1, pk_1) \leftarrow \mathrm{KEM.KeyGen}()$
$sk_2 \leftarrow\!\!\$\ \varepsilon_h$
$\boxed{sk_2 \leftarrow\!\!\$\ \varepsilon_u \quad \triangleright G_1 - G_4}$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$(k_1^*, c_1^*) \leftarrow \mathrm{KEM.Enc}(pk_1)$
$\boxed{\mathbf{K}[k_1] \leftarrow c_1 \quad \triangleright G_3 - G_4}$
$sk_e \leftarrow\!\!\$\ \varepsilon_h$
$\boxed{sk_e \leftarrow\!\!\$\ \varepsilon_u \quad \triangleright G_1 - G_4}$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(c_2, sk_e)$
$c^* \leftarrow (c_1^*, c_2^*)$
$s \leftarrow \mathrm{label} \| k_1^* \| k_2^* \| c_1^* \| pk_2$
if $\sum[s] = \perp$ then
    $\sum[s] \leftarrow\!\!\$\ \{0,1\}^n$
$k^* \leftarrow \sum[s]$
$\boxed{k^* \leftarrow\!\!\$\ \{0,1\}^n \quad \triangleright G_4}$
$b' \leftarrow \mathcal{A}^{\mathsf{Dec}(\cdot), H(\cdot)}(pk, c^*, k^*)$
return $b'$
```

```
Oracle $\mathsf{Dec}(c)$
if $c = c^*$ then
    return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \mathrm{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ then
    return $\perp$
$\triangleright G_3 - G_4 \quad \triangleleft$
if $\mathbf{K}[k_1] \neq \perp \wedge \mathbf{K}[k_1] \neq c_1$ then
    $abort_2$
$s \leftarrow \mathrm{label} \| k_1 \| k_2 \| c_2 \| pk_2$
if $\sum[s] = \perp$ then
    $\sum[s] \leftarrow\!\!\$\ \{0,1\}^n$
$k \leftarrow \sum[s]$
return $k$
```

```
$H(m)$
$\triangleright G_2 - G_4 \quad \triangleleft$
if $\mathrm{label} \| k_1 \| k_2 \| c_2 \| pk_2 \leftarrow m$ then
    if $k_2 = k_2^*$ then
        $abort_1$
if $\sum[m] = \perp$ then
    $\sum[m] \leftarrow\!\!\$\ \{0,1\}^n$
return $\sum[m]$
```

Figure 5: Game $G_0 - G_5$.

```
**Adversary** $\mathcal{B}^{\mathsf{DH}(\cdot,\cdot)}(X,Y)$
$\sum[\cdot] \leftarrow \perp$
$\mathbf{E}[\cdot] \leftarrow \perp$
$(sk_1, pk_1) \leftarrow \text{KEM}.\mathsf{KeyGen}()$
$pk_2 \leftarrow X$
$pk \leftarrow (pk_1, pk_2)$
$(k_1^*, c_1^*) \leftarrow \text{KEM}.\mathsf{Enc}(pk_2)$
$c_2^* \leftarrow Y$
$c^* \leftarrow (c_1^*, c_2^*)$
$k^* \leftarrow\$ \{0,1\}^n$
$b' \leftarrow \mathcal{A}^{\mathsf{Dec}(\cdot),H(\cdot)}(pk, c^*, k^*)$
```

```
**Oracle** $\mathsf{Dec}(c)$
**if** $c = c^*$ **then**
  **return** $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM}.\mathsf{Dec}(c_1, sk_1)$
**if** $k_1 = \perp$ **then**
  **return** $\perp$
**if** $\mathbf{E}[(c_2, k_2)] = \perp$ **then**
  $\mathbf{E}[(c_2, k_2)] \leftarrow\$ \{0,1\}^n$
**return** $\mathbf{E}[(c_2, k_2)]$

$H(m)$
**if** $\text{label}\|k_1\|k_2\|c_2\|pk_2 \leftarrow m$ **then**
  **if** $\mathsf{DH}(Y, k_2) = 1$ **then**
    stop with $k_2$
  **if** $\mathsf{DH}(c_2, k_2) = 1$ **then**
    **if** $\mathbf{E}[(c_2, k_2)] = \perp$ **then**
      $\mathbf{E}[(c_2, k_2)] \leftarrow\$ \{0,1\}^n$
    **return** $\mathbf{E}[(c_2, k_2)]$
**if** $\sum[m] = \perp$ **then**
  $\sum[m] \leftarrow\$ \{0,1\}^n$
**return** $\sum[m]$
```

Figure 6: Adversary $\mathcal{B}$.

Claim 2

$$|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]| \leq Pr[abort_1] = \text{Adv}^{\text{SDH}}_{\mathcal{N},\mathcal{B}}.$$

Proof To prove this claim, we show the existence of adversary $\mathcal{B}$ that wins the SDH game exactly when $\mathcal{A}$ would query H_2 containing the k_2^* used to generate k^*. Consider adversary $\mathcal{B}$ from Figure 6 playing the SDH game and simulation $\mathcal{A}$'s view on the IND-CCA game.

Adversary $\mathcal{B}$ uses the challenge group elements $X = \mathsf{exp}(g, x)$ as the static public key and $Y = \mathsf{exp}(g, y)$ as the ephemeral public key for the challenge ciphertext. Therefore, the shared challenge key k^* is supposed to be $H_2(label\|k_1^*\|\mathsf{exp}(Y, x)\|c_2^*\|pk_2)$. $\mathcal{B}$ is unable compute $\mathsf{exp}(Y, x)$ and therefore sets k^* to a uniform random value. For $\mathcal{A}$ to win the IND-CCA game, it would have to query H_2 with $\mathsf{exp}(Y, x)$. $\mathcal{B}$ can use the DH oracle to detect when $\mathcal{A}$ queries H_2 with the result of $\mathsf{exp}(Y, x)$, since $\mathsf{DH}(Y, k_2^*) = 1$ iff $k_2^* = \mathsf{exp}(Y, x)$. Therefore, if this check, in the H oracle, evaluates to 1 we know that the k_2 provided has to equal k_2^*. When this happens, $\mathcal{A}$ has provided $\mathcal{B}$ with the solution to the SDH game. A similar approach can be used to simulate the Dec oracle. When calculating the shared key in the Dec oracle, the input for the random oracle has to have the form $label\|k_1\|\mathsf{exp}(c_2, x)\|c_2\|pk_2$ which $\mathcal{B}$ is not able to compute. $\mathcal{B}$ can use the same method as before to check whether $\mathcal{A}$ performs such a query using the DH oracle. We then use the table $\mathbf{E}$ to ensure consistency between the random oracle H and the Dec oracle. Therefore, $\mathcal{B}$ simulates $\mathcal{A}$'s view on the IND-CCA game perfectly and wins the SDH game iff the adversary queries H with

```
Adversary $\mathcal{C}(sk_1, pk_1)$
$\sum[\cdot] \leftarrow \perp$
$\mathbf{K}[\cdot] \leftarrow \perp$
$sk_2 \leftarrow\!\!\$\ \varepsilon_u$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$(k_1^*, c_1^*) \leftarrow \text{KEM.}\mathsf{Enc}(pk_1)$
$\mathbf{K}[k_1^*] \leftarrow c_1^*$
$sk_e \leftarrow\!\!\$\ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(c_2, sk_e)$
$c^* \leftarrow (c_1^*, c_2^*)$
$s \leftarrow \text{label}\|k_1^*\|k_2^*\|c_1^*\|pk_2$
if $\sum[s] = \perp$ then
  $\sum[s] \leftarrow\!\!\$\ \{0,1\}^n$
$k^* \leftarrow \sum[s]$
$b' \leftarrow \mathcal{A}^{\mathsf{Dec}(\cdot),H(\cdot)}(pk, c^*, k^*)$
```

```
Oracle $\mathsf{Dec}(c)$
if $c = c^*$ then
  return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ then
  return $\perp$
if $\mathbf{K}[k_1] \neq \perp \wedge \mathbf{K}[k_1] \neq c_1$ then
  stop with $(\mathbf{K}[k_1], c_1)$
$k \leftarrow H_2(\text{label}\|k_1\|k_2\|c_2\|pk_2)$
return $k$

$H(m)$
if $\text{label}\|k_1\|k_2\|c_2\|pk_2 \leftarrow m$ then
  if $k_2 = k_2^*$ then
    $abort_1$
if $\sum[m] = \perp$ then
  $\sum[m] \leftarrow\!\!\$\ \{0,1\}^n$
return $\sum[m]$
```

Figure 7: Adversary $\mathcal{C}$.

the result of $\mathsf{exp}(Y, x)$, which is the k_2^* used to generate k^*. Note that the adversary $\mathcal{B}$ makes at most $2q_h$ queries to the DH oracle.

Game 3 G_3 introduces changes that prevent the adversary from querying the Dec oracle with a ciphertext c_1 that is different from the ciphertext c_1^* that was used to generate the challenge ciphertext c^*, but results in the same k_1 as the k_1^* that was used to generate k^*.

Claim 3

$$|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]| \leq Pr[abort_2] = \text{Adv}^{\text{CCR}}_{\text{KEM},\mathcal{C}}.$$

Proof To prove this claim, we show the existence of an adversary $\mathcal{C}$, depicted in Figure 7, that simulates $\mathcal{A}$'s view on the IND-CCA game and wins iff $\mathcal{A}$ would trigger $abort_2$.

All oracles are simulated perfectly. This is easily possible, since $\mathcal{C}$ also obtains the secret key from its challenger. $\mathcal{C}$ stops exactly when the $abort_2$ would be hit in G_3. When the stop instruction is reached, $\mathcal{C}$ obtained a k_1 that is equal to k_1^* but where c_1 and c_1^* are different. This means that $\mathcal{C}$ found two different ciphertext that decapsulate to the same shared key under the same secret key. These ciphertexts are a valid solution for the CCR game.

Game 4 In G_4 we replace the shared challenge key k^* with a uniform random key. With this change we reached the IND-CCA^1 game instantiated with QSF. Therefore,

Claim 4

$$\Pr[G_4^{\mathcal{A}} \Rightarrow 1] = \Pr[\text{IND-CCA}^1_{\text{QSF},\mathcal{A}} \Rightarrow 1].$$

```
Game $G_0/G_1/G_2/G_3/G_4$
$sk_1, pk_1 \leftarrow \mathrm{KEM.KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
$k_1^*, c_1^* \leftarrow \mathrm{KEM.Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(c_2^*, sk_e)$
$\boxed{k_1^* \leftarrow\$ \mathcal{K} \qquad \triangleright G_1 - G_3}$
$k^* \leftarrow H(\mathrm{label}\|k_1^*\|k_2^*\|c_1^*\|pk_2)$
$\boxed{k^* \leftarrow\$ \{0,1\}^n \qquad \triangleright G_2 - G_4}$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return $b'$

Oracle $\mathsf{Dec}(c)$
if $c = c^*$ then
    return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \mathrm{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ then
    return $\perp$
$\boxed{\text{if } c_1 = c_1^* \text{ then } \quad \triangleright G_1 - G_3}$
$\boxed{\quad k_1 \leftarrow k_1^*}$
$k \leftarrow H(\mathrm{label}\|k_1\|k_2\|c_1\|pk_2)$
$\boxed{\text{if } c_1 = c_1^* \text{ then } \quad \triangleright G_2}$
$\boxed{\quad k \leftarrow\$ \{0,1\}^n}$
return $k$
```

Figure 8: Games G_0 to G_4.

Proof To prove that this change is not detectable by $\mathcal{A}$ we need to argue that $\mathcal{A}$ cannot obtain k^* by calling any of the oracles. The abort condition in H prevents the adversary from querying the random oracle with the value of k_2^* used to generate the shared key from the challenge. This prevents $\mathcal{A}$ from calling H directly with the input used to generate k^*. The abort condition in Dec also prevents the adversary from obtaining the hash value used to generate k^* since either c_1 or c_2 must differ from the challenge ciphertext c^*. If c_1 differs, then the resulting key k_1 has to differ from k_1^*, as otherwise the abort condition would have been triggered. If c_2 differs, then the input to the random oracle has to also differ, since it is included in the input. Therefore, $\mathcal{A}$ is not able to obtain k^* from any of the oracles.

This concludes the proof of Theorem 1.

□

6.2 Reduction to security of KEM in the standard model

Theorem 2. *QSF is IND-CCA secure as long as KEM is IND-CCA secure and H is a secure PRF with an output length of nwhen keyed on k_1. More precisely, for any PPT adversary $\mathcal{A}$ against the IND-CCA security of QSF placing at most q_d decapsulation queries, we can construct adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{C}_1$, and $\mathcal{C}_2$, such that,*

$$Adv^{QSF}_{IND\text{-}CCA,\mathcal{A}} \leq Adv^{KEM}_{IND\text{-}CCA,\mathcal{B}_1} + Adv^{KEM}_{IND\text{-}CCA,\mathcal{B}_2} + Adv^{H}_{PRF,\mathcal{C}_1} + Adv^{H}_{PRF,\mathcal{C}_2} + 2\delta\,,$$

where δ is the correctness bound for KEM. The run-times of $\mathcal{B}_i$ and $\mathcal{C}_i$ are roughly the same as that of $\mathcal{A}$. Adversaries $\mathcal{B}_i$ place at most q_d queries to their own decapsulation oracles.

Proof. This proof follows closely the proof for [GHP18, Theorem 1].

Game 0 This is the standard IND-CCA security game for QSF and so,

$$\Pr[\text{IND-CCA}^0_{\text{QSF},\mathcal{A}} \Rightarrow 1] = \Pr[G_0 \Rightarrow 1].$$

Game 1 The challenge shared key produced by KEM is replaced with a random key.

Claim 1 This change should not be noticeable by the adversary $\mathcal{A}$. If it is, then we can construct an efficient adversary $\mathcal{B}_1$ against the IND-CCA security of KEM with roughly the same running time as $\mathcal{A}$, such that,

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \text{Adv}_{\text{KEM}}^{\text{IND-CCA},\mathcal{B}_1} + \delta.$$

Proof We construct $\mathcal{B}_1$ as in Figure 9. The adversary $\mathcal{B}_1$ simulates the environment of $\mathcal{A}$ by calculating the nominal group components of QSF itself and embedding the KEM challenge into the QSF challenge. The decapsulation oracle is simulated by $\mathcal{B}_1$ as follows: it can calculate the nominal group part itself and query its own KEM decapsulation oracle on all ciphertexts except c_1^*. Because of this, it will simply use its own challenge shared key k_1^* if c_1^* is in the decapsulation query by $\mathcal{A}$.

The strategy adopted by $\mathcal{B}_1$ correctly interpolates between games G_0 and G_1, except when c_1^* would decapsulate to something other than k_1^* in G_0. We can bound the probability of this inconsistency using the correctness of KEM, which justifies the δ term in the claim. More precisely, we have

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G^0_{\text{KEM},\mathcal{B}_1} \Rightarrow 1]\right| \leq \delta,$$

and

$$\Pr[G_1^{\mathcal{A}} \Rightarrow 1] = \Pr[G^1_{\text{KEM},\mathcal{B}_1} \Rightarrow 1],$$

which means that

$$\left|\Pr[G_0^{\mathcal{A}} \Rightarrow 1] - \Pr[G_1^{\mathcal{A}} \Rightarrow 1]\right| \leq \text{Adv}^{\text{KEM}}_{\text{IND-CCA},\mathcal{B}_1} + \delta,$$

and $\mathcal{B}_1$ will clearly perform at most as many decapsulation queries to its KEM decapsulation oracle as $\mathcal{A}$ makes decapsulation queries.

Game 2 In this game, all QSF shared keys that are provided to the adversary computed using k_1^* are replaced with random values.

Claim 2 This change should not be noticeable by the adversary $\mathcal{A}$. If it is, then we can construct an efficient PPT adversary $\mathcal{C}_1$ against the PRF security of H with roughly the same running time as $\mathcal{A}$, such that

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \text{Adv}^{H}_{\text{PRF},\mathcal{C}_1}.$$

Proof This is a simple reduction shown in Figure 10. This adversary can generate all cryptographic parameters except k_1^*, for which it uses its PRF oracle. As we can see, when the challenge PRF key is real, then $\mathcal{C}_1$ will be using a real output of the pseudorandom function and hence running G_1; whereas if the challenge key is random, $\mathcal{C}_1$ will be using a uniformly sampled output and running G_2. Hence,

$$\Pr[G_1^{\mathcal{A}} \Rightarrow 1] = \Pr[G^0_{H,\mathcal{C}_1} \Rightarrow 1],$$

and

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[G_{H,\mathcal{C}_1}^1 \Rightarrow 1],$$

which means that

$$\left|\Pr[G_1^{\mathcal{A}} \Rightarrow 1] - \Pr[G_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathrm{Adv}_{\mathrm{PRF},\mathcal{C}_1}^{H},$$

and $\mathcal{C}_1$ will perform one evaluation query for generating the challenge key and perform at most one evaluation query per decapsulation query by $\mathcal{A}$.

Game 3 We undo the modification introduced in the previous game, but only for keys output by the decapsulation oracle.

Claim 3 We justify this hop with a reduction to the PRF property of H very similar to the previous one.

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathrm{Adv}_{\mathrm{PRF},\mathcal{C}_2}^{H}.$$

Proof We construct $\mathcal{C}_1$ as in Figure 10, and the analysis is similar to the previous hop.

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathrm{PRF}_{H,\mathcal{C}_2}^0 \Rightarrow 1],$$

and

$$\Pr[G_3^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathrm{PRF}_{H,\mathcal{C}_2}^1 \Rightarrow 1],$$

which means that

$$\left|\Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathrm{Adv}_{\mathrm{PRF},\mathcal{C}_2}^{H},$$

and $\mathcal{C}_2$ will perform at most one evaluation query per decapsulation query by $\mathcal{A}$ as it will never call the evaluation oracle in its main algorithm.

Game 4 Finally, we revert the changes introduced in Game 1 and use a real key for k_1^* once more. This makes the decapsulation oracle identical to Game 0, and the only remaining change in the main experiment is that k^* is random.

Claim 4 This hop is justified in a way that is very similar to the jump to Game 1. We introduce adversary $\mathcal{B}_2$ against the IND-CCA security of KEM, with roughly the same running time as $\mathcal{A}$, in Figure 9. We claim that,

$$\left|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathrm{Adv}_{\text{IND-CCA},\mathcal{B}_2}^{\mathrm{KEM}} + \delta\,.$$

Proof Again, the reduction to the IND-CCA security of KEM is perfect, except if c_2^* would not decapsulate to k_1^* in Game 4. This event can be bound by the correctness of KEM, and the claim follows.

Claim 5 $\Pr[G_4^{\mathcal{A}} \Rightarrow 1] = \Pr[\text{IND-CCA}_{\mathrm{QSF},\mathcal{A}}^1 \Rightarrow 1]$

Proof This follows from the definition of $\text{IND-CCA}_{\mathrm{QSF},\mathcal{A}}^1$.

The theorem follows from collecting all the terms in the claims.

$\square$

```
Adversary $\mathcal{B}_{1,2}^{\mathsf{Dec}_O(\cdot)}(pk_1, k_1^*, c_1^*)$
$sk_2 \leftarrow\!\!\$\ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk_e \leftarrow\!\!\$\ \varepsilon_u$
$k_1, c_1^* \leftarrow \mathrm{KEM}.\mathsf{Enc}(pk_1)$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(c_2^*, sk_e)$
$k^* \leftarrow H(\mathrm{label}\|k_1^*\|k_2^*\|c_1^*\|pk_2)$
$k^* \leftarrow\!\!\$\ \{0,1\}^n$                    $\triangleright \mathcal{B}_2$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return $b'$

Oracle $\mathsf{Dec}(c)$
if $c = c^*$ then
    return $\perp$
$(c_1, c_2) \leftarrow c$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $c_1 = c_1^*$ then
    $k_1 \leftarrow k_1^*$
else $k_1 \leftarrow \mathsf{Dec}_O(c_1)$
if $k_1 = \perp$ then
    return $\perp$
$k \leftarrow H(\mathrm{label}\|k_1\|k_2\|c_1\|pk_2)$
return $k$
```

Figure 9: Adversary $\mathcal{B}_1$ and $\mathcal{B}_2$ against the IND-CCA security of KEM. Note that both place at most q_d queries to Dec_O, one for each decapsulation query placed by $\mathcal{A}$.

```
Adversary $\mathcal{C}_{1,2}^{\mathsf{Eval}_H(\cdot)}()$
$sk_1, pk_1 \leftarrow \mathrm{KEM}.\mathsf{KeyGen}()$
$sk_2 \leftarrow\!\!\$\ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
$k_1, c_1^* \leftarrow \mathrm{KEM}.\mathsf{Enc}(pk_1)$
$sk_e \leftarrow\!\!\$\ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(c_2^*, sk_e)$
$k^* \leftarrow \mathsf{Eval}_H(\mathrm{label}\|k_2^*\|c_2^*\|pk_2)$
$k^* \leftarrow\!\!\$\ \{0,1\}^n$                    $\triangleright \mathcal{C}_2$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return $b'$

Oracle $\mathsf{Dec}(c)$
if $c = c^*$ then
    return $\perp$
$(c_1, c_2) \leftarrow c$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $c_1 = c_1^*$ then
    $k \leftarrow \mathsf{Eval}_H(\mathrm{label}\|k_2\|c_1\|pk_2)$
    return $k$
$k_1 \leftarrow \mathrm{KEM}.\mathsf{Dec}(sk_1, c_1)$
if $k_1 = \perp$ then
    return $\perp$
$k \leftarrow H(\mathrm{label}\|k_1\|k_2\|c_1\|pk_2)$
return $k$
```

Figure 10: Adversary $\mathcal{C}_1$ and $\mathcal{C}_2$ against the PRF security of H.

7 X-Wing

Now that we have proven the security of the QSF framework, we want to introduce one concrete instantiation of QSF using X25519, ML-KEM-768 and SHA3-256. We believe that this instantiation provides a secure and efficient KEM suitable for most applications. As a name for this instantiation, we choose X-Wing. For our QSF proof to apply to X-Wing we need to show that X25519 can be modelled as a nominal group and that ML-KEM-768 is CCR secure. This is done in this section, followed by the definition of X-Wing.

7.1 X25519 is a nominal group

X-Wing uses X25519 as specified in [LHT16]. As shown in [ABH+21] X25519 can be modelled as a nominal group $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$. We summarise the results in this section.

We first define $\mathcal{G}$ to be all 256 bit long bitstrings. We then define an encoding function $\mathsf{encode\_pk}$ as well as a decoding function $\mathsf{decode\_pk}$, mapping point on Curve25519 to a bitstring and vice versa. For this, we can use the encoding and decoding functions as defined in [LHT16]. p is the order of the largest prime order subgroup and g is a Curve25519 point of order p, such that the number of points on Curve25519 is $8p$. $\varepsilon_h = \{8n | n \in [2^{251}, 2^{252} - 1]\}$ stands for all valid secret keys, respectively for the key generation function of X25519. $\mathsf{exp}(X, y) = \mathsf{encode\_pk}(y \cdot \mathsf{decode\_pk}(X))$, which corresponds to the X25519.DH function. $\varepsilon_u = \{8n | n \in [(p + 1)/2, p - 1]\}$. This provides us with a statistical difference between the uniform distribution over ε_u and ε_h of $\Delta_{\mathcal{N}} < 2^{-126}$ [ABH+21].

Therefore, the definition of this nominal group matches the definition of X25519 perfectly.

7.2 Collision Resistance of ML-KEM-768

For the security of X-Wing it is important that ML-KEM-768 used is CCR secure. We will show the ciphertext collision resistance of ML-KEM-768 in this section.

Theorem 3 (ML-KEM-768 is CCR secure)**.** *Let k, d_u, d_v be integers greater than or equal to 1, let PKE.Enc and PKE.Dec be deterministic algorithms and let G and J be independent random oracles. For any PPT $\mathcal{A}$ we get,*

$$Adv^{CCR}_{\text{ML-KEM-768},\mathcal{A}} \leq \frac{3}{2^{128}}.$$

Proof. Let us recall the definition of the decapsulation algorithm of ML-KEM-768, depicted in Figure 11.

We notice that the shared key K can be the original shared key K resulting from the hash function call to G, in the case of an accepted ciphertext c, or the result of the hash function J, in the case of an implicit reject. This leaves us with three cases to consider. The first is that two distinct accepting ciphertexts result in the same shared key. The second case is that two distinct rejecting ciphertexts result in the same shared key. The last case is that one accepting and one rejecting ciphertext result in the same shared key. For each of the cases we have to consider the probability of the inputs to the hash functions colliding and the probability that their outputs collide.

Case 1: Two distinct accepting ciphertext result in the same shared key

Case 1.1: The input to G collides Let us assume that PKE.Dec outputs the same m' for two distinct ciphertexts c_1 and c_2. Then the intermediate values K and r' are the same during both decapsulations. Since PKE.Enc is deterministic, c' is the same for both

```
$\underline{\textbf{ML-KEM-768}.\mathsf{Dec}(sk \in \{0,1\}^{768k+96}, c \in \{0,1\}^{256(d_u k+d_v)})}$
$sk_{PKE} \leftarrow sk[0 : 384k]$
$pk_{PKE} \leftarrow sk[384k : 768k + 32]$
$h \leftarrow sk[768k + 32 : 768k + 64]$
$z \leftarrow sk[768k + 64 : 768k + 96]$
$m' \leftarrow \text{PKE}.\mathsf{Dec}(sk_{PKE}, c)$
$(K, r') \leftarrow G(m' \| h)$
$\overline{K} \leftarrow J(z \| c, 32)$
$c' \leftarrow \text{PKE}.\mathsf{Enc}(pk_{PKE}, m', r')$
$\textbf{if}\ \ c \neq c'\ \ \textbf{then}$
  $K \leftarrow \overline{K}$
$\textbf{return}\ K$
```

Figure 11: ML-KEM-768 decapsulation [NIS23]

decapsulations. This also means that the check $c_i = c'$ cannot succeed for both ciphertexts. This leads to a contradiction, as at least one of the ciphertext has to be rejected.

Case 1.2: The output of G collides The output of G is interpreted as the two values K and r'. An adversary can already find a collision in the decapsulation function if it can find two inputs to G that result in the same output K. When modelling G as a random oracle, we can bound the probability of K colliding independent of r' by using the generic birthday bound. K is a bitstring of length 256, which gives us as collision probability of $\frac{1}{2^{128}}$.

Case 2: Two distinct rejecting ciphertexts result in the same shared key

Case 2.1: The input to J collides Since the ciphertext is part of the input to J, and both ciphertexts must be different, the inputs cannot collide.

Case 2.2: The output of J collides Since J is also modelled as a random oracle, we can use the generic birthday bound to bound the probability of two outputs colliding. We again get a probability of K colliding of $\frac{1}{2^{128}}$, as K is a 256-bit bitstring.

Case 3: An accepting and a rejecting ciphertext result in the same shared key

Case 3.1: The input to G and J collide This cannot happen as the input to G and J are of different size. G gets as input a 256-bit-long bitstring, while the ciphertext, which gets passed into J, already has a size greater than 256 bits.

Case 3.2: The outputs of G and J collide Since both functions are modelled as a random oracle, we again get the generic birthday bound of K and K' colliding with $\frac{1}{2^{128}}$.

This concludes the proof for Theorem 3. □

Note that all KEMs that are the result of the Fujisaki-Okamoto transformation and use explicit rejection are also CCR secure, since the argument of Case 1 holds for them as well.

```
Algorithm KeyGen()
$sk_1, pk_1 \leftarrow$ ML-KEM-768.$\mathsf{KeyGen}()$
$sk_2 \leftarrow \mathsf{random}(32)$
$pk_2 \leftarrow$ X25519.$\mathsf{DH}(sk_2, g_{\text{X25519}})$
$sk \leftarrow (sk_1, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
return $(sk, pk)$
```

```
Algorithm Enc(pk)
$(pk_1, pk_2) \leftarrow pk$
$sk_e \leftarrow \mathsf{random}(32)$
$c_2 \leftarrow$ X25519.$\mathsf{DH}(sk_e, g_{\text{X25519}})$
$k_1, c_1 \leftarrow$ ML-KEM-768.$\mathsf{Enc}(pk)$
$k_2 \leftarrow$ X25519.$\mathsf{DH}(sk_e, pk_2)$
$s \leftarrow$ "\.//^\"$\|k_1\|k_2\|c_2\|pk_2$
$k \leftarrow$ SHA3-256$(s)$
$c \leftarrow (c_1, c_2)$
return $(k, c)$
```

```
Algorithm Dec(c, sk)
$(sk_1, sk_2) \leftarrow sk$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow$ ML-KEM-768.$\mathsf{Dec}(c_1, sk_1)$
$k_2 \leftarrow$ X25519.$\mathsf{DH}(sk_2, c_2)$
$s \leftarrow$ "\.//^\"$\|k_1\|k_2\|c_2\|pk_2$
$k \leftarrow$ SHA3-256$(s)$
return $k$
```

Figure 12: X-Wing KEM. $\mathsf{random}(N)$ generates N random bytes. X25519.DH is the byte-oriented function X25519 defined in section 5 of RFC 7748. g_{X25519} is the X25519 base point where $u = 9$ on curve25519 defined in section 6.1 of RFC 7748. ML-KEM-768.KeyGen, ML-KEM-768.Enc, and ML-KEM-768.Dec are defined in FIPS 203. SHA3-256 is defined in FIPS 202. The X-Wing label is inlined as the first 6 ASCII-encoded bytes of the SHA3-256 input.

7.3 Definition of X-Wing

Definition 8 (X-Wing KEM)**.** Given the ML-KEM-768 KEM, the X25519 ECDH KEA, and the SHA3-256 hash, the X-Wing KEM is defined as a tuple of algorithms $\{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$, which are in turn defined in Figure 12.

Acknowledgements

This research was supported by Deutsche Forschungsgemeinschaft (DFG, German research Foundation) as part of the Excellence Strategy of the German Federal and State Governments – EXC 2092 CASA - 390781972; and by the European Commission through the ERC Starting Grant 805031 (EPOQUE); and by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project UIDP/50014/2020.

References

[ABH+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE standard. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 87–116, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-77870-5_4.

[AP05] Michel Abdalla and David Pointcheval. Interactive Diffie-Hellman assumptions with applications to password-based authentication. In Andrew Patrick and Moti Yung, editors, *FC 2005: 9th International Conference on Financial Cryptography and Data Security*, volume 3570 of *Lecture Notes in Computer Science*, pages 341–356, Roseau, The Commonwealth Of Dominica, February 28 – March 3, 2005. Springer, Heidelberg, Germany.

[BBLW22] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180, February 2022. URL: https://www.rfc-editor.org/info/RFC9180, doi:10.17487/RFC9180.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany. doi:10.1007/11745853_14.

[CDM23] Cas Cremers, Alexander Dax, and Niklas Medinger. Keeping up with the kems: Stronger security notions for kems. Cryptology ePrint Archive, Paper 2023/1933, 2023. https://eprint.iacr.org/2023/1933. URL: https://eprint.iacr.org/2023/1933.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 190–218, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-76578-5_7.

[LHT16] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic Curves for Security. RFC 7748, January 2016. URL: https://www.rfc-editor.org/info/rfc7748, doi:10.17487/RFC7748.

[NIS23] NIST. Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS 203 (Initial Public Draft), August 2023. doi:10.6028/NIST.FIPS.203.ipd.

[O'B23] Devon O'Brien. Protecting chrome traffic with hybrid kyber kem. Chromium Blog, 2023. https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html.

[OWK23] Mike Ounsworth, Aron Wussler, and Stavros Kousidis. Combiner function for hybrid key encapsulation mechanisms (Hybrid KEMs). Internet-Draft draft-ounsworth-cfrg-kem-combiners-04, Internet Engineering Task Force, July 2023. Work in Progress. URL: https://datatracker.ietf.org/doc/draft-ounsworth-cfrg-kem-combiners/04/.

[WR22] Bas Westerbaan and Cefan Rubin. Defending against future threats: Cloudflare goes post-quantum. The Cloudflare Blog, 2022. https://blog.cloudflare.com/post-quantum-for-all/.