StarFortress

Hybrid KEMs with Diffie-Hellman Inlining

Deirdre Connolly[1,2] and Paul Grubbs[3]

[1] Oracle, USA
[2] Selkie Cryptography, USA
[3] University of Michigan, USA

Abstract. This short paper formally specifies and analyzes the UG hybrid KEM construction from the IRTF CFRG's recent draft on hybrid (post-quantum/traditional) KEMs. The UG construction is an optimized hybrid of a Diffie-Hellman (DH)-based KEM in a nominal group and a generic IND-CCA KEM. The main optimization is that the group elements derived in the DH-based KEM are "inlined" in the key derivation, saving unnecessary hashing. We perform two security analyses of the UG construction: one shows UG is IND-CCA even if the generic IND-CCA KEM is broken; the other complementary analysis shows UG is IND-CCA even if the DH assumptions in the nominal group are broken (by, e.g., a cryptographically-relevant quantum computer).

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

1 Introduction

The transition to post-quantum key agreement is underway. NIST has standardized ML-KEM, other state-level bodies are providing their own cryptographic recommendations, and organizations are beginning to update to to protect against future quantum adversaries. However, these algorithms are relatively new, and there remains a possibility of unexpected cryptanalytic breakthroughs or vulnerable implementations [CD23]. A natural hedge against this risk is to deploy hybrid constructions that combine post-quantum and classical algorithms, maintaining security as long as at least one component remains secure.

This "defense in depth" approach is appealing: if the post-quantum KEM is broken by classical cryptanalysis or a software bug, the classical component provides classical security; if the classical component is broken by a quantum computer, the post-quantum component provides quantum resistance. However, constructing provably secure hybrid KEMs requires care. The combiner function must preserve security properties under the right assumptions, and different designs involve different trade-offs between the cryptographic assumptions of the components and efficiency of the whole scheme.

The Internet Research Task Force (IRTF)'s Crypto Forum Research Group (CFRG) is working towards finalizing a recommendation for specific constructions of hybrid KEMs. The current CFRG document [CBG25] gives four constructions named CG, CK, UG, and UK. The CG and UK constructions are, roughly speaking, covered by existing analyses of similar schemes. These analyses do not apply to CK and UG. The goal of this paper is to close this gap and provide a clean analysis of UG; a subset of this paper's authors are also working in parallel on a similar paper for CK.

E-mail: durumcrustulum@gmail.com (Deirdre Connolly), paulgrub@umich.edu (Paul Grubbs)

Our Contributions. Our first contribution is to formally specify the UG construction (Section 3). At a high level, UG (hereafter UG) combines a post-quantum KEM with a classically-secure nominal group. Like other hybrids, UG concatenates shared secrets, ciphertexts, and encapsulation keys from both components, then applies a key derivation function (KDF) to produce the final shared secret. Its design is overall conservative—UG includes all public values in the KDF input. The main departure from other hybrid constructions is that the group elements derived during the DH key computation are put into the KDF directly instead of being hashed once and then having the result hashed again in the final KDF. This breaks an abstraction boundary and renders prior security analyses inapplicable, but saves redundant hashing and eases implementation with existing libraries.

The next contribution (Section 4) is a clean, modular analysis of the IND-CCA security of UG. Like other hybrid KEMs, we prove two complementary results corresponding to two different possible worlds. In Section 4.1 we show that even if the KEM component is broken, UG remains secure (modelling the KDF as a random oracle) as long as the strong Diffie-Hellman problem in the nominal group holds. Our proof roughly follows the analogous result from X-Wing [CSW24], though because UG hashes the KEM ciphertext we need not reduce to ciphertext preimage resistance (C2PRI) [BCD+24].

In Section 4.2 we show that IND-CCA security of UG also holds when reducing *only* to the IND-CCA of the component KEM and the PRF security of the KDF. This proof largely follows the template for proving CCA of KEM combiners given in [GHP18]. Notably, this proof does not require modelling the KDF as a random oracle. We end the security analyses with a brief discussion of other security goals, like key-binding [CDM24].

We conclude the paper in Section 5 by describing some concrete choices of nominal group and KEM that can be used to instantiate UG. Since UG requires fairly minimal assumptions from its components, it can be instantiated with diverse combinations of elliptic curves and KEMs.

Overall, our work complements and supports ongoing work on standardizing hybrid KEM designs that strike desirable tradeoffs between performance and security. UG represents a conservative point in the design space: it achieves strong security guarantees with minimal assumptions, at the cost of hashing additional public data, while eliminating layers of abstraction.

2 Definitions

2.1 Key encapsulation mechanisms

2.1.1 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.$\mathsf{KeyGen}(\,) \xrightarrow{\$} (sk, pk)$ Randomized algorithm that outputs a secret (private) key $sk \in \mathcal{SK}$, and a public key $pk \in \mathcal{PK}$.
- KEM.$\mathsf{Enc}(pk) \xrightarrow{\$} (k, c)$ 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.$\mathsf{Dec}(c, sk) \rightarrow y \in \{k, \perp\}$ 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 $\perp$.

2.1.2 Correctness

The correctness of a KEM imposes that, except with small probability drawn over the random 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:

$$\Pr\left[k \neq k' \;\middle|\; \begin{smallmatrix}(sk,pk)\leftarrow\$\mathsf{KeyGen}(\,)\\(k,c)\leftarrow\$\mathsf{Enc}(pk)\\k'\leftarrow\mathsf{Dec}(c,sk)\end{smallmatrix}\right] \leq \delta.$$

2.1.3 Security

The IND-CCA security game for KEMs is denoted as $\text{IND-CCA}^b_{\text{KEM},\mathcal{A}}$ and is shown in Figure 1.

Game $\text{IND-CCA}^b_{\text{KEM},\mathcal{A}}$

1: $(sk, pk) \leftarrow\$ \text{KEM.KeyGen}(\,)$
2: $(k_0, c^*) \leftarrow\$ \text{KEM.Enc}(pk)$
3: $k_1 \leftarrow\$ \mathcal{K}$
4: $b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k_b)$
5: **return** b'

Oracle $\mathsf{Dec}(c)$

1: **if** $c = c^*$ **then**
2: **return** $\perp$
3: $k \leftarrow \text{KEM.Dec}(c, sk)$
4: **return** k

Figure 1: IND-CCA security game for KEMs.

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

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

A KEM is IND-CCA secure for if for any PPT adversary $\mathcal{A}$, $\text{Adv}^{\text{KEM}}_{\text{IND-CCA},\mathcal{A}}$ is negligible.

2.2 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 2 as

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

Game $\text{PRF}^b_{f,\mathcal{A}}$

1: $\mathbf{T}[\cdot] \leftarrow \perp$
2: $k \leftarrow\$ \mathcal{K}$
3: $b' \leftarrow\$ \mathcal{A}^{\mathsf{Eval}(\cdot)}$
4: **return** b'

Oracle $\mathsf{Eval}_f(x)$

1: **if** $x \in \mathbf{T}$ **then**
2: **return** $\mathbf{T}[x]$
3: $y_0 \leftarrow f(k, x)$; $y_1 \leftarrow\$ \mathcal{Y}$
4: $\mathbf{T}[x] \leftarrow y_b$
5: **return** y_b

Figure 2: PRF security games.

2.3 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 recognizable 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 $\varepsilon_u \subset \mathbb{Z}/p\mathbb{Z}$, 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: $\Delta_{\mathcal{N}} = \Delta[D_H, D_U] = \frac{1}{2} \sum_{x \in \mathbb{Z}} |\Pr_{D_U}(x) - \Pr_{D_H}(x)|$ [ABH+21].

2.4 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

$$\mathrm{Adv}^{\mathrm{SDH}}_{\mathcal{N},\mathcal{A}} = \Pr\left[Z = g^{xy} \,\middle|\, x, y \leftarrow \varepsilon_u; Z \leftarrow \mathcal{A}^{DH(\cdot,\cdot)}(g^x, g^y)\right].$$

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.

3 The UG Construction

Here we describe the UG hybrid KEM construction.

Intuition for the design. UG opts for a conservative construction while providing ease of implementation given a common set of crypto tools: a KDF, an elliptic curve group, and a post-quantum KEM. Relying directly on the SDH security of the group allows us to inline the DH-KEM construction and avoid extra layers of hashing, which is necessary elsewhere to achieve an IND-CCA KEM from a group and then use that classical KEM as component of the hybrid scheme. As group-based KEMs are less common in crypto libraries than plain elliptic curve groups, this allows easier implementation and deployment.

Including all ciphertexts of the component schemes lets us achieve IND-CCA security [GHP18] in the pre- and post- quantum settings without relying on C2PRI or other analysis of the PQ KEM component. By including all the public key material of the component schemes, we mitigate multi-target/multi-user attacks [GHS25], and ensure "binding" [CDM24] to the public data pertaining to the computation of the hybrid shared secret. (We discuss binding in more detail in Section 4.3.)

Definition 6 (UG)**.** 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 UG KEM as depicted in fig. 3.

Algorithm $\mathsf{KeyGen}()$

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

Algorithm $\mathsf{Enc}(pk)$

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

Algorithm $\mathsf{Dec}(c, sk)$

$(sk_1, sk_2, pk) \leftarrow sk$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ **then**
  return $\perp$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| \text{label})$
return k

Figure 3: UG Framework.

3.1 Correctness of UG

The correctness of UG follows from the correctness of the KEM used to instantiate UG. Formally, if UG is instantiated with a δ-correct KEM, then UG is also δ-correct.

4 Security of UG

The objective of a hybrid of this form is that the hybrid remains secure so long as one of the component algorithms remains secure. In other words, one algorithm may be found to contain weaknesses without compromising the hybrid. The UG construction is similar to a generic composition of the classic DH-based KEM in the nominal group and the IND-CCA KEM. Thus, the reader may ask why the proof of GHP [GHP18] does not apply. The basic reason this is true is because UG is not a fully generic composition of two IND-CCA KEMs: the traditional DH-based KEM is not IND-CCA as it would be used in implementations because of x-coordinate truncation: namely, the shared secret output from the KEM would actually just be the x-coordinate of the group element $k_2 := \mathsf{exp}(pk_2, sk_e)$.

Proof-strategy. Our proofs cover multiple scenarios, including the possibility of cryptographically relevant quantum computers becoming a reality, and thus breaking the IND-CCA security of pre-quantum KEMs, and the possibility of post-quantum cryptography such as a KEM being classically broken, such as by an attack, vulnerability, or implementation error.

Pre-Quantum Security. This the case where one of the component KEMs is assumed to be classically IND-CCA-secure but where the post-quantum KEM may not offer any IND-CCA security. We prove in the standard model that UG is IND-CCA-secure when:

1. The KDF is modeled as a secure PRF;
2. The pre-quantum component is SDH-secure; and

3. The post-quantum KEM is modeled as a possibly broken KEM that provides no security.

Our proof also demonstrates the opposite scenario:

Post-Quantum Security: This the case where the component KEMs is assumed to be IND-CCA-secure against a quantum adversary but the pre-quantum component group may not offer any security at all. We prove in the standard model that UG is IND-CCA-secure when:

1. The KDF is modeled as a PRF;

2. The pre-quantum component group no longer provides any security; and

3. The post-quantum KEM provides IND-CCA security against a quantum attacker.

These results do not rely on the ROM nor the QROM: it is sufficient to assume the standard model PRF security property for the employed hash function F.

4.1 Reduction to SDH security in the ROM

Theorem 1. *Let $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$ be a nominal group and KEM be a 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 UG in Definition 6 making at most q_h queries to the random oracles. Then there exists an adversary $\mathcal{B}$ such that,*

$$Adv^{IND\text{-}CCA}_{UG,\mathcal{A}} \leq 4\Delta_{\mathcal{N}} + 2Adv^{SDH}_{\mathcal{N},\mathcal{B}}$$

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

Proof. We consider the following games:

Game 0 We start with the standard IND-CCA security game for UG as defined in Figure 4. In this game, Σ is used to model $H(m)$ as lazily sampled random oracle. As G_0 is the IND-CCA^0 game instantiated with UG, by definition,

$$\Pr[\text{IND-CCA}^0_{\text{UG},\mathcal{A}} \Rightarrow 1] = \Pr[G^{\mathcal{A}}_0 \Rightarrow 1].$$

Game G_0

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

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\bot$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \bot$ **then**
  return $\bot$
$s \leftarrow k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label$
if $\sum[s] = \bot$ **then**
  $\sum[s] \leftarrow\$\ \{0,1\}^n$
$k \leftarrow \sum[s]$
return k

Oracle $H(m)$

$k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label \leftarrow m$
if $\sum[m] = \bot$ **then**
  $\sum[m] \leftarrow\$\ \{0,1\}^n$
return $\sum[m]$

Figure 4: Game G_0.

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.

Game G_1

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

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\bot$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \bot$ **then**
  return $\bot$
$s \leftarrow k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label$
if $\sum[s] = \bot$ **then**
  $\sum[s] \leftarrow\$\ \{0,1\}^n$
$k \leftarrow \sum[s]$
return k

Oracle $H(m)$

$k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label \leftarrow m$
if $\sum[m] = \bot$ **then**
  $\sum[m] \leftarrow\$\ \{0,1\}^n$
return $\sum[m]$

Figure 5: Game G_1.

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_2

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

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\bot$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \bot$ **then**
  return $\bot$
$s \leftarrow k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label$
if $\sum[s] = \bot$ **then**
  $\sum[s] \leftarrow\!\!\$\ \{0,1\}^n$
$k \leftarrow \sum[s]$
return k

Oracle $H(m)$

$k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label \leftarrow m$
if $k_2 = k_2^*$ **then**
  $abort_1$
if $\sum[m] = \bot$ **then**
  $\sum[m] \leftarrow\!\!\$\ \{0,1\}^n$
return $\sum[m]$

Figure 6: Game G_2.

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 containing the k_2^* used to generate k^*. Consider adversary $\mathcal{B}$ from Figure 7 playing the SDH game and simulating $\mathcal{A}$'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 $H(k_1^* \| \mathsf{exp}(Y, x) \| c_1^* \| c_2^* \| pk_1 \| pk_2 \| label)$. $\mathcal{B}$ is unable to 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 with $\mathsf{exp}(Y, x)$. $\mathcal{B}$ can use the DH oracle to detect when $\mathcal{A}$ queries H 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 $k_1 \| \mathsf{exp}(c_2, x) \| c_1 \| c_2 \| pk_1 \| pk_2 \| label$ 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 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.

Adversary $\mathcal{B}^{\mathsf{DH}(\cdot,\cdot)}(X, Y)$

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

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\bot$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(c_1, sk_1)$
if $k_1 = \bot$ **then**
  return $\bot$
if $\mathbf{E}[(k_1, \star, c_1, c_2, pk_1, pk_2, label)] = \bot$ **then**
  $\mathbf{E}[(k_1, \star, c_1, c_2, pk_1, pk_2, label)] \leftarrow\$ \{0,1\}^n$
return $\mathbf{E}[(k_1, \star, c_1, c_2, pk_1, pk_2, label)]$

Oracle $H(m)$

$k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label \leftarrow m$
if $\mathsf{DH}(Y, k_2) = 1$ **then**
  output k_2
if $\mathsf{DH}(c_2, k_2) = 1$ **then**
  if $\mathbf{E}[m] = \bot$ **then**
    $\mathbf{E}[m] \leftarrow\$ \{0,1\}^n$
  return $\mathbf{E}[m]$
if $\sum[m] = \bot$ **then**
  $\sum[m] \leftarrow\$ \{0,1\}^n$
return $\sum[m]$

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

Game 3 In G_3 we replace the shared challenge key k^* with a uniform random key. Therefore,

Claim 3

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[G_3^{\mathcal{A}} \Rightarrow 1].$$

Proof To prove that this change is undetectable by $\mathcal{A}$, we need to argue that $\mathcal{A}$'s view when interacting with the various oracles is independent of the value of k^*. 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 k^* from the challenge. Since the adversary $\mathcal{A}$ cannot query the random oracle H with the input used to generate k^*, the output of the random oracle H is independent of k^*. The abort condition in Dec also prevents the adversary from querying the Dec oracle in a way that would result in k^* being outputted. Either c_1 or c_2 must be different from c_1^* or c_2^*. If c_1 is different, then the resulting key k_1 must be different from k_1^*, otherwise the abort condition would be triggered. If c_2 is different, then the input to the random oracle

must also be different, since it is included in its input. Therefore, the behavior of the Dec oracle is independent of the value of k^*. Since the behavior of all oracles is independent of k^*, the adversary cannot detect this game hop.

Game G_3

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

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\bot$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \bot$ **then**
  return $\bot$
$s \leftarrow k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label$
if $\sum[s] = \bot$ **then**
  $\sum[s] \leftarrow\!\!\$\ \{0,1\}^n$
$k \leftarrow \sum[s]$
return k

Oracle $H(m)$

$k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label \leftarrow m$
if $k_2 = k_2^*$ **then**
  $abort_1$
if $\sum[m] = \bot$ **then**
  $\sum[m] \leftarrow\!\!\$\ \{0,1\}^n$
return $\sum[m]$

Figure 8: Game G_3.

Game 4 Finally, we revert the changes introduced in G_1 and G_2. This makes the oracles identical to Game 0, and the only remaining change in the main experiment is that k^* is random. Undoing the hops accounts for the extra factors in the advantage.

Game G_4

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

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\bot$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \bot$ **then**
  return $\bot$
$s \leftarrow k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label$
if $\sum[s] = \bot$ **then**
  $\sum[s] \leftarrow\!\!\$\ \{0,1\}^n$
$k \leftarrow \sum[s]$
return k

Oracle $H(m)$

$k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label \leftarrow m$
if $k_2 = k_2^*$ **then**
  $abort_1$
if $\sum[m] = \bot$ **then**
  $\sum[m] \leftarrow\!\!\$\ \{0,1\}^n$
return $\sum[m]$

Figure 9: Game G_4.

Claim 4

$$|\Pr[G_3^{\mathcal{A}} \Rightarrow 1] - \Pr[G_4^{\mathcal{A}} \Rightarrow 1]| \leq 2\Delta_{\mathcal{N}} + \text{Adv}^{\text{SDH}}_{\mathcal{N},\mathcal{B}}.$$

Proof As in G_1, we change the distribution from which the two group exponents are chosen back to the honest distribution. This incurs $2\Delta_{\mathcal{N}}$ as an upper bound for $\mathcal{A}$ to detect this change. Undoing the change in G_2, we remove the abort in H so that a DH oracle doesn't help anymore, incurring a similar advantage difference.

Now k^* is chosen uniformly at random, which means that G_4 matches the definition of the $\text{IND-CCA}^1_{\text{UG},\mathcal{A}}$ game perfectly. Therefore,

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

This concludes the proof of Theorem 1.

□

4.2 Reduction to IND-CCA security of KEM

Theorem 2. *Let $\mathcal{N} = (\mathcal{G}, g, p, \varepsilon_h, \varepsilon_u, \mathsf{exp})$ be a nominal group and KEM be a key encapsulation mechanism, let H be a secure PRF with an output size of n when keyed on k_1, and let $\mathcal{A}$ be an adversary against the IND-CCA security of UG in Definition 6. Then, there exist $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{E}$, such that,*

$$Adv^{UG}_{IND\text{-}CCA,\mathcal{A}} \leq Adv^{KEM}_{IND\text{-}CCA,\mathcal{B}} + Adv^{KEM}_{IND\text{-}CCA,\mathcal{C}} + Adv^{H}_{PRF,\mathcal{D}} + Adv^{H}_{PRF,\mathcal{E}} + 2\delta\,,$$

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

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

Game 0 We start with the standard IND-CCA security game for UG. Thus,

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

Game G_0

$sk_1, pk_1 \leftarrow\$ \text{KEM.KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
$k_1^*, c_1^* \leftarrow\$ \text{KEM.Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k^* \leftarrow H(k_1^* \| k_2^* \| c_1^* \| c_2^* \| pk_1 \| pk_2 \| label)$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return b'

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ **then**
  return $\perp$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 10: Game G_0.

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

Game G_1

$sk_1, pk_1 \leftarrow\$ \text{KEM.KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
~~$k_1^*, c_1^* \leftarrow\$ \text{KEM.Enc}(pk_1)$~~
$_, c_1^* \leftarrow\$ \text{KEM.Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k_1^* \leftarrow\$ \mathcal{K}$
$k^* \leftarrow H(k_1^* \| k_2^* \| c_1^* \| c_2^* \| pk_1 \| pk_2 \| label)$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return b'

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ **then**
  return $\perp$
if $c_1 = c_1^*$ **then**
  $k_1 \leftarrow k_1^*$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 11: Game G_1.

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

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

Proof We construct $\mathcal{B}$ as in Figure 12. The adversary $\mathcal{B}$ simulates the environment of $\mathcal{A}$ by calculating the nominal group components of UG itself and embedding the KEM challenge into the UG challenge. The decapsulation oracle is simulated by $\mathcal{B}$ 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 $\mathcal{B}$ correctly interpolates between games G_0 and G_1, except when c_1^* would decapsulate to something other than k_1^* in G_0 (this could happen if, e.g., decapsulation failed). We can bound the probability of this inconsistency using the correctness of KEM (defined in Section 2.1.2), which adds a factor of δ. More precisely, we have

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

and

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

which means that

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

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

Adversary $\mathcal{B}^{\mathsf{Dec}_O(\cdot)}(pk_1, k_1^*, c_1^*)$

$sk_2 \leftarrow\!\!\$\ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk_e \leftarrow\!\!\$\ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k^* \leftarrow H(k_1^* \| k_2^* \| c_1^* \| c_2^* \| pk_1 \| pk_2 \| label)$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
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 $\perp$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 12: Adversary $\mathcal{B}$ 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 $\mathcal{A}$.

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

Game G_2

$sk_1, pk_1 \leftarrow\$ \text{KEM.KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
$_, c_1^* \leftarrow\$ \text{KEM.Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k_1^* \leftarrow\$ \mathcal{K}$
$k^* \leftarrow H(k_1^* \| k_2^* \| c_1^* \| c_2^* \| pk_1 \| pk_2 \| label)$
$k^* \leftarrow\$ \{0,1\}^n$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return b'

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ **then**
  return $\perp$
if $c_1 = c_1^*$ **then**
  $k_1 \leftarrow k_1^*$
  $k \leftarrow\$ \{0,1\}^n$
  return k
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 13: Game G_2.

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

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

Proof This is a simple reduction shown in Figure 14. This adversary can generate all cryptographic parameters except k_1^*, for which it uses its PRF oracle Eval_H. As we can see, when the challenge PRF key is real, then $\mathcal{D}$ will be using a real output of the pseudorandom function and hence running G_1; whereas if the challenge key is random, $\mathcal{D}$ will be using a uniformly sampled output and running G_2. Hence,

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

and

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[G^1_{H,\mathcal{D}} \Rightarrow 1],$$

which means that

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

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

Adversary $\mathcal{D}^{\mathsf{Eval}_H(\cdot)}()$

$sk_1, pk_1 \leftarrow\$ \text{KEM.}\mathsf{KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2, pk_2)$
$k_1^*, c_1^* \leftarrow\$ \text{KEM.}\mathsf{Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k^* \leftarrow \mathsf{Eval}_H(k_2^* \| c_1^* \| c_2^* \| pk_1 \| pk_2 \| label)$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
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**
  $k \leftarrow \mathsf{Eval}_H(k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
  return k
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(sk_1, c_1)$
if $k_1 = \perp$ **then**
  return $\perp$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 14: Adversary $\mathcal{D}$ against the PRF security of H.

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

Game G_3

$sk_1, pk_1 \leftarrow\$ \text{KEM.}\mathsf{KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
$_, c_1^* \leftarrow\$ \text{KEM.}\mathsf{Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k_1^* \leftarrow\$ \mathcal{K}$
$k^* \leftarrow\$ \{0,1\}^n$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return b'

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.}\mathsf{Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ **then**
  return $\perp$
if $c_1 = c_1^*$ **then**
  $k \leftarrow\$ \{0,1\}^n$
  return k
  $k_1 \leftarrow k_1^*$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 15: Game G_3.

Claim 3 We justify this hop with a reduction to the PRF property of H very similar to the previous one, again using the PRF oracle Eval_H keyed with k_1^*.

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

Proof We construct $\mathcal{E}$ as in Figure 16, and the analysis is similar to the previous hop.

$$\Pr[G_2^{\mathcal{A}} \Rightarrow 1] = \Pr[\text{PRF}^0_{H,\mathcal{E}} \Rightarrow 1],$$

and

$$\Pr[G_3^{\mathcal{A}} \Rightarrow 1] = \Pr[\text{PRF}^1_{H,\mathcal{E}} \Rightarrow 1],$$

which means that

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

and $\mathcal{E}$ will perform at most one evaluation query per decapsulation query by $\mathcal{A}$ as it will never call the evaluation oracle to compute the challenge ciphertext.

Adversary $\mathcal{E}^{\mathsf{Eval}_H(\cdot)}()$

1: $sk_1, pk_1 \leftarrow\$ \mathrm{KEM.KeyGen}()$
2: $sk_2 \leftarrow\$ \varepsilon_h$
3: $pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
4: $pk \leftarrow (pk_1, pk_2)$
5: $sk \leftarrow (sk_1, sk_2, pk_2)$
6: $k_1, c_1^* \leftarrow\$ \mathrm{KEM.Enc}(pk_1)$
7: $sk_e \leftarrow\$ \varepsilon_u$
8: $c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
9: $k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
10: $k^* \leftarrow\$ \{0,1\}^n$
11: $c^* \leftarrow (c_1^*, c_2^*)$
12: $b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
13: **return** b'

Oracle $\mathsf{Dec}(c)$

1: **if** $c = c^*$ **then**
2: **return** $\perp$
3: $(c_1, c_2) \leftarrow c$
4: $k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
5: **if** $c_1 = c_1^*$ **then**
6: $k \leftarrow \mathsf{Eval}_H(k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| \mathit{label})$
7: **return** k
8: $k_1 \leftarrow \mathrm{KEM.Dec}(sk_1, c_1)$
9: **if** $k_1 = \perp$ **then**
10: **return** $\perp$
11: $k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| \mathit{label})$
12: **return** k

Figure 16: Adversary $\mathcal{E}$ against the PRF security of H.

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. Note that the only change between Game 0 and Game 4 is that k^* is random.

Claim 4 This hop is justified in a way that is similar to the jump to Game 1. We introduce adversary $\mathcal{C}$ against the IND-CCA security of KEM, with roughly the same running time as $\mathcal{A}$, in Figure 18. We claim that,

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

Proof Again, the reduction to the IND-CCA security of KEM is perfect, except if c_1^* would not decapsulate to k_1^* in Game 4. The event of c_1^* not decapsulating to k_1^* in Game 4 can be bound by the correctness of KEM, and the claim follows.

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

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

Game G_4

$sk_1, pk_1 \leftarrow\$ \text{KEM.KeyGen}()$
$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk \leftarrow (sk_1, sk_2)$
$_, c_1^* \leftarrow\$ \text{KEM.Enc}(pk_1)$
$k_1^*, c_1^* \leftarrow\$ \text{KEM.Enc}(pk_1)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k_1^* \leftarrow\$ \mathcal{K}$
$k^* \leftarrow\$ \{0,1\}^n$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
return b'

Oracle $\mathsf{Dec}(c)$

if $c = c^*$ **then**
  return $\perp$
$(c_1, c_2) \leftarrow c$
$k_1 \leftarrow \text{KEM.Dec}(c_1, sk_1)$
$k_2 \leftarrow \mathsf{exp}(c_2, sk_2)$
if $k_1 = \perp$ **then**
  return $\perp$
if $c_1 = c_1^*$ **then**
  $k_1 \leftarrow k_1^*$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 17: Game G_4.

Adversary $\mathcal{C}^{\mathsf{Dec}_O(\cdot)}(pk_1, k_1^*, c_1^*)$

$sk_2 \leftarrow\$ \varepsilon_h$
$pk_2 \leftarrow \mathsf{exp}(g, sk_2)$
$pk \leftarrow (pk_1, pk_2)$
$sk_e \leftarrow\$ \varepsilon_u$
$c_2^* \leftarrow \mathsf{exp}(g, sk_e)$
$k_2^* \leftarrow \mathsf{exp}(pk_2, sk_e)$
$k^* \leftarrow\$ \{0,1\}^n$
$c^* \leftarrow (c_1^*, c_2^*)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{Dec}(\cdot)}(pk, c^*, k^*)$
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 $\perp$
$k \leftarrow H(k_1 \| k_2 \| c_1 \| c_2 \| pk_1 \| pk_2 \| label)$
return k

Figure 18: Adversary $\mathcal{C}$ against the IND-CCA security of KEM. Note that $\mathcal{C}$ makes at most q_d queries to Dec_O, one for each decapsulation query placed by $\mathcal{A}$.

Game 4 is the ending game. We can conclude that Theorem 2 follows from collecting all the terms in the claims.

□

4.3 Other Security Goals

The CFRG draft identifies two additional "binding" properties that a hybrid KEM construction must have: LEAK-BIND-K-PK and LEAK-BIND-K-CT. Roughly, these properties imply that even given a leaked secret key it is difficult for an adversary to equivocate public keys or ciphertexts for an honestly-generated output key. We will not formally prove these properties hold for UG; however, we remark that the CFRG draft's proofs of these properties for UK go through basically unmodified for UG.

5 Guidance for Concrete Instantiations

The UG construction places minimal requirements on its component algorithms, allowing for diverse instantiations. Here we discuss some possible concrete choices that satisfy the security assumptions of our theorems, especially when targeting approximately 128-bit security.

Nominal Groups. For the nominal group $\mathcal{N}$, both X25519 [LHT16] and the NIST P-256 curve [oST23] are suitable choices. It is generally held that both curves achieve SDH under standard assumptions, with the best known classical attacks being generic discrete logarithm algorithms. The nominal group abstraction from [ABH+21] captures both Montgomery curves like Curve25519 and short Weierstrass curves like P-256.

For X25519, the set of honest exponents ε_h consists of 32-byte strings with the standard clamping applied, while $\varepsilon_u = \mathbb{Z}/p\mathbb{Z}$ for the curve's prime order subgroup. The statistical distance $\Delta_{\mathcal{N}}$ between honestly-generated and uniform exponents is negligible ($< 2^{-128}$), as analyzed in [ABH+21, BCD+24]. Similar reasoning applies to P-256 when using rejection sampling or other standard techniques to generate uniform scalars: the two distributions D_H and D_U are identical, thus the $\Delta_{\mathcal{N}}$ between honestly-generated and uniform exponents is 0, and all elements have the same probability.

Post-Quantum KEMs. ML-KEM-768 [NIS23] is a popular choice for the IND-CCA KEM component. NIST's standardization process included extensive analysis of ML-KEM's IND-CCA security, reducing IND-CCA security to the hardness of Module-LWE [BDK+18]. ML-KEM-768 targets NIST Security Level 3, which should be approximately as difficult and expensive to attack as running a key search on a 192-bit block cipher. The level 3 parameter set has been paired with 128-bit curves in other hybrid constructions such as X-Wing to allow for some cushion and hedging against improvements in lattice cryptanalysis in the long term. The correctness error δ is bounded by 2^{-164}, which is negligible in the security parameter.

Key Derivation Functions. For the hash function H used as the KDF, both SHA3-256 (or SHAKE256, its XOF form) and HKDF-SHA256 are appropriate choices.

SHA-3/SHAKE256. The Keccak sponge construction underlying SHA-3 [Dwo15] provides a clean instantiation when modeling H as a random oracle. When keyed, SHAKE256 also achieves PRF security under the assumption that the Keccak permutation behaves as an ideal permutation [BDPV08]. Sponge-based constructions like SHA-3 have been shown to be indifferentiable against classical [BDPV08] as well as quantum adversaries [ACMT25].

HKDF-SHA256. In the standard model, HKDF [Kra10] instantiated with HMAC-SHA256 provides PRF security under the assumption that HMAC-SHA256 is a secure PRF when keyed with a high-entropy key. Krawczyk's analysis [Kra10] establishes that HKDF-Expand is a PRF when the input key material has sufficient min-entropy, which is guaranteed in UG since k_1 is either an honestly-generated KEM shared secret or uniform random. HKDF has been shown to be indifferentiable from a random oracle under specific constraints [LBB19]: that HMAC is indifferentiable from a random oracle, which for HMAC-SHA-256 has been shown in [DRST12], assuming the compression function underlying SHA-256 is a random oracle, which it is indifferentiably when used prefix-free, and the values of HKDF's `IKM` input do not collide with values of `info || 0x01`.

The choice of the KDF H's security level ought to be made based on the security level provided by the constituent KEM.

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, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 87–116. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_4. 4, 18

[ACMT25] Gorjan Alagic, Joseph Carolan, Christian Majenz, and Saliha Tokat. The Sponge is quantum indifferentiable. Cryptology ePrint Archive, Report 2025/731, 2025. URL: https://eprint.iacr.org/2025/731. 18

[BCD+24] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. X-wing. *CiC*, 1(1):21, 2024. doi:10.62056/a3qj89n4e. 2, 18

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy*, pages 353–367. IEEE Computer Society Press, April 2018. doi:10.1109/EuroSP.2018.00032. 18

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Berlin, Heidelberg, April 2008. doi:10.1007/978-3-540-78967-3_11. 18

[CBG25] Deirdre Connolly, Richard Barnes, and Paul Grubbs. Hybrid PQ/T key encapsulation mechanisms. Technical report, Internet Engineering Task Force, October 2025. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-hybrid-kems/07/. 1

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_15. 1

[CDM24] Cas Cremers, Alexander Dax, and Niklas Medinger. Keeping up with the KEMs: Stronger security notions for KEMs and automated analysis of KEM-based protocols. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1046–1060. ACM Press, October 2024. doi:10.1145/3658644.3670283. 2, 4

[CSW24] Deirdre Connolly, Peter Schwabe, and Bas Westerbaan. X-Wing: general-purpose hybrid post-quantum KEM. draft-connolly-cfrg-xwing-kem, 2024. URL: https://datatracker.ietf.org/doc/draft-connolly-cfrg-xwing-kem/. 2

[DRST12] Yevgeniy Dodis, Thomas Ristenpart, John P. Steinberger, and Stefano Tessaro. To hash or not to hash again? (In)differentiability results for H^2 and HMAC. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 348–366. Springer, Berlin, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_21. 18

[Dwo15] Morris J. Dworkin. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. National Institute of Standards and Technology (NIST),

July 2015. URL: http://dx.doi.org/10.6028/NIST.FIPS.202, doi:10.6028/nist.fips.202. 18

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018. doi:10.1007/978-3-319-76578-5_7. 2, 4, 5, 11

[GHS25] Lewis Glabush, Kathrin Hövelmanns, and Douglas Stebila. Tight multi-challenge security reductions for key encapsulation mechanisms. Cryptology ePrint Archive, Report 2025/343, 2025. URL: https://eprint.iacr.org/2025/343. 4

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_34. 18

[LBB19] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. A mechanised cryptographic proof of the WireGuard virtual private network protocol. In *2019 IEEE European Symposium on Security and Privacy*, pages 231–246. IEEE Computer Society Press, June 2019. doi:10.1109/EuroSP.2019.00026. 18

[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. 18

[NIS23] NIST. Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS 203 (Initial Public Draft), August 2023. doi:10.6028/NIST.FIPS.203.ipd. 18

[oST23] National Institute of Standards and Technology. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. Technical Report NIST SP 800-186, U.S. Department of Commerce, February 2023. URL: https://doi.org/10.6028/NIST.SP.800-186. 18