Leaking-cascades: an optimized construction for KEM hybridization

Céline Chevalier[1,2], Guirec Lebrun[1,3], and Ange Martinelli[3]

[1] DIENS, École normale supérieure, CNRS, PSL University, Inria, Paris, France
[2] CRED, Paris-Panthéon-Assas University
[3] ANSSI, Paris, France

Abstract. Hybrid post-quantum cryptography is a cautious approach that aims to guard against the threat posed by the quantum computer, through the simultaneous use of Post-Quantum (PQ) and classical (i.e. pre-quantum) cryptosystems, in case the post-quantum schemes used would turn out to be insecure.
Regarding the hybridization of Key Encapsulation Mechanisms (KEMs), most recent studies focus on how to safely combine the symmetric keys output by a parallel execution of classical and post-quantum KEMs. As simple as this architecture is, it however appears not to be the most efficient, computationally speaking as well as regarding the bandwidth of the exchanges. Hence, we propose a new method to hybridize several KEMs more effectively, by combining the underlying Public Key Encryption schemes (PKEs) in an innovative variant of the cascade composition that we call "leaking-cascade". We prove that this architecture constitutes an IND-CPA-secure robust combiner for the encryption schemes, which permits to create an IND-CCA2 KEM upon the generated hybrid PKE. The leaking-cascade is at least as computationally effective as the commonly used parallel combination, and has a bandwidth gain - when it comes to the ciphertext produced - that may exceed 13 % compared to the latter.

Keywords: PKE combiner · KEM hybridization · Cascade · Post-Quantum Cryptography · Hybrid Key Exchange

1 Introduction

Faced to the looming threat posed by the quantum computer to the classical public key cryptography, the American National Institute of Standards and Technology (NIST) launched in 2017 a Post-Quantum Cryptography (PQC) competition aiming to select the best post-quantum KEMs and signature algorithms. KEMs, in particular, were chosen as basic bricks to ensure the public key encryption functionality.

A KEM is a black-box algorithm based on a public key encryption scheme which consists, for a sender, in randomly drawing a symmetric key and encrypting it with the related PKE. The recipient is then able to decapsulate the received ciphertext and recover the transmitted key. Many KEMs, and especially most PQ ones, are built by applying on an OW-CPA[4] or IND-CPA-secure PKE an operation named "Fujisaki-Okamoto (FO) transformation" ([FO99]) which turns the whole scheme into an IND-CCA2 KEM.

NIST's PQC competition reached a turning point in summer 2022, when a KEM (Crystals Kyber, see [BDK+17]) and several signature schemes were selected for standardization. One may thus be tempted to use Kyber from now on in order to avoid the "harvest now - decrypt later" attacks that may occur with the quantum computer. However, due to the lack of maturity of post-quantum cryptography as a field of research, especially when it comes to the parametrization of real-life implementations of PQ primitives, it appears much safer to hybridize the KEMs by using simultaneously classical and PQ algorithms, in order to benefit from the best security of both worlds, yet at the cost of some overhead.

The PQ KEM hybridization that is currently most studied is the double one, where a classical key exchange scheme is combined with a single PQ KEM. However, when it comes to sensitive data

[4] One-Way Chosen-Plaintext Attacks. This encryption security model, weaker than IND-CPA, only aims to prevent the full recovery of the plaintext by an adversary having access to an encryption oracle.

whose secrecy must be ensured for at least few decades (and which are, consequently, particularly vulnerable to the aforementioned "harvest now - decrypt later" attacks), some users may want to ensure that they do not rely on only one PQ cryptosystem that could be shown insecure in a couple of years, in which case the hybridization would have been totally useless. A n-hybridization with one classical algorithm and two or more PQ KEMs solves this problem, but it suffers from an important computational and bandwidth overhead that should be limited as much as possible.

There are two different ways to carry out a KEM hybridization: either with a straightforward combination of the selected KEMs or by the combination of the PKEs comprised in these KEMs.

KEM combination The first method - the combination of KEMs seen as black-boxes - is the most common. Its classical architecture is the parallel composition (cf. section 2.4) where all component KEMs are running in parallel and output their own symmetric key and related ciphertext. Then, a dedicated primitive named "key combiner" (aka "core function" in [GHP18]) mixes the symmetric keys with the ciphertexts to produce a combined key.

Using KEMs as core components is a natural approach, since these primitives were precisely targeted by the NIST for standardization in its PQC competition. However, the parallel composition of KEMs appears not to be the most effective way to generate an hybrid KEM, for the following reasons:

- Regarding the computation cost, combining n KEMs built with a FO transformation means carrying out this FO transformation n times as well, whereas we would like to reduce this number to a single operation.
- More importantly, the bandwidth of a key exchange using such an hybrid KEM (n public keys and n ciphertexts) is not optimized, and this issue is even more pronounced with PQ algorithms that are all quite cumbersome.

PKE combination The other - less studied - method is the combination of the PKEs within the KEMs that we want to hybridize. The main idea here is to save the computation cost of $n-1$ FO transformations, as this one must be applied only once, on the hybrid PKE. Furthermore, as the latter only gets to be OW or IND-CPA secure, the key combiner can afford to be simpler - and thus computationally more sober - than in the IND-CCA2 security model. Indeed, it was proven by [GHP18] that merely XORing the keys of component IND-CPA KEMs was sufficient to ensure the same security for the hybrid algorithm[5].

To the best of our knowledge, the only study of an hybrid KEM based on a PKE combination is [HV21], which presents a generic construction where PKEs run in parallel composition before the resulting hybrid PKE undergo a single FO transformation. However, the gains of this construction, compared to a KEM combination, are purely computational and in particular, the bandwidth is leaved unaltered.

Outside the scope of public key cryptography, the combination of encryption primitives has been far more studied, and notably the cascade composition (cf. section 2.4). The early works of [EG85] and [MM93] focus on cascades of block ciphers; the former proves the security of the hybrid scheme against message-recovery attacks, whereas the latter shows that a cascade of ciphers is at least as secure as the first cipher but that the security of the construction cannot be ensured, in the corresponding security model, by only the second cipher.

At a higher level, [Her02] proves that a cascade of encryption schemes yields IND-RCCA security[6] if at least one of its components achieves that security. In a similar study, [ZHSI04] introduces the notion of "multi-encryption" and shows that the cascade scheme can reach IND-gCCA security[6] in their slightly different multi-encryption security model.

[5] This result can be extended to the case of encryption schemes.

[6] IND-RCCA ([CKN03]) and IND-gCCA([ADR02]) are relaxed versions of the IND-CCA2 security notion, designed to exclude trivial security failures in the IND-CCA2 model that come from "benign malleability".

Our contributions and outline of this paper We present in the section 3 of this paper a new way to combine encryption schemes, designed to keep the computational gains of the PKE parallel combination of [HV21] while offering a lighter bandwidth. This construction, that we call "leaking-cascade", is a mix between parallel and cascade combinations. We prove that our leaking-cascade scheme is a robust encryption combiner for IND-CPA security (in the sense of Definition 1), which is a sufficient condition to build an IND-CCA2 KEM from the hybrid PKE, *via* a FO transformation.

We study in section 4 the functionalities offered by such an architecture, mainly the aforementioned KEM hybridization but also a lighter Authenticated Key Exchange (AKE) protocol. We also detail in appendix A some possible improvements, including the Integrated Diffie-Hellman (IDH) KEM in which the first classical PKE in the cascade chain is replaced by a Diffie-Hellman key agreement scheme.

We then instantiate in section 5 two types of leaking-cascade hybrid KEMs:
- a first one combining a classical encryption scheme (ElGamal) and a PQ PKE (Crystals Kyber);
- a second one, where another PQ PKE (either NTRU-HRSS ([HRSS17]), its variant NTRU' from [SXY18] or BAT ([FKPY22])) is added to the former double hybridization.

2 Preliminaries

2.1 Notations and terminology

In all the document, a sampling from a space $\mathcal{S}$ with uniform distribution is represented by "$x \xleftarrow{\$} \mathcal{S}$". The output of a probabilistic algorithm is represented by "$\leftarrow$" and the one of a deterministic algorithm is given by ":=".

"||" and "Cat( )" are used for the concatenation operation. "[ ]" denotes optional values or parameters. $\lfloor\ \rceil$ and $\lceil\ \rceil$ respectively denote the rounding and ceiling values of a decimal number.

In the case of lattice problems, vectors are written in lower case bold characters, matrices in upper case characters and scalars in lower case roman characters.

In the whole paper, we include both symmetric and public key encryption schemes with the generic term of "encryption scheme", our leaking-cascade combiner being proved secure for both primitives. However, for practical purpose, we tend to take the notations of public key cryptography in most instances and figures, as we use PKEs to build KEMs. This does not narrow the range of our study.

2.2 Public Key Encryption schemes

Description A probabilistic public key encryption scheme $\mathcal{E}$ is a tuple of algorithms (KeyGen, Enc, Dec) such that:
- **KeyGen** takes as optional input the security parameter λ and probabilistically outputs a couple of public and private keys: $\boxed{(pk, sk) \leftarrow KeyGen([\lambda])}$.
- **Enc** takes as input a public key $pk \in \mathcal{PK}$ and a plaintext $m \in \mathcal{M}$. It draws a random coin $r \xleftarrow{\$} \mathcal{R}$ and probabilistically yields a ciphertext c: $\boxed{c \leftarrow Enc(pk, m; r)}$.
- **Dec** takes as input a secret key $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$ and deterministically generates a plaintext m: $\boxed{m := Dec(sk, c)}$.

  If the encryption scheme is correct, we have:
  $\forall (m, r) \in \mathcal{M} \times \mathcal{R}, \forall (pk, sk) \leftarrow KeyGen(),\ Dec(sk, Enc(pk, m; r)) = m.$

Security The security game corresponding to the IND-CPA security of a PKE $\mathcal{E}$, with two experiments $\mathbf{Exp}^{\mathrm{IND-CPA-b}}$ ($b \in \{0, 1\}$), is detailed in Figure 1. The advantage of any probabilistic polynomial-time (PPT) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in this model (where $\mathcal{A}$ outputs 1 and wins the security experiment $\mathbf{Exp}^{\mathrm{IND-CPA-b}}$ if its guessing bit $\hat{b}$ equals b) is:

$$adv_{\mathcal{E}}^{IND-CPA}(\mathcal{A}) = \mid Pr[\mathcal{A} = 1]_{\mathcal{E}}^{Exp\ IND-CPA-1} - Pr[\mathcal{A} = 1]_{\mathcal{E}}^{Exp\ IND-CPA-0} \mid.$$

Experiment $\mathbf{Exp}^{\text{IND-CPA-b}}$	Encryption oracle $\mathcal{O}^{Enc}(m)$
1 : $(pk, sk) \leftarrow \text{KeyGen}()$	1 : $c \leftarrow \text{Enc}(pk, m)$
2 : $(m^0, m^1, s) \leftarrow \mathcal{A}_1^{\mathcal{O}^{Enc}(.)}(pk)$	2 : **return** c
3 : $c^* \leftarrow \text{Enc}(m^b)$	
4 : $\hat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}^{Enc}(.)}(pk, c^*, m^0, m^1, s)$	
5 : **return** $\hat{b}$	

Fig. 1: IND-CPA security game of a public key encryption scheme.

2.3 Key Encapsulation Mechanisms (KEMs)

Description As stated above, the goal of a KEM is to perform a (symmetric) "key transport" functionality, thanks to a PKE that permits the encryption and decryption of the randomly chosen symmetric key that has to be transmitted. It is a tuple of algorithms defined as follows:

- **KeyGen** takes as optional input the security parameter λ and probabilistically outputs a couple of public and private keys: $\boxed{(pk, sk) \leftarrow KeyGen([\lambda])}$.
- **Encaps** takes as input a public key $pk \in \mathcal{PK}$. It internally randomly draws a symmetric key $k \xleftarrow{\$} \mathcal{K}$ and probabilistically encapsulates it within a ciphertext c: $\boxed{(k, c) \leftarrow Encaps(pk)}$.
- **Decaps** takes as input a secret key $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$ and deterministically recovers the associated symmetric key k: $\boxed{k := Decaps(sk, c)}$.

Security The security that is generally expected for a KEM - and that was explicitly stated by the NIST for its PQC competition, is the IND-CCA2 security. As for the IND-CPA KEM security model, it relies on the symmetric key indistinguishability, but it additionally gives the adversary access to a decapsulation oracle.

Construction / FO transformation A common construction of an IND-CCA2 KEM in the (Quantum) Random Oracle Model ((Q)ROM) consists in applying, on an OW-CPA or IND-CPA secure PKE, an operation called "Fujusaki-Okamoto (FO) transformation" ([FO99]). Nowadays, the mostly used FO transformation is a variant by [HHK17], which follows a modular approach and uses several subroutines applied according to the features of the underlying PKE. The main two of these subroutines are:

- The **"T" transformation**, which turns a probabilistic OW-CPA or IND-CPA PKE into a deterministic OW-PCVA[7] PKE by deriving the randomness r of the encryption scheme from the random plaintext m: $\boxed{r := H(m[\ ||\ pk\ ])}$, with H a cryptographic hash function modeled as a (quantum) random oracle. In the decryption stage, the validity of the decrypted plaintext also generally needs to be checked *via* a reencryption.
- The **"(Q)U$^{\not\perp}_{[\mathbf{m}]}$" transformation**: this operation, which regroups several variants, derives the symmetric key from the random plaintext that was encapsulated (instead of straightly using this message as the key):
  $\boxed{k := G(m\ [\ ||\ c\ ])}$, with G a cryptographic hash function.

Hence we basically have (see [HHK17] for more details):

$$\underset{\text{PPKE}}{^{\text{OW/IND-CPA}}} \xrightarrow{\text{T}} \underset{\text{DPKE}}{^{\text{OW-PCVA}}} \xrightarrow{\text{(Q)U}^{\not\perp}_{[\text{m}]}} \underset{\text{KEM}}{^{\text{IND-CCA2}}}$$

[7] The One-Way Plaintext-Checking and Validity Attacks (OW-PCVA) security notion is OW-CPA security where the adversary has an additional access to both a Plaintext Checking Oracle (which states if a couple (plaintext, ciphertext) is related) and a Ciphertext Validity Oracle (which asserts the validity of a proposed ciphertext).

2.4 KEM and encryption combination

In hybrid cryptography, the goal of combining several primitives is to maintain a certain property if at least a determined number of these primitives have themselves this property. The way they are mixed together is called a combiner, that has to be "robust" in the sense of the following definition from [Her02].

Definition 1 (Robust combiner ([Her02])). *Let $\mathbb{P}$ denote the set of all programs, with a fixed encoding and machine model, e.g. Turing machines. A combiner (of plurality n) is an algorithm c, whose input is a set of n programs $P_1, \cdots, P_n \in \mathbb{P}^n$ and whose output is a single program $c(P_1, \cdots, P_n)$. We say that $c : \mathbb{P}^n \to \mathbb{P}$ is a (k, n)-robust combiner of $\mathbb{P}$ for specification (predicate) $s : \mathbb{P} \to \{0, 1\}$, if:*

$$\forall (P_1, \cdots, P_n) \in \mathbb{P}^n, \quad \sum_{i=1}^{n} s(P_i) \geq k \quad \Rightarrow \quad s\big(c(P_1, \cdots, P_n)\big) = 1$$

Regarding the combination of KEMs or encryption schemes, we focus in $(1, n)$-robust combiners for a certain security level (that represents the specification s), so that the combined scheme yields that security level as long as at least one of the component primitives is itself secure.

We present beneath the two main modes of KEM and encryption scheme combination: the parallel and cascade (sequential) compositions.

Parallel composition The common way to hybridize KEMs or encryption schemes is called the "parallel composition" (cf. [GHP18] for KEM combination). This method consists in running in parallel each one of the n component primitives that we want to hybridize, in order to generate a combined public key $pk := (pk_1, \cdots, pk_n)$, a combined ciphertext $c := c_1 \,||\, \cdots \,||\, c_n$ and, in the case of KEMs, a combined key $k_c := \mathrm{Comb}(k_1, \cdots, k_n, c)$, with Comb a key combiner.

Cascade composition The combination of n encryption schemes $(\mathcal{E}_i := (\mathrm{KeyGen}_i, \mathrm{Enc}_i, \mathrm{Dec}_i))_{i \in [1..n]}$ is called a cascade (aka sequential) composition if every ciphertext output by the first $n-1$ encryption schemes is given as input to the following primitive:

$$\mathrm{Enc}_{casc}((pk_1, \cdots, pk_n), m) := \mathrm{Enc}_n(pk_n, \mathrm{Enc}_{n-1}(pk_{n-1}, \cdots \mathrm{Enc}_1(pk_1, m)))$$

3 Leaking-cascade hybridization of encryption schemes

3.1 Description

Because the ciphertexts of most post-quantum PKEs are much larger than their inputs, it is not possible to combine them in a regular cascade. We consequently conceived, for the purpose of PQ KEM hybridization, the leaking-cascade combination where only a part of the intermediate ciphertexts is encapsulated as in a regular cascade. The rest of these ciphertexts is joined to the last ciphertext c_n of the chain, to produce the global ciphertext of the hybrid scheme (cf. Figure 2 & Figure 3).

This architecture aims to keep most of the advantages brought by the regular cascade hybridization, compared to the more common parallel composition, when the regular cascade combination of encryption schemes appears impossible. It indeed performs an hybridization more effective than the parallel one in terms of bandwidth and can still be proven a robust combiner - in the sense of Definition 1 - under certain assumptions, as detailed beneath.

Definition 2 (Leaking-cascade combination of encryption schemes). *The combination of n encryption schemes $(\mathcal{E}_i := (\mathrm{KeyGen}_i, \mathrm{Enc}_i, \mathrm{Dec}_i))_{i \in [1..n]}$ is called a (partially) **L-leaking-cascade** composition if there exists a non empty subset of the $n-1$ first indices, $L \subseteq \{1, \cdots, n-1\}$, such that:*

1. *For every encryption scheme in this subset $\mathcal{E}_j$ $(j \in L)$, called a "leaking primitive":*

- *the generated ciphertext is of the form* $\boxed{c_j := (u_j, v_j) \text{ or } c_j := (v_j, u_j)}$;
- u_j *constitutes a part of the global ciphertext, along with the output* c_n *of the last scheme of the chain:* $\boxed{c := \text{Cat}((u_j)_{j\in L}) \mid\mid c_n}$;
- v_j *is given as input to the following encryption algorithm* Enc_{j+1}*:* $\boxed{c_{j+1} := \text{Enc}_{j+1}(pk_{j+1}, v_j)}$

2. *The encryption schemes* $(\mathcal{E}_i)_{i\notin L}$ *outside the subset* L *are composed in regular cascade and thus feed the next primitives* Enc_{i+1} *with their full ciphertext* c_i*.*

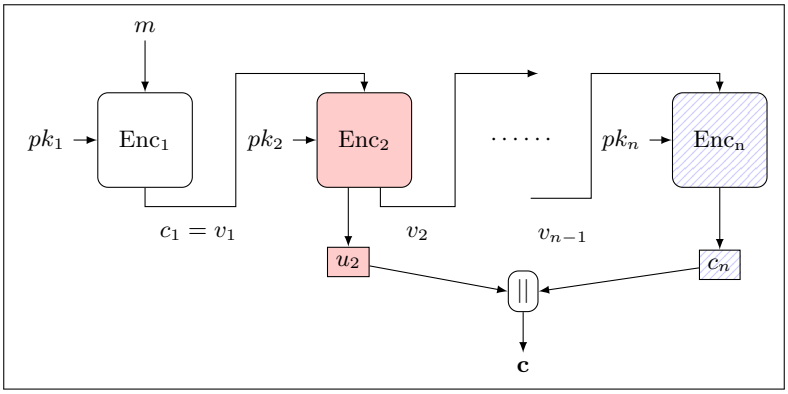

Fig. 2: Architecture of a {2}-leaking-cascade encryption combiner. In this instance, $\mathcal{E}_2$ is thus a leaking primitive which publicly displays a part of its ciphertext (u_2), whereas the encryption algorithms $\mathcal{E}_1$ and $\mathcal{E}_3$ to $\mathcal{E}_n$ are combined in a regular (i.e. non leaking) cascade.

In this paper, we focus on double and triple hybridization, with upstream primitives that are all leaking (i.e. {1}-leaking-cascade double hybridization and {1, 2}-leaking-cascade triple hybridization). We call this type of architecture a fully leaking-cascade, as stated below.

Definition 3 (Fully leaking-cascade). *A fully leaking-cascade composition is a* $\{1, \cdots, n-1\}$*-leaking-cascade scheme, i.e. a leaking-cascade combination where* ***all*** *the* $n-1$ *first primitives* $(\mathcal{E}_i)_{i\in[1..n-1]}$ *are leaking a part of their ciphertexts in the global ciphertext.*

3.2 Partitioned-ciphertext encryption schemes

The security offered by the leaking-cascade architecture relies on an encryption scheme property, that we name "partitioned-ciphertext", which states that the ciphertext yielded by this encryption scheme can be decomposed into two parts, c_r and c_m, such that c_r is not a function of the encrypted plaintext but rather generally depends on the randomness of the probabilistic encryption, whereas c_m comprises the whole plaintext.

As it is proved hereunder, the security of a L-leaking-cascade composition can be ensured only if the leaking-primitives $(\mathcal{E}_j)_{j\in L}$ of the chain have this partitioned-ciphertext property and if their c_m element is included in the encapsulated part v of their ciphertext : $\boxed{c_{m_j} \subseteq v_j,\ \forall j \in L}$ (which, in turn, implies that $\boxed{c_{r_j} \supseteq u_j,\ \forall j \in L}$).

Definition 4 (Partitioned-ciphertext encryption scheme). *An encryption scheme* $\mathcal{E} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ *is said to have "partitioned ciphertexts" if any ciphertext* c *output by the scheme can be decomposed into two bit strings* c_r *and* c_m[8] *such that:*

[8] As in Definition 2, the relative order of c_r and c_m in the ciphertext c is not determined in the general case.

Leaking-cascade $\mathbf{Enc_{lc}}(pk, m, L)$	Leaking-cascade $\mathbf{Dec_{lc}}(sk, c, L)$
$(pk_1, \cdots, pk_n) := Parse(pk)$	$(sk_1, \cdots, sk_n) := Parse(sk)$
$c := \varnothing$	$((u_i)_{i\in L}, c_n) := Parse(c)$
$v_0 := m$	$v_{n-1} := \mathrm{Dec}_n(sk_n, c_n)$
for $i \in [1..n-1]$ **do** :	**for** $i \in [n-1..1]$ **do** :
if $i \in L$ **then** :	**if** $i \in L$ **then** :
$(u_i, v_i) \leftarrow \mathrm{Enc}_i(pk_i, v_{i-1})$	$c_i := (u_i, v_i)$
$c := c \,\|\|\, u_i$	**else** :
else :	$c_i := v_i$
$v_i \leftarrow \mathrm{Enc}_i(pk_i, v_{i-1})$	$v_{i-1} := \mathrm{Dec}_i(sk_i, c_i)$
$c_n \leftarrow \mathrm{Enc}_n(pk_n, v_{n-1})$	$m := v_0$
$c := c \,\|\|\, c_n$	**return** m
return c	

Fig. 3: Encryption and decryption algorithms of a L-leaking-cascade encryption combiner. $L \subseteq \{1, \cdots, n-1\}$ denotes the set of leaking primitives in the cascade chain. The key generation, identical to a parallel combination, is not detailed.

1. *The bit-lengths of c_r and c_m are constant over the message space $\mathcal{M}$;*
2. *c_r is not a function of the plaintext m.*

More formally: $\forall pk \in \mathcal{PK},\ \forall r \in \mathcal{R},\ \forall (m, m') \in \mathcal{M}^2$,
$(c_r, c_m) := \mathrm{Enc}(pk, m; r)$ and $(c'_r, c'_m) := \mathrm{Enc}(pk, m'; r)$ s.t.
1. $|c_r| = |c'_r|$ and $|c_m| = |c'_m|$
2. $c_r = c'_r$.

A common instance of a partitioned-ciphertext classical encryption scheme is ElGamal. Regarding PQ algorithms, the PKEs of three of the four KEMs that are most likely to be standardized in the future[9] have this partitioned-ciphertext property: Kyber, BIKE and HQC. The proofs of that statement for ElGamal and Kyber, which are specifically implemented in our instantiations of section 5, are detailed in appendix B.

3.3 IND-CPA security of leaking-cascade

Theorem 1. *The combination of n encryption schemes $(\mathcal{E}_i = (\mathrm{KeyGen}_i, \mathrm{Enc}_i, \mathrm{Dec}_i))_{i\in[1..n]}$ in a L-leaking-cascade mode constitutes a (1, n)-robust encryption combiner for IND-CPA security, in the sense of Definition 1[10], if all leaking primitives are partitioned-ciphertext algorithms whose c_m part of the ciphertext is encapsulated by the following primitive (i.e. $c_{m_j} \subseteq v_j,\ \forall j \in L$).*

Proof sketch Our security proof of the generic leaking-cascade construction relies on the security of a particular case: the combination of two encryption schemes in leaking-cascade. The case of a leaking-cascade with an arbitrary number n of primitives is then deducted from the latter by an simple induction argument.

The 2-primitives leaking-cascade proof is decomposed in two parts, corresponding to the following statements:

[9] These promising algorithms are the current winner of round 3 of NIST's PQC competition (Crystals Kyber) as well as the remaining three candidates of the fourth round, studied as an alternative solution to Kyber (BIKE, ClassicMcEliece & HQC).

[10] In other terms, the hybrid scheme is IND-CPA secure if at least anyone of its component primitives is IND-CPA secure as well.

1. The combination in leaking-cascade mode of two encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$ yields IND-CPA security if the first scheme $\mathcal{E}_1$ is IND-CPA, regardless of the security of the second scheme $\mathcal{E}_2$.
2. The combination in leaking-cascade mode of two encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$ yields IND-CPA security if the second scheme $\mathcal{E}_2$ is IND-CPA and if the first scheme $\mathcal{E}_1$ is a partitioned-ciphertext algorithm s.t. $c_{m_1} \subseteq v_1$.

Proof of statement 1 Let us consider a leaking-cascade hybrid encryption scheme $\mathcal{E}$ composed of two component encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$, with the first scheme $\mathcal{E}_1$ being IND-CPA-secure. No security property is demanded for the second scheme $\mathcal{E}_2$.

Game 0 This is the experiment $\mathbf{Exp}^{\text{IND-CPA-0}}_{leak-casc}$ of the leaking-cascade security game (cf. Figure 1). The global ciphertext produced by the hybrid scheme is : $c \leftarrow u^0 \,||\, \text{Enc}_2(pk_2, v^0)$, with $(u^0, v^0) \leftarrow \text{Enc}_1(pk_1, m^0)$

Game 1 We replace in this game both terms u^0 and v^0 by u^1 and v^1, with $(u^1, v^1) \leftarrow \text{Enc}_1(pk_1, m^1)$: $c \leftarrow \boxed{u^1} \,||\, \text{Enc}_2(pk_2, \boxed{v^1})$.

The advantage of any attacker to distinguish Game 1 from Game 0 is:
$\epsilon_1 \leq adv^{IND-CPA}_{\mathcal{E}_1}$.

Proof. From any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ trying to distinguish Games 0 and 1, we can build an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the CPA-indistinguishability of the encryption scheme $\mathcal{E}_1$, as follows:

- $\mathcal{B}_1$ acts as a challenger for $\mathcal{A}_1$ and thus collects the encryption request (m^0, m^1) sent by $\mathcal{A}_1$, which it straightly forwards to its own challenger.
- From its challenger's answer $(c_1^* = (u^b, v^b) \leftarrow \mathcal{O}^{Enc}_{\mathcal{E}_1})$, $\mathcal{B}_1$ computes $c^* \leftarrow (u^b \,||\, \text{Enc}_2(pk_2, v^b))$.
- $\mathcal{B}_2$ runs then the adversary $\mathcal{A}_2^{\mathcal{O}^{Enc}_{lc}(.)}$ with this input c^* and forwards to its challenger the guessing bit output by $\mathcal{A}_2$.

Consequently, $\mathcal{A}$'s advantage is bounded as follows:
$\epsilon_1 = adv^{G0-G1}(\mathcal{A}) \leq adv^{IND-CPA}_{\mathcal{E}_1}(\mathcal{B})$.

Game 1 corresponds to the experiment $\mathbf{Exp}^{\text{IND-CPA-1}}_{leak-casc}$ of the leaking-cascade security game. Consequently, the overall advantage of an adversary against a leaking-cascade encryption scheme with an IND-CPA-secure first encryption scheme is: $\boxed{adv^{IND-CPA}_{leak-casc} \leq adv^{IND-CPA}_{\mathcal{E}_1}}$.

Proof of statement 2 Let us now consider a similar leaking-cascade hybrid encryption scheme $\mathcal{E}$ comprising an IND-CPA-secure second scheme $\mathcal{E}_2$ and a first scheme $\mathcal{E}_1$ with partitioned-ciphertext property and s.t. $c_{m_1} \subseteq v_1$.

Game 0 This is the experiment $\mathbf{Exp}^{\text{IND-CPA-0}}_{leak-casc}$ of the leaking-cascade security game (cf. Figure 1). The global ciphertext produced by the hybrid scheme is : $c \leftarrow u^0 \,||\, \text{Enc}_2(pk_2, v^0)$, with $(u^0, v^0) \leftarrow \text{Enc}_1(pk_1, m^0)$.

Game 1 We replace in this game the term v^0 by the ciphertext $\tilde{v}$ of a random message $\tilde{m}$ from the message space: $(\tilde{u}, \tilde{v}) \leftarrow \text{Enc}_1(pk_1, \tilde{m})$, with $\tilde{m} \xleftarrow{\$} \mathcal{M}$. Thus, we have: $c \leftarrow u^0 \,||\, \text{Enc}_2(pk_2, \boxed{\tilde{v}})$.

The advantage of any attacker $\mathcal{A}$ to distinguish Game 0 from Game 1 is bounded by the IND-CPA security of the scheme $\mathcal{E}_2$: $\epsilon_1 \leq adv^{IND-CPA}_{\mathcal{E}_2}$.

Proof. We can once again construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the CPA-indistinguishability of the encryption scheme $\mathcal{E}_2$ from any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ trying to distinguish Games 0 and 1, as detailed in Figure 4.

Adv. $(\mathcal{B}_1)^{\varepsilon_2}$	**Adv.** $(\mathcal{B}_2)^{\varepsilon_2}(c_2^*, u^0, s)$	**Enc. oracle** $\mathcal{O}^{Enc}_{G0-G1}(m\ [,\ u^0])$
$(pk_1, sk_1) \leftarrow \text{KeyGen}_1()$	$c^* := u^0 \mid\mid c_2^*$	**if** $u^0 = \varnothing$ **then** :
$(m^0, \tilde{m}) \xleftarrow{\$} \mathcal{M}^2$	$\hat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}^{Enc}_{G0-G1}(\cdot, u^0)}(c^*)$	$m^0 \xleftarrow{\$} \mathcal{M}$
$(u^0, v^0) \leftarrow \text{Enc}_1(pk_1, m^0)$	**return** $\hat{b}$	$(u^0, v^0) \leftarrow \text{Enc}_1(pk_1, m^0)$
$(\tilde{u}, \tilde{v}) \leftarrow \text{Enc}_1(pk_1, \tilde{m})$		$(u, v) \leftarrow \text{Enc}_1(pk_1, m)$
$s \leftarrow \mathcal{A}_1^{\mathcal{O}^{Enc}_{G0-G1}(\cdot)}(u^0)$		$c_2 \leftarrow \mathcal{O}^{Enc}_{\mathcal{E}_2}(v)$
return $((v^0, \tilde{v}), u^0, s)$		$c := u^0 \mid\mid c_2$
		return c

Fig. 4: Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the CPA-indistinguishability of the second encryption scheme $\mathcal{E}_2$ of a leaking-cascade, based on an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the indistinguishability of games G0 and G1.

Game 2 We replace u^0 by u^1: $\quad c \leftarrow \boxed{u^1} \mid\mid \text{Enc}_2(pk_2, \tilde{v})$.

Neither u^0 nor u^1 is related in any way to the term $\text{Enc}_2(pk_2, \tilde{v})$, since $\tilde{v}$ originates from a randomly drawn element $\tilde{m}$ from $\mathcal{M}$. Moreover, u^0 and u^1 follow the exact same distribution independent from their underneath plaintexts m^0 and m^1, because of the "partitioned-ciphertext" property of the encryption scheme $\mathcal{E}_1$ and the fact that $c_{m_1} \subseteq v_1$. Consequently, the advantage of any attacker in distinguishing Game 2 from Game 1 is null: $\epsilon_2 = 0$.

Game 3 We replace $\tilde{v}$ by v^1: $\quad c \leftarrow u^1 \mid\mid \text{Enc}_2(pk_2, \boxed{v^1})$.

Similarly to the passage from Game 0 to Game 1, the advantage of any attacker to distinguish Game 3 from Game 2 is : $\epsilon_3 \leq adv^{IND-CPA}_{\mathcal{E}_2}$.

Game 3 corresponds to the experiment $\mathbf{Exp}^{\text{IND-CPA}-1}_{leak-casc}$ of the leaking-cascade security game. Consequently, the overall advantage of an adversary against a leaking-cascade encryption scheme with an IND-CPA secure second encryption scheme and a first "partitioned-ciphertext" encryption scheme, is:
$\boxed{adv^{IND-CPA}_{leak-casc} \leq 2.adv^{IND-CPA}_{\mathcal{E}_2}}$, which completes the proof of the theorem.

4 Functionalities of a leaking-cascade

We present hereunder two functionalities brought by the leaking-cascade combination of encryption schemes.

4.1 KEM hybridization

At a higher level than the combination of encryption schemes, the leaking-cascade architecture is used for KEM hybridization, in order to maintain the security of the hybrid encapsulation scheme even in case of failure of some of its components. A (leaking-)cascade KEM hybridization is realized by combining the PKEs of the KEMs we want to associate in order to generate an hybrid PKE, which in turn is transformed into an hybrid KEM with a FO transformation.

Enhanced variations We study two possible enhancements of our leaking-cascade KEM hybridization, that are detailed in appendix A.

Integrated Diffie-Hellman (IDH) KEM The IDH KEM is a cascade combination of one or several PQ PKEs with a Diffie-Hellman (DH) key agreement scheme (in replacement of the classical encryption scheme). The idea here is to be more efficient in terms of bandwidth, given the fact that

a DH key agreement scheme produces a ciphertext[11] lighter than a similar encryption algorithm (twice smaller, for instance, than its ElGamal counterpart).

Integration of the public keys Another enhancement consists in the encapsulation in one another of the public keys of the PKEs to be hybridized, in order to gain once again some bandwidth. We provide here a method to encapsulate an ECDH or ElGamal public key in Kyber's one. This proposal is unfortunately not generically applicable to all classical or PQ algorithms, since it depends on the structure of the wrapping public key. However, the fact that Kyber and the ECDH schemes are the most commonly used key exchange primitives, respectively in the classical and post-quantum settings, mitigates this drawback.

4.2 Authenticated key exchange with leaking-cascade

The cascade combination may also be used in the context of an authenticated key exchange, where the parties would be implicitly authenticated by the additional use of KEMs instead of signatures. This approach, stated for instance by [BDK+17], may appear particularly interesting in the framework of post-quantum authentication, as most current PQ signatures still suffer from a larger bandwidth than their KEM counterparts.

Figure 5 compares a standard implicit Unilateral Authenticated Key Exchange (UAKE) with a similar protocol using a leaking-cascade combiner.

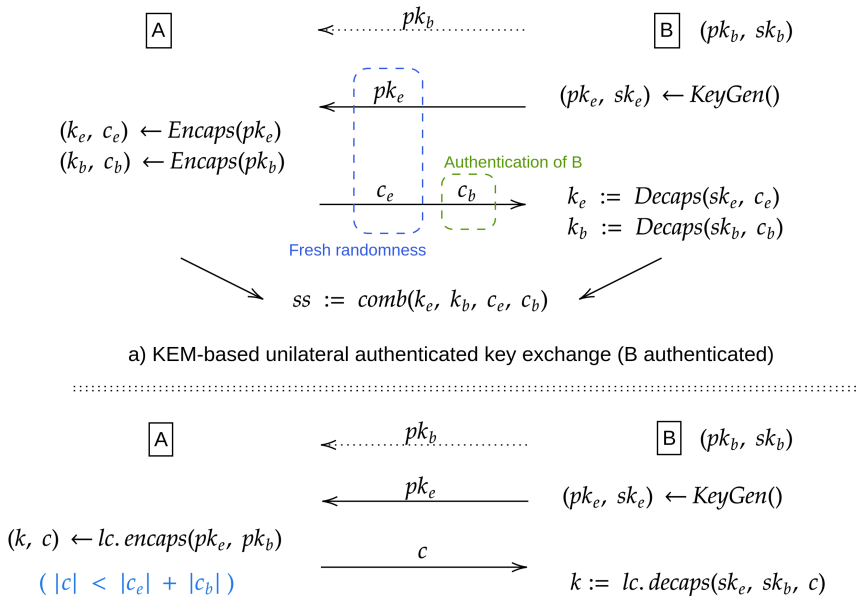

Fig. 5: KEM-based implicit UAKE, with authentication of party B. Figure *a* shows a standard authentication procedure, as proposed in [BDK+17]. Figure *b* details the variant with a leaking-cascade hybrid-KEM.

Here, the (leaking-)cascade architecture does no longer have to prevent the failure of some of the KEMs. It instead plays the role of a multi-key encryption scheme, for which several secret keys (the ephemeral one, used to refresh the randomness, as well as the recipient's static one, used as an authentication mean) are needed to decrypt a ciphertext.

[11] We model here the "ciphertext" of a DH key agreement scheme by the public DH element output by the sender. See appendix A for additional details.

In this paradigm, we can rely on the same algorithm for the whole cascade. As a consequence, the initial constraint of Theorem 1, stating that the upstream PKE must be partitioned ciphertext, may be relaxed (since this property is only needed, in the security proof of the leaking-cascade, when the first encryption scheme is broken and the second one remains secure).

5 Instantiations and performances

5.1 Instantiations

We now detail the instantiation of two hybrid schemes that we esteem particularly relevant in real world applications:

Leaking-cascade double-hybridization The first hybrid KEM consists in a {1}-leaking-cascade combination of the classical ElGamal cryptosystem (on elliptic curves) and the PKE of the PQ KEM Crystals Kyber.

As ElGamal yields partitioned ciphertexts (see the proof in appendix B), according to Theorem 1, the resulting hybrid PKE has IND-CPA security and the related KEM, when applied the FO transformation, is IND-CCA2 secure.

Leaking-cascade triple-hybridization The second hybrid KEM combiner, adapted to highly sensitive data, comes from a triple hybridization in {1,2}-leaking-cascade composition, with one classical encryption scheme and two post-quantum PKEs (cf. Figure 6). In this case, the choice of PQ algorithms is strongly constrained by the ciphertext length of the upstream PQ PKE and the input size of the downstream PQ PKE.

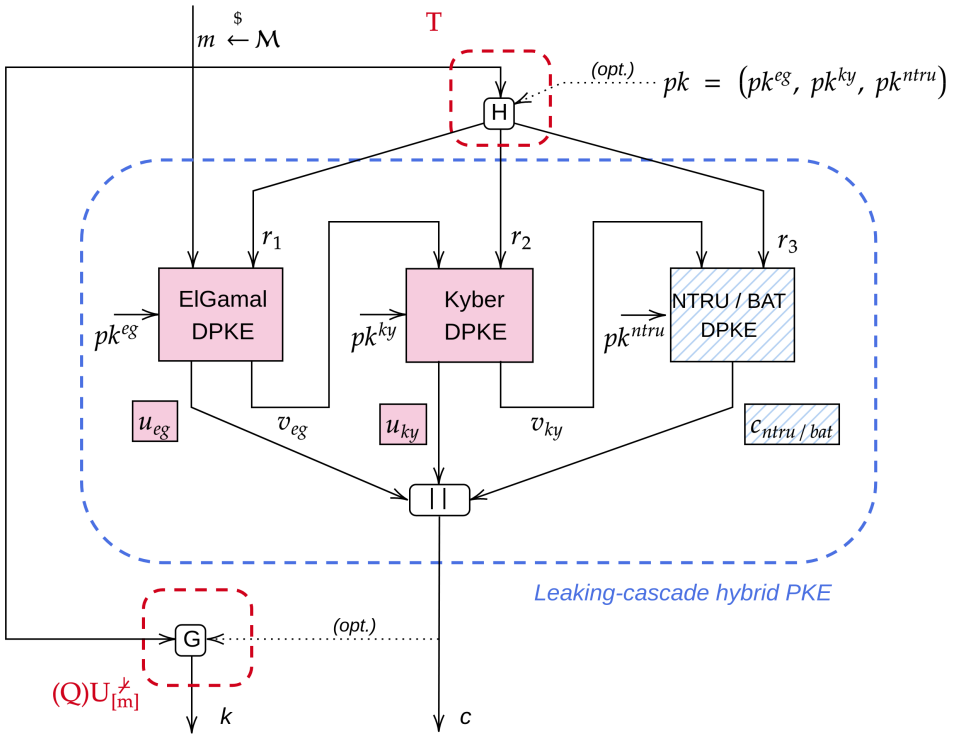

Fig. 6: Architecture of the triple hybridization in {1,2}-leaking-cascade ElGamal-Kyber-NTRU/BAT. The leaking PKEs, here both ElGamal and Kyber, are depicted in pink color. The dashed blue box represents the hybrid PKE and the dashed red boxes, the modular FO transformation.

In our instantiation, the classical algorithm and the upstream PQ PKE are the same as in the double hybridization: ElGamal and Kyber. The security of the combiner is ensured by the fact that Kyber, which is a leaking primitive in the scheme along with ElGamal, is also a partitioned-ciphertext encryption scheme (cf. proof in appendix B).

Regarding the downstream PQ algorithm, we considered two candidates both based on NTRU lattices: NTRU-HRSS ([HRSS17]) and the more recent algorithm BAT ([FKPY22]). We also implemented a variant of NTRU-HRSS proposed by [SXY18], that we call NTRU', based on a tweak by [PS96], where the probabilistic NTRU-HRSS PKE is turned into a deterministic one by including the randomness r into the input of the PKE, along with the plaintext m (since it is possible to recover r as well in the decryption process).

The criteria considered for this choice were:

- The good performances of these algorithms, that make them realistic alternatives to Kyber in case this one would be broken.
- Their differences with Kyber in terms of mathematical foundations, so that a vulnerability on Kyber or even on the Module-Learning with Error problem would not necessarily compromise them.
- More importantly, the input lengths[12] of their PKEs, of 140 bytes for NTRU-HRSS and 160 bytes for BAT, permit the encapsulation of the 128 byte-long c_m part of Kyber's ciphertext ($c_{m_{ky}} \subseteq v_{ky}$), which is the necessary condition to implement a secure leaking-cascade combination of these KEMs.

We underline that NTRU-HRSS PKE takes as input a bit string that is encoded afterwards into a ternary polynomial. Because the original encoding algorithm does not permit to recover, in the decryption process, an arbitrary input given to the PKE (which is not a problem when NTRU is only used as a KEM), we had to slightly modify this encoding algorithm. Our modification is presented in appendix C.

5.2 Performances

Computational cost The computational advantage of our (leaking-)cascade combiner upon the parallel hybridization comes from the fact that it avoids the computations of the key combination and of all but one FO transformations.

As a proof of concept, we tested, on a laptop running Ubuntu 22.04 and equipped with an Intel(R) Core(TM) i7-8565U @1.80GHz octo-core CPU, a double KEM hybridization based on ElGamal and Kyber, implemented in Python.

Operation	Running time (ms)	
	Parallel comb.	Cascade comb.
Encapsulation	83.20	82.85
Decapsulation	120.59	119.93

Fig. 7: Compared **computational performances** of parallel and leaking-cascade combinations of an ElGamal-Kyber KEM hybridization.

Figure 7 details the average running time, over 2,500 runs, of the encapsulation and decapsulation operations[13] for both parallel and leaking-cascade schemes. It indeed confirms a slightly better computational performance of the leaking-cascade scheme, mainly due to a lesser number of hashing operations resulting to the single FO transformation.

This computational advantage is mitigated, to some extent, by the possibility offered by the parallel composition to parallelize the computations, which is not possible for a cascade construction. We consider nevertheless that in most use cases where computational optimization really

[12] For the parameters corresponding to the NIST security level 3.

[13] The key generation stage was not evaluated, as it is performed similarly in both schemes.

matters, in particular servers performing a huge number of operations, the parallelization of subroutines of a single encapsulation or decapsulation is less interesting than the parallelization of these encapsulations or decapsulations themselves.

Bandwidth The main advantage of an hybridization in (leaking-)cascade is the gain in terms of bandwidth. The gains brought by this construction depend on the features of the PKEs used in the chain: ciphertexts lengths of all primitives and input sizes of the last $n-1$ PKEs.

Hybridization	Parallel comb. |ct| (bytes)	Leak-casc. comb. |ct| (bytes)	Gain of leak-casc. comb.
KEM hybridization			
ElGamal_Kyber	1,152	1,120	2.8 %
ElGamal_BAT	1,070	1,006	6.0 %
ElGamal_Kyber_NTRU	2,290	2,121	7.4 %
ElGamal_Kyber_BAT	2,158	1,966	8.9 %
ElGamal_Kyber_NTRU'	2,290	1,984	13.4 %
Authentication			
Kyber_Kyber	2,176	2,144	1.5 %
NTRU_NTRU	2,276	2,139	6.0 %
BAT_BAT	2,012	1,852	8.0 %
NTRU'_NTRU'	2,276	2,002	12.0 %

Fig. 8: Compared **bandwidth performances** of parallel and leaking-cascade combinations for different types of hybridizations and KEM-based authentications relying on ElGamal, Kyber, NTRU, NTRU' and BAT PKEs.

In our instantiations of KEM hybridization (cf. Figure 8), the best optimization brought by our leaking-cascade scheme exceeds 13 %, in the case of the triple hybridization ElGamal_Kyber_NTRU', with a reduction of 306 bytes in the output ciphertext.

Even though this improvement remains moderate, the (leaking-)cascade combination offers interesting perspectives for lighter constructions in the future, as its performances closely depend on the primitives involved, which may appear in the future more appropriate to such a combination. Furthermore, in some use cases, even a small decrease in the size of the data transmitted can avoid reaching the maximum transmission size of a packet in a network, and thus its fragmentation.

References

ADR02. Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, EUROCRYPT 2002, volume 2332 of LNCS, pages 83–107. Springer, Heidelberg, April / May 2002.

BDK+17. Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634, 2017. .

Ber06. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, PKC 2006, volume 3958 of LNCS, pages 207–228. Springer, Heidelberg, April 2006.

CDS+19. Congwu Chen, Oussama Danba, Jennifer Stein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. Ntru algorithm specifications and supporting documentation. 2019.

CKN03. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, CRYPTO 2003, volume 2729 of LNCS, pages 565–582. Springer, Heidelberg, August 2003.

DK05. Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In Joe Kilian, editor, TCC 2005, volume 3378 of LNCS, pages 188–209. Springer, Heidelberg, February 2005.

EG85. S. Even and O. Goldreich. On the power of cascade ciphers. ACM Tras. Computer Systems, 3:108–116, 1985.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, Advances in Cryptology, pages 10–18, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

FKPY22. Pierre-Alain Fouque, Paul Kirchner, Thomas Pornin, and Yang Yu. BAT: Small and fast KEM over NTRU lattices. Cryptology ePrint Archive, Report 2022/031, 2022. .

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, CRYPTO'99, volume 1666 of LNCS, pages 537–554. Springer, Heidelberg, August 1999.

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, Heidelberg, March 2018.

GLN02. Oded Goldreich, Yoad Lustig, and Moni Naor. On chosen ciphertext security of multiple encryptions. Cryptology ePrint Archive, Report 2002/089, 2002. .

Her02. Amir Herzberg. Folklore, practice and theory of robust combiners. Cryptology ePrint Archive, Report 2002/135, 2002. .

HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. .

HLW12. Susan Hohenberger, Allison B. Lewko, and Brent Waters. Detecting dangerous queries: A new approach for chosen ciphertext security. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT 2012, volume 7237 of LNCS, pages 663–681. Springer, Heidelberg, April 2012.

HRSS17. Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. Cryptology ePrint Archive, Report 2017/667, 2017. .

HV21. Loïs Huguenin-Dumittan and Serge Vaudenay. FO-like combiners and hybrid post-quantum cryptography. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, CANS 21, volume 13099 of LNCS, pages 225–244. Springer, Heidelberg, December 2021.

MM93. Ueli M. Maurer and James L. Massey. Cascade ciphers: The importance of being first. Journal of Cryptology, 6:55–61, 1993. .

PS96. Jeffrey Hoffstein Jill Pipher and Joseph H. Silverman. Ntru: A new high speed public key cryptosystem. In draft from at CRYPTO 96 rump session, 1996. .

SXY18. Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, EUROCRYPT 2018, Part III, volume 10822 of LNCS, pages 520–551. Springer, Heidelberg, April / May 2018.

ZCW+16. Cong Zhang, David Cash, Xiuhua Wang, Xiaoqi Yu, and Sherman S. M. Chow. Combiners for chosen-ciphertext security. In Thang N. Dinh and My T. Thai, editors, Computing and Combinatorics, pages 257–268, Cham, 2016. Springer International Publishing.

ZHSI04. Rui Zhang, Goichiro Hanaoka, Junji Shikata, and Hideki Imai. On the security of multiple encryption or CCA-security+CCA-security=CCA-security? In Feng Bao, Robert Deng, and Jianying Zhou, editors, PKC 2004, volume 2947 of LNCS, pages 360–374. Springer, Heidelberg, March 2004.

A Enhancements of the cascade combiner

A.1 Integrated Diffie-Hellman (IDH) KEM

As stated in section 4.1, this tweaked cascade combiner replaces, at the first position of the chain, the classical encryption scheme by a Diffie-Hellman key agreement scheme on Elliptic Curves (ECDH).

To do so, we model the recipient's static public DH element as his public key (pk_B^{dh}) and the sender's ephemeral public DH element ($X := G.x$, with $x \xleftarrow{\$} \{0,1\}^n$ and G a public point of the elliptic curve) as his ciphertext.

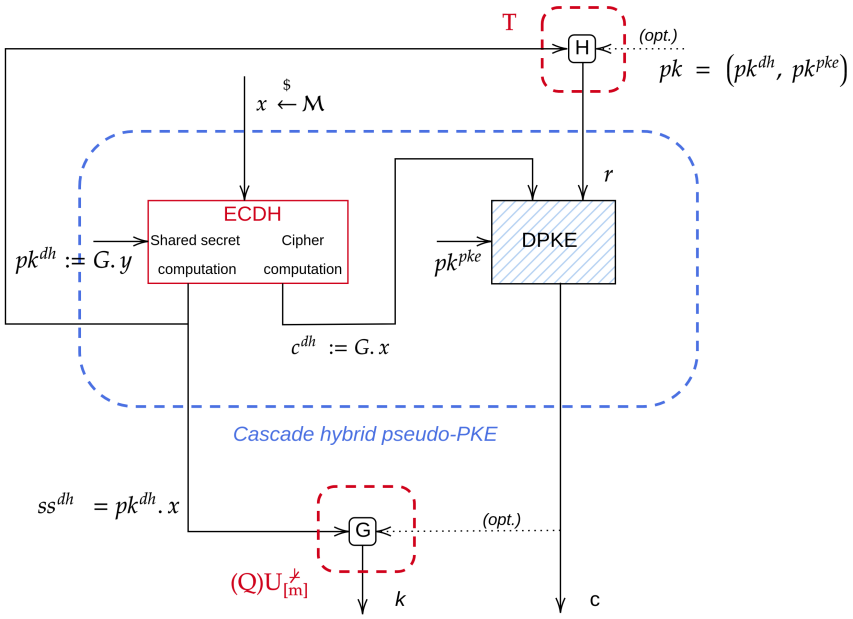

Fig. 9: Encapsulation step of an Integrated Diffie-Hellman KEM. The decapsulator's public DH element $G.y$ is seen as its DH public key pk^{dh} while the encapsulator's public DH element is modeled as a ciphertext $c^{dh} := G.x$.

For identical parameters (choice of the elliptic curve and representation of the points' coordinates), such an agreement scheme is twice lighter than an encryption scheme like ElGamal, since only one point of the curve needs to be transmitted instead of two. Consequently, with an adequate choice of parameters, the ciphertext can be reduced to 32 bytes, which permits to carry out a regular cascade combination instead of a leaking one and saves 32 bytes of bandwidth.

Security considerations Contrary to a standard PKE cascade combiner, the hybrid primitive generated here by the cascade composition is not a real PKE since the decapsulator receiving the ciphertext c is not able to recover the original plaintext x but a shared secret ss^{dh} derived from it.

Nevertheless, the security proof of a PKE cascade-combination (cf. section 3.3) could be adapted to the present case to demonstrate that the hybrid primitive produced is as secure as its most secure component. To do so, we note that:

- the recipient's public DH element $G.y$ has the exact same form than an ElGamal public key, and thus can be considered as his DH public key pk_B^{dh};
- encapsulating the sender's public DH element $G.x$ in the downstream PKE has the exact same effect than encapsulating a "real" PKE ciphertext, in the sense that an adversary will need to break both the downstream PKE and the upstream key agreement to eventually recover the shared secret ss^{dh}.

We then consider that applying a FO transformation on this hybrid primitive has the same effect of generating an IND-CCA2 KEM than it does upon a standard PKE. The main difference

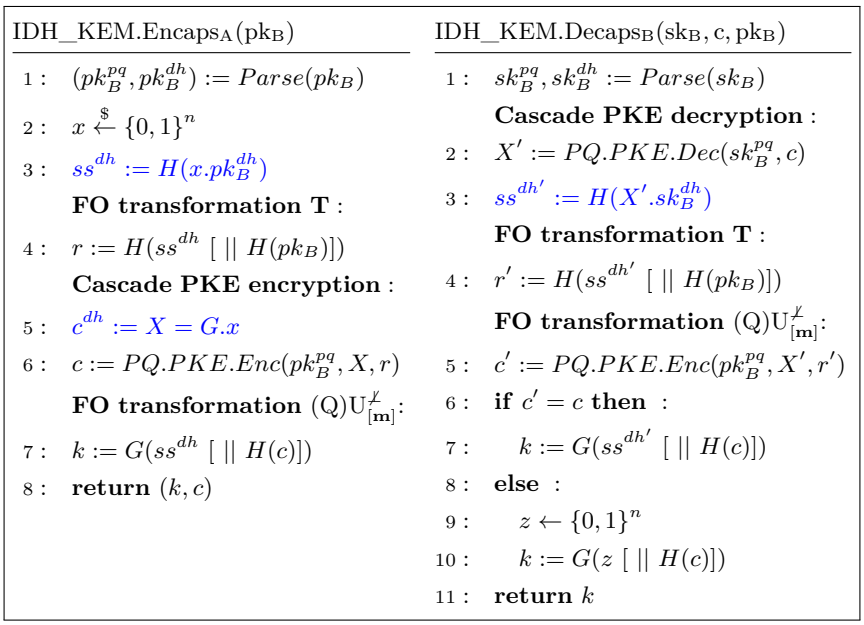

Fig. 10: Encapsulation and decapsulation algorithms of the IDH-KEM. The key generation step is identical to a cascade or a parallel KEM combiner. The brackets represent variants of the FO transformation. The blue lines show the additional computations compared to a single PQ KEM.

here with a standard KEM construction from a PKE is that the transformation (Q)U$^{\not\perp}_{[m]}$ is not applied directly on the random input seed x but on the shared secret ss^{dh} derived from it. As this operation merely consists in hashing the seed, we do not see any difference, in terms of security, to do it on the derived value ss^{dh} instead of on the original seed. Actually, this approach is similar to the FO transformation variant used in Kyber, where (Q)U$^{\not\perp}_{[m]}$ is applied not on the seed m itself but on the pre-key $\hat{k}$ that originates from a hash of this seed.

Instantiation of the ECDH scheme The PKEs of most PQ KEMs are set to take as input a 256 bit-long pseudorandom string and to output a symmetric key of the same length. It is thus necessary to select ECDH algorithms that produce public elements and shared secret of that sizes. X25519, the key-agreement scheme based on Curve25519 (cf. [Ber06]), appears an excellent candidate in that matter, with its public element of precisely 32 bytes long, almost uniformly distributed in $\{0,1\}^{256}$. Its main drawback is that it yields a shared secret of "only" 255 bits, which lowers by one bit the security of the scheme, even after hashing the shared secret into the expected 256 bits string.

A.2 Integration of the public keys

We detail hereunder another possible enhancement, where an encapsulation is no longer carried out on the ciphertexts but on the public keys. Because it relies on the structure of these public keys, this tweak cannot unfortunately be generically applied to any PKE. The principle is to use a short public key (generally from an encryption or key agreement scheme on an elliptic curve with a 256 bit-long order) as a replacement for a seed that is part of another public key.

Our method works for any Learning with Error (LWE), Ring-LWE or Module-LWE cryptosystem. These ones indeed use a public matrix A and an error term $\mathbf{e}$ to hide the secret key $\mathbf{s}$. Because this matrix can be deterministically generated from a random seed, some schemes choose to transmit the seed instead of the whole matrix to save some communication cost. This is precisely the case of the Module-LWE-based Kyber KEM, which only transmits in its public key the seed ρ used to build the matrix A: $\boxed{pk^{ky} := \mathbf{t}^{ky} \,||\, \rho}$, with $\mathbf{t}^{ky} := A.\mathbf{s} + \mathbf{e}$.

When instantiated with any key-agreement scheme and Kyber, this enhancement saves 32 bytes of bandwidth.

As for the security of the scheme, hashing the DH public key (which is almost uniformly distributed on $\{0,1\}^{256}$) yields a string that is, in the (Q)ROM, fully uniformly distributed on

ECDH_Kyber.KeyGen(G)

ECDH key generation :

1 : $y \xleftarrow{\$} \{0,1\}^{256}$

2 : $pk^{dh} := Y = G.y$

3 : $sk^{dh} := y$

Kyber key generation :

3 : $\rho := H(pk^{dh})$ ~~$\rho \xleftarrow{\$} \{0,1\}^{256}$~~

4 : $\sigma \xleftarrow{\$} \{0,1\}^{256}$

5 : $A := \text{Expand}(\rho)$

6 : $(\mathbf{s}^{ky}, \mathbf{e}) := \text{Expand}(\sigma)$

7 : $\mathbf{t}^{ky} := A.\mathbf{s}^{ky} + \mathbf{e}$

8 : **return** $(pk := (\mathbf{t}^{ky}, pk^{dh}), sk := (\mathbf{s}^{ky}, sk^{dh}))$

Fig. 11: Key generation of an ECDH-Kyber combiner with encapsulated public keys. The blue line represents the only change in the key generation of this scheme, compared to a stand-alone execution of Kyber.

$\{0,1\}^{256}$. We consequently esteem that Kyber's implementation should not suffer from this change, even if a formal security analysis would be necessary before any real use of this tweaked scheme.

B Proofs for partitioned-ciphertext algorithms

We detail hereunder the proofs of the partitioned-ciphertext property of ElGamal and Kyber encryption schemes, used in the practical instantiations of section 5.

B.1 ElGamal

Theorem 2. *ElGamal public key encryption algorithm ([ElG85]) is a partitioned-ciphertext encryption scheme.*

Proof. We recall that in a cyclic group G of order q, with a generator g, an ElGamal ciphertext corresponding to a plaintext $m \in \mathcal{M}$ if of the form:
$\boxed{c_{eg} := (c_{r_{eg}} = g^r, c_m = m.X^r)}$, where $X := g^x$ is the recipient's public key and $r \xleftarrow{\$} \mathbb{Z}_q$ a secret random element drawn by the sender.

For a given set of parameters $\{G, q, g\}$, the sizes of the components $c_{r_{eg}}$ and $c_{m_{eg}}$ clearly remain constant for all messages $(m, m') \in \mathcal{M}^2$ s.t. $|m| = |m'|$. Furthermore, $c_{r_{eg}}$ is not a function of the encrypted plaintext m but only of the randomness r, which completes the proof.

B.2 Kyber

Theorem 3. *Crystals Kyber public key encryption algorithm ([BDK+17]) is a partitioned-ciphertext encryption scheme.*

Proof. The ciphertext output by Kyber is of the form $\mathbf{c}_{kyber} := (\mathbf{u}, v)$, with

$$\boxed{\begin{aligned} c_{r_{ky}} &:= \mathbf{u} = \text{Compress}(A^T.\mathbf{r} + \mathbf{e_1}) \\ c_{m_{ky}} &:= v = \text{Compress}(\mathbf{t}^T.\mathbf{r} + e_2 + \left\lfloor \tfrac{q}{2} \right\rceil.m) \end{aligned}}$$

where the modulus $q = 3329$ is a fixed public parameter, $\mathbf{t}$ and $A = \text{Expand}(\rho)$ are issued from the recipient's public key $\boxed{pk := \mathbf{t} \mid\mid \rho}$ and $\mathbf{e_1}$, e_2 and $\mathbf{r}$ are sampled from a secret seed $r \xleftarrow{\$} \{0,1\}^{256}$.

As any semantically-secure encryption scheme, Kyber is"length-uniform" (cf. [Her02]), which implies that: $\forall (pk, pk') \in \mathcal{K}^2, \forall (m, m') \in \mathcal{M}^2, |m| = |m'| \Rightarrow |\text{Enc}(pk, m)| = |\text{Enc}(pk', m')|$.
Consequently: $\forall m \in \mathcal{M}, \exists \kappa \in \mathbb{N} : |c_{r_{ky}}| + |c_{m_{ky}}| = \kappa$.

Furthermore, the element $c_{r_{ky}}$ is not a function of the plaintext m. It only depends on the recipient's public key (since the matrix A directly comes from the random seed ρ from pk) and on the random seed r used to deterministically produce $\mathbf{r}$, $\mathbf{e_1}$ and e_2, ρ and r being themselves independent from m. It thus fulfills condition 2 of Definition 4 and implies that $|c_{r_{ky}}|$ is constant over $\mathcal{M}$. Considering the previous paragraph, we can deduce that $|c_{m_{ky}}|$ is constant over $\mathcal{M}$ as well, which terminates the proof of the theorem.

C NTRU-HRSS encoding variant for the leaking-cascade

C.1 Original input encoding of the NTRU-HRSS PKE

As indicated in subsection 5.1, the PKE of NTRU-HRSS[14] cannot recover any arbitrary input byte array. Indeed, as specified in [CDS+19], one of the first operations in the encryption stage consists in turning this byte array into a ternary coefficient polynomial, *via* the operation "unpack_S3".

During this process, every byte $(b_1, \cdots, b_8)$ of the string (256 possible values) is transformed into a tuple of 5 trigits $(c_1, \cdots, c_5) \in \{0, 1, 2\}^5$ (243 possible values), which implies a loss of information that prevents the recovery of the original plaintext during the decryption stage.

As NTRU - as specified in [CDS+19] - is designed to be only used as a KEM, this encoding issue is overcome by treating in advance the byte array given as input to the PKE, so that this string only encodes 243 values per byte instead of the 256 that could be possible[15]. Consequently, no information is lost in the encryption-then-decryption process. This *modus operandi* works well in the framework of a KEM because the original value of the plaintext itself does not matter, as long as a random shared secret can be derived from it. Indeed, in NTRU-HRSS KEM algorithm, the shared secret is a hash of the "treated" plaintext.

C.2 Our modified encoding

However, this solution is not relevant in the more general case of a PKE, in which any type of input of the good length has to be successfully recovered.

We therefore propose a slight variation of the encoding operation "unpack_S3" (and its reverse "pack_S3"), designed so that no information is lost during the encoding and thus the decryption can recover the arbitrary encrypted plaintext. Figure 12 details the changes brought by our proposal. In a nutshell, we encode the input bits not per byte - as in the original encoding - but in packs of 11 bits (2,048 possible values) that are turned into sets of 7 trigits (2,187 values). As the final set of values is bigger than the initial one, all possible values from the byte array can be encoded in the ternary coefficient polynomial.

In terms of efficiency, for NTRU-HRSS we have $n = 701$ and the input length is $|m + r| = 280$ bytes. We thus carry out $\gamma = 100$ times our modified encoding operation detailed in lines 5 to 7 of Figure 12, which permits to encode 137 bytes instead of the original 140 bytes for each part r and m, so 274 bytes instead of 280 in total. The efficiency loss induced by our technics is minor, while it brings a general scope to the PKE of NTRU-HRSS.

[14] As well as with NTRU-HPS, its counterpart from the same submission package to the NIST's PQC competition.

[15] This data treatment, carried out in an operation named "sample_rm", consists in sampling, both for m and r, a random ternary coefficient polynomial of degree $n - 2$. This one is then transformed into a byte array, *via* the operation "pack_S3".

Unpack_S3(B: byte array)

1 : $\gamma := \lceil (n-1)/5 \rceil$
2 : $(b_1, \cdots, b_{8.\gamma}) := Parse(B)$
3 : $\mathbf{v} := 0$
4 : $i := 0$
5 : **while** $i < \gamma$ **do** :
6 : set $(c_1, \cdots, c5) \in \{0,1,2\}^5$ *s.t.* :
$$\sum_{j=0}^{7} 2^j.b_{8i+1+j} = \sum_{j=0}^{4} 3^j.c_{1+j}$$
7 : $(v_{5i+1}, \cdots, v_{5i+5}) := (c_1, \cdots, c_5)$
8 : $i := i + 1$
9 : **return** $S3(\mathbf{v})$: polynomial

Modified_unpack_S3(B: byte array)

1 : $\gamma := \lceil (n-1)/7 \rceil$
2 : $(b_1, \cdots, b_{11.\gamma}) := Parse(B)$
3 : $\mathbf{v} := 0$
4 : $i := 0$
5 : **while** $i < \gamma$ **do** :
6 : set $(c_1, \cdots, c_7) \in \{0,1,2\}^7$ *s.t.* :
$$\sum_{j=0}^{10} 2^j.b_{11i+1+j} = \sum_{j=0}^{6} 3^j.c_{1+j}$$
7 : $(v_{7i+1}, \cdots, v_{7i+7}) := (c_1, \cdots, c_7)$
8 : $i := i + 1$
9 : **return** $S3(\mathbf{v})$: polynomial

Pack_S3(**v**: polynomial)

1 : $\gamma := \lceil (n-1)/5 \rceil$
2 : $\mathbf{v} := \underline{S3}(a)$
3 : $B = (b_1, \cdots, b_{8.\gamma}) := (0, \cdots, 0)$
4 : $i := 0$
5 : **while** $i < \gamma$ **do** :
6 : set $(c_1, \cdots, c_5) \in \{0,1,2\}^5$ *s.t.* :
$$c_j \equiv v_{5i+j} \pmod 3$$
7 : set $(b_{8i+1}, \cdots, b_{8i+8})$ *s.t.* :
$$\sum_{j=0}^{7} 2^j.b_{8i+1+j} = \sum_{j=0}^{4} 3^j.c_{1+j}$$
8 : $i := i + 1$
9 : **return** B: bytearray

Modified_pack_S3(**v**: polynomial)

1 : $\gamma := \lceil (n-1)/7 \rceil$
2 : $\mathbf{v} := \underline{S3}(a)$
3 : $B = (b_1, \cdots, b_{11.\gamma}) := (0, \cdots, 0)$
4 : $i := 0$
5 : **while** $i < \gamma$ **do** :
6 : set $(c_1, \cdots, c_7) \in \{0,1,2\}^7$ *s.t.* :
$$c_j \equiv v_{7i+j} \pmod 3$$
7 : set $(b_{11i+1}, \cdots, b_{11i+11})$ *s.t.* :
$$\sum_{j=0}^{10} 2^j.b_{11i+1+j} = \sum_{j=0}^{6} 3^j.c_{1+j}$$
8 : $i := i + 1$
9 : **return** B: bytearray

Fig. 12: Unpack_S3 and pack_S3 algorithms from the original NTRU-HRSS PKE ([CDS+19]) (left column), along with our modified versions (right column). In the encryption and decryption stages, these operations are performed both on the plaintext m and on the randomness r. $\underline{S3}(\mathbf{v})$ and $S3(\mathbf{v})$ respectively denote the canonical and non-canonical representatives of a polynomial $\mathbf{v}$ in the quotient-ring $S/3 = \mathbb{Z}[X]/(3, \mathbf{\Phi}_n)$.