Leaking-Cascade: an Optimal 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, should the post-quantum schemes used prove insecure.
Regarding the hybridization of Key Encapsulation Mechanisms (KEMs), most recent studies focus on safely combining the symmetric keys output by a parallel execution of classical and post-quantum KEMs. While this architecture is straightforward, it appears to lack computational efficiency and bandwidth optimization.
Hence, we propose a novel method for more effectively hybridizing several KEMs, by combining the underlying Public-Key Encryption schemes (PKEs) in an innovative variant of the cascade composition that we call "leaking-cascade", before turning the hybrid PKE into a KEM with a FO transformation. We prove that this architecture constitutes a robust combiner for encryption schemes up to IND-CPA security, which permits to eventually generate an IND-CCA2-secure KEM.
In terms of performance, our leaking-cascade scheme is at least as computationally efficient and has a better communication cost than the commonly used parallel combination, with a bandwidth gain of its ciphertext that may exceed 13 % compared to the latter. Moreover, we prove that for given PKEs that need to be hybridized, the leaking-cascade has an optimal ciphertext communication cost.

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 KEM 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 generally 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” [10] 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 [3]) 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 KEMs by simultaneously using 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 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 ($n > 2$) with one classical algorithm and two or more PQ KEMs solves this problem, but it suffers from important computational and bandwidth overheads 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 [11]) mixes the symmetric keys with the ciphertexts to produce a combined key.

[4] One-Way Chosen-Plaintext Attacks. This encryption security model, which aims to prevent the full recovery of a plaintext by an adversary having access to an encryption oracle, is weaker than models relying on “semantic security”, where the adversary tries to distinguish the ciphertexts associated to chosen plaintexts, with the help of an encryption oracle (IND-CPA) and potentially an adaptive decryption oracle (IND-CCA2).

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, in order to produce an hybrid PKE that a FO transformation will turn into an IND-CCA2 KEM. The main idea here is to save the computation cost of $(n-1)$ FO transformations, since this one must be applied only once, on the hybrid PKE. Furthermore, as this (intermediate) hybrid PKE only gets to be OW-CPA 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 proved by [11] 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 [16], which presents a generic construction where PKEs are run in parallel composition before the resulting hybrid PKE undergoes 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 secret-key encryption primitives has been far more studied, and notably the cascade composition (cf. section 2.4). The early works of [8] and [18] 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 the second cipher only.

At a higher level, [13] 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, [23] introduces the notion of “multi-encryption” and shows

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

[6] IND-RCCA [4] and IND-gCCA [1] 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”.

that the cascade scheme can reach IND-gCCA security[6] in their slightly different multi-encryption security model.

Our Contributions and Outline of this Paper. We present in section 3 of this paper a new way to combine encryption schemes, designed to keep the computational gains of the PKE parallel combination of [16] 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 2), which is a sufficient condition to build an IND-CCA2 KEM from the hybrid PKE, *via* a FO transformation.

We study in subsection 3.4 the functionalities offered by such an architecture, mainly the aforementioned KEM hybridization but also a lighter Authenticated Key Exchange (AKE) protocol that can be used, for instance, to implement the recent KEM-TLS proposal [21].

We then instantiate in section 4 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 [17], its variant NTRU' from [20] or BAT [9]) is added to the former double hybridization.

We also refer to some possible improvements that are further detailed in appendix D, including one, named "Integrated Diffie-Hellman (IDH) KEM", in which the first classical PKE in the cascade chain is replaced by a Diffie-Hellman key agreement scheme.

Finally, we prove in section 5 that the leaking-cascade architecture has an *optimal* ciphertext communication cost when it comes to PKE combination, and we compare in practice the computational and communication efficiencies of the instantiations from section 4.

2 Preliminaries

2.1 Notations and Terminology

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

"$.||.$" is used for the concatenation operation. "[.]" denotes optional values or parameters. $|.|$ represents the bit length of a given value. $\lfloor\ \rceil$ and $\lceil\ \rceil$ respectively denote the rounding and ceiling values of a decimal number.

In the context of lattice-based cryptography, 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 cryptosystems under the generic term of "encryption scheme", our leaking-cascade combiner being proved secure for both primitives. However, for practical purpose, we tend to use 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 Encryption Schemes

Secret-Key Encryption Scheme (SKE). A stateless probabilistic secret-key encryption scheme is a tuple of polynomial-time algorithms $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ such that:

- **KeyGen** takes as input the security parameter λ and randomly draws a (symmetric) secret key from the key space $\mathcal{K}$: $\boxed{k \leftarrow \mathrm{KeyGen}(1^\lambda)}$.
- **Enc** takes as input a secret key $k \in \mathcal{K}$ and a plaintext $m \in \mathcal{M}$ and probabilistically yields a ciphertext c: $\boxed{c \leftarrow \mathrm{Enc}(k, m)}$.
- **Dec** takes as input a secret key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and deterministically outputs a plaintext m: $\boxed{m := \mathrm{Dec}(k, c)}$.

If the encryption scheme is *correct*, we have:
$\forall m \in \mathcal{M}, \forall k \leftarrow \mathrm{KeyGen}(1^\lambda),\ \mathrm{Dec}(k, \mathrm{Enc}(k, m) = m.$

Public-Key Encryption Scheme (PKE). A probabilistic public-key encryption scheme is a tuple of polynomial-time algorithms $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ such that:

- **KeyGen** takes as input the security parameter λ and probabilistically outputs a couple of public and private keys: $\boxed{(pk, sk) \leftarrow \mathrm{KeyGen}(1^\lambda)}$.
- **Enc** takes as input a public-key $pk \in \mathcal{PK}$ and a plaintext $m \in \mathcal{M}$ and probabilistically yields a ciphertext c: $\boxed{c \leftarrow \mathrm{Enc}(pk, m)}$.
- **Dec** takes as input a secret-key $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$ and deterministically outputs a plaintext m: $\boxed{m := \mathrm{Dec}(sk, c)}$.

If the encryption scheme is *correct*, we have:
$\forall m \in \mathcal{M}, \forall (pk, sk) \leftarrow \mathrm{KeyGen}(1^\lambda),\ \mathrm{Dec}(sk, \mathrm{Enc}(pk, m) = m.$

IND-CPA Security of an Encryption Scheme. The security games corresponding to the IND-CPA security of the public-key and secret-key encryption schemes, with two experiments $\mathbf{Exp}^{\mathrm{IND-CPA-b}}$ ($b \in \{0, 1\}$), are detailed in Figure 1.

In the general case of an encryption scheme $\mathcal{E}$, the advantage of any Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in this model (where $\mathcal{A}$ wins the security experiment $\mathbf{Exp}^{\mathrm{IND-CPA-b}}$ if its guessing bit $\hat{b}$ equals b) is:

$$adv_{\mathcal{E}}^{IND-CPA}(\mathcal{A}) = \left| Pr[1 \leftarrow \mathcal{A}]_{\mathcal{E}}^{Exp\ IND-CPA-1} - Pr[1 \leftarrow \mathcal{A}]_{\mathcal{E}}^{Exp\ IND-CPA-0} \right|$$

```
Experiment $\mathbf{Exp}_{\mathrm{PKE}}^{\mathrm{IND-CPA-b}}$
1 : $(pk, sk) \leftarrow \mathrm{KeyGen}(1^\lambda)$
2 : $(m^0, m^1, st) \leftarrow \mathcal{A}_1(pk)$
3 : $c^* \leftarrow \mathrm{Enc}(pk, m^b)$
4 : $\hat{b} \leftarrow \mathcal{A}_2(pk, c^*, m^0, m^1, st)$
5 : return $\hat{b}$
```

```
Experiment $\mathbf{Exp}_{\mathrm{SKE}}^{\mathrm{IND-CPA-b}}$
1 : $k \leftarrow \mathrm{KeyGen}(1^\lambda)$
2 : $(m^0, m^1, st) \leftarrow \mathcal{A}_1^{\mathcal{O}^{\mathrm{Enc}}(.)}$
3 : $c^* \leftarrow \mathrm{Enc}(k, m^b)$
4 : $\hat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathrm{Enc}}(.)}(c^*, m^0, m^1, st)$
5 : return $\hat{b}$

Encryption oracle $\mathcal{O}^{\mathrm{Enc}}(m)$
1 : $c \leftarrow \mathrm{Enc}(k, m)$
2 : return $c$
```

Fig. 1: IND-CPA security games for public-key (left) and secret-key (right) encryption schemes.

2.3 Key Encapsulation Mechanisms (KEMs)

Description As stated above, the goal of a KEM is to perform a (symmetric) "key transport" functionality, usually 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 (KeyGen, Encaps, Decaps) defined as follows:

- **KeyGen** takes as input the security parameter λ and probabilistically outputs a couple of public and private keys: $\boxed{(pk, sk) \leftarrow \mathrm{KeyGen}(1^\lambda)}$.
- **Encaps** takes as input a public-key $pk \in \mathcal{PK}$. It internally randomly draws a symmetric key $k \overset{\$}{\leftarrow} \mathcal{K}$ and probabilistically encapsulates it within a ciphertext c: $\boxed{(k, c) \leftarrow \mathrm{Encaps}(pk)}$.
- **Decaps** takes as input a secret-key $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$ and deterministically outputs a symmetric key k: $\boxed{k := \mathrm{Decaps}(sk, c)}$.

If the KEM is *correct*, we have:
$\forall (pk, sk) \leftarrow \mathrm{KeyGen}(1^\lambda),\ \big(\mathrm{Decaps}(sk, c), c\big) \leftarrow \mathrm{Encaps}(pk)$.

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. Similarly to 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" [10]. Nowadays, the mostly used FO transformation is a variant by [14], which follows a modular approach and uses several subroutines that depend on 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 its randomness r from the input plaintext m instead of randomly drawing it: $\boxed{r := H(m \mid\mid [pk])}$, 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 re-encryption.
– The **“(Q)U$^{\not\perp}_{[\mathbf{m}]}$” transformation**: this operation, with several variants, derives the symmetric key from the random plaintext that was encapsulated (instead of straightly using this message as the key): $\boxed{k := G(m \mid\mid [c])}$, with G a cryptographic hash function.

Hence we basically have (see [14] for more details):

$$\begin{tikzcd} \overset{\text{OW/IND}-\text{CPA}}{\text{PPKE}} \arrow[r, "\text{T}"] & \overset{\text{OW}-\text{PCVA}}{\text{DPKE}} \arrow[r, "(\text{Q})\text{U}^{\not\perp}_{[\text{m}]}"] & \overset{\text{IND}-\text{CCA2}}{\text{KEM}} \end{tikzcd}$$

2.4 Combination of KEMs and Encryption Schemes

In hybrid cryptography, the goal of combining several primitives is to maintain a certain property if at least a determined number of these primitives fulfill this property themselves. The way they are mixed together is called a combiner, that has to be “robust” in the sense of the following definition from [13].

Definition 1 (Robust Combiner [13]). *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:*

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

Regarding the combination of KEMs or encryption schemes, we focus on $(1, n)$-robust combiners for a certain security level. We define hereunder (Definition 2) the case of a combiner for encryption schemes, upon which we rely in this paper.

Definition 2 (Robust Combiner for Encryption Schemes). *Given a security property “SEC”[8], we call “SEC robust combiner for encryption schemes” a $(1, n)$-robust combiner, in the sense of Definition 1, for which the combined*

[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).

[8] Such as one-wayness or indistinguishability against known or chosen plaintexts or ciphertexts.

cryptographic primitives are encryption schemes and the predicate is the security property "SEC".

In other terms, a SEC robust combiner for encryption schemes leads to an hybrid encryption scheme that yields the security property "SEC" if at least one of its component schemes is itself SEC-secure.

Nota: When all the combined encryption schemes are public-key cryptosystems, the related combiner is called a "PKE robust combiner".

We also define beneath the notion of efficiency for combiners, which aims to exclude from our analysis artificial combiners for encryption schemes which decrease their bandwidth at the cost of several calls to the decryption algorithm.

An instance of such combiners, applicable to any encryption scheme provided that the size of the encrypted plaintext is shorter than the input size of the encryption algorithm, consists of the following steps:

– At the encryption stage, the combiner pads the plaintext m with a known value *pad*.
– It then enhances the bandwidth of the hybrid scheme by truncating the original ciphertext by a number π of bits.
– During decryption, the protocol carries out an exhaustive search on the missing bits from the ciphertext and, for each guess, checks whether the padding value matches the one that was added before encrypting (*pad*).

This method allows a bandwidth gain of π bits, limited by the computational resources that the recipient is ready to dedicate to the decryption process of a single ciphertext (since it takes up to 2^π guesses to recover the plaintext). In any case, π is bounded by the security parameter λ ($\pi \ll \lambda$).

Definition 3 (Efficient Combiner). *A robust combiner is said to be (computationally) "efficient" if none of its component primitives runs more than once during each of the encryption and decryption processes.*

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

Parallel Composition. The common way to hybridize KEMs or encryption schemes is called the "parallel composition" (cf. [11] 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 $c := c_1 \mid\mid \cdots \mid\mid c_n$ 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 encryption scheme:

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

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 conceive, 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 proved a robust combiner – in the sense of Definition 2 – under certain assumptions, as detailed beneath.

Definition 4 (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:* $\boxed{c := \big((u_j)_{j \in L}, c_n\big)}$;
   - *v_j is given as input to the following encryption algorithm* Enc_{j+1}*:* $\boxed{c_{j+1} := \mathrm{Enc}_{j+1}(pk_{j+1}, v_j)}$.
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:* $\boxed{c_{i+1} := \mathrm{Enc}_{i+1}(pk_{i+1}, c_i)}$.

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 5 (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.*

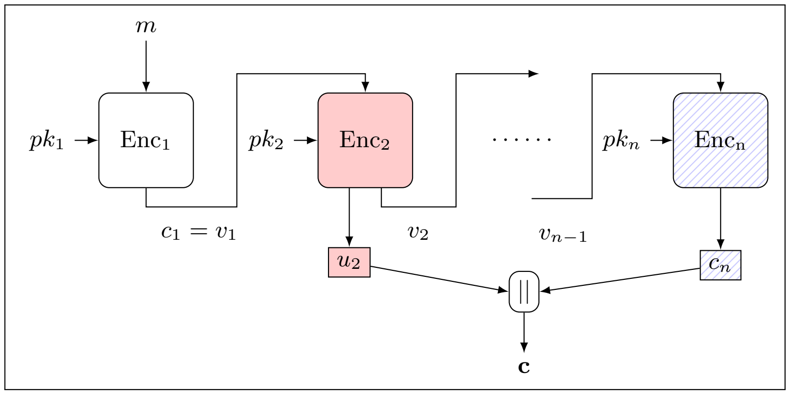

Fig. 2: Architecture of a $\{2\}$-leaking-cascade encryption combiner. In this instance, $\mathcal{E}_2$ is thus a leaking primitive which publicly displays a part of its ciphertext (called u_2), whereas the encryption algorithms $\mathcal{E}_1$ and $\mathcal{E}_3$ to $\mathcal{E}_n$ are combined in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

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 does not depend on the encrypted plaintext but rather (generally) 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 if and 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 6 (Partitioned-Ciphertext Encryption Scheme). *An encryption scheme $\mathcal{E} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ 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[9] such that:*

1. *The bit-lengths of c_r and c_m are constant over the message space $\mathcal{M}$;*
2. *c_r does not depend on 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.

[9] As in Definition 4, the relative order of c_r and c_m in the ciphertext c is not determined in the general case.

Leaking-cascade $\mathbf{Enc_{lc}}(pk, m, L)$

```
$(pk_1, \cdots, pk_n) := Parse(pk)$
$c := \varnothing$
$v_0 := m$
for $i \in [\![1, n-1]\!]$ do :
    if $i \in L$ then :
        $(u_i, v_i) \leftarrow \mathrm{Enc}_i(pk_i, v_{i-1})$
        $c := c \mid\mid u_i$
    else :
        $v_i \leftarrow \mathrm{Enc}_i(pk_i, v_{i-1})$
$c_n \leftarrow \mathrm{Enc}_n(pk_n, v_{n-1})$
$c := c \mid\mid c_n$
return $c$
```

Leaking-cascade $\mathbf{Dec_{lc}}(sk, c, L)$

```
$(sk_1, \cdots, sk_n) := Parse(sk)$
$((u_i)_{i \in L}, c_n) := Parse(c)$
$v_{n-1} := \mathrm{Dec}_n(sk_n, c_n)$
for $i \in [\![n-1, 1]\!]$ do :
    if $i \in L$ then :
        $c_i := (u_i, v_i)$
    else :
        $c_i := v_i$
    $v_{i-1} := \mathrm{Dec}_i(sk_i, c_i)$
$m := v_0$
return $m$
```

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 [16], is not detailed.

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[10] have this partitioned-ciphertext property: Kyber, BIKE and HQC.

We prove in appendix A the partitioned-ciphertext property of two of these algorithms (ElGamal, Kyber) that we use in our instantiations (cf. section 4).

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 robust encryption combiner for IND-CPA security, in the sense of Definition 2* [11]*, if and only if all leaking primitives are partitioned-ciphertext algorithms whose "message part" of the ciphertext (c_m) is encapsulated by the next encryption scheme in the chain (i.e. $c_{m_j} \subseteq v_j$, $\forall j \in L$).*

Nota : Breaking one of the leaking primitives in a leaking-cascade chain (due to a mathematical flaw or a sufficient computational power of an adversary) does not question at all the partitioned-ciphertext property of this encryption scheme. All information regarding the plaintext remains in the "message part" c_m of its ciphertext, which is encapsulated by the next encryption scheme in the leaking-cascade chain.

[10] 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).

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

Proof Sketch. Our security proof for 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 deduced from the latter by an induction argument that is detailed at the end of the section.

The proof for the 2-primitives leaking-cascade consists of three parts: the first two statements underneath correspond to the "sufficient" condition of Theorem 1, while the third statement deals with the "necessary" condition:

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. $c_{m_1} \subseteq v_1$, regardless of the security of $\mathcal{E}_1$.
3. The combination in leaking-cascade of two encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$ is an IND-CPA-unsecure combiner for encryption schemes if the upstream leaking primitive $\mathcal{E}_1$ is not partitioned-ciphertext or if the message part of its ciphertext is not entirely encapsulated by the downstream encryption scheme $\mathcal{E}_2$ ($c_{m_1} \not\subseteq v_1$).

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

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

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

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 ($c_1^* = (u^b, v^b) \leftarrow \mathcal{O}_{\mathcal{E}_1}^{\mathrm{Enc}}$), $\mathcal{B}_1$ computes $c^* \leftarrow (u^b \mid\mid \mathrm{Enc}_2(pk_2, v^b))$.
– $\mathcal{B}_2$ then runs the adversary $\mathcal{A}_2^{\mathcal{O}_{lc}^{\mathrm{Enc}}(.)}$ 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:
$\epsilon_1 = adv^{G0-G1}(\mathcal{A}) \leq adv_{\mathcal{E}_1}^{IND-CPA}(\mathcal{B})$.

Game 1 corresponds to the experiment $\mathbf{Exp}_{leak-casc}^{\mathrm{IND-CPA-1}}$ 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: $\boxed{adv_{leak-casc}^{IND-CPA} \leq adv_{\mathcal{E}_1}^{IND-CPA}}$.

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

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: $(\tilde{u}, \tilde{v}) \leftarrow \mathrm{Enc}_1(pk_1, \tilde{m})$, with $\tilde{m} \xleftarrow{\$} \mathcal{M}$. Thus, we have: $c \leftarrow u^0 \mid\mid \mathrm{Enc}_2(pk_2, \boxed{\tilde{v}})$.

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

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.** $(\mathcal{B}_1)_{\mathcal{E}_2}$
$(pk_1, sk_1) \leftarrow \mathrm{KeyGen}_1()$
$(m^0, \tilde{m}) \xleftarrow{\$} \mathcal{M}^2$
$(u^0, v^0) \leftarrow \mathrm{Enc}_1(pk_1, m^0)$
$(\tilde{u}, \tilde{v}) \leftarrow \mathrm{Enc}_1(pk_1, \tilde{m})$
$s \leftarrow \mathcal{A}_1^{\mathcal{O}_{G0-G1}^{\mathrm{Enc}}(.)}(u^0)$
**return** $((v^0, \tilde{v}), u^0, s)$
```

```
**Adv.** $(\mathcal{B}_2)_{\mathcal{E}_2}(c_2^*, u^0, s)$
$c^* := u^0 \mid\mid c_2^*$
$\hat{b} \leftarrow \mathcal{A}_2^{\mathcal{O}_{G0-G1}^{\mathrm{Enc}}(.,u^0)}(c^*)$
**return** $\hat{b}$
```

```
**Enc. oracle** $\mathcal{O}_{G0-G1}^{\mathrm{Enc}}(m, [u^0])$
**if** $u^0 = \varnothing$ **then** :
    $m^0 \xleftarrow{\$} \mathcal{M}$
    $(u^0, v^0) \leftarrow \mathrm{Enc}_1(pk_1, m^0)$
$(u, v) \leftarrow \mathrm{Enc}_1(pk_1, m)$
$c_2 \leftarrow \mathcal{O}_{\mathcal{E}_2}^{\mathrm{Enc}}(v)$
$c := u^0 \mid\mid c_2$
**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: $c \leftarrow \boxed{u^1} \mid\mid \mathrm{Enc}_2(pk_2, \tilde{v})$.

Neither u^0 nor u^1 is related in any way to the term $\mathrm{Enc}_2(pk_2, \tilde{v})$, 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 : $\epsilon_3 \leq adv_{\mathcal{E}_2}^{IND-CPA}$.

Game 3 corresponds to the experiment $\mathbf{Exp}_{leak-casc}^{\text{IND-CPA}-1}$ 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:
$\boxed{adv_{leak-casc}^{IND-CPA} \leq 2.adv_{\mathcal{E}_2}^{IND-CPA}}$.

Proof of Statement 3. We show here that if all information regarding the plaintext m in the first ciphertext c_1 is not entirely encapsulated by the downstream encryption scheme $\mathcal{E}_2$ and if the leaking upstream primitive $\mathcal{E}_1$ is not IND-CPA secure, then the leaking-cascade scheme is IND-CPA-unsecure as well, even if $\mathcal{E}_2$ is itself IND-CPA-secure. This includes both the case where $\mathcal{E}_1$ is not partitioned-ciphertext (which implies that the information related to m is scattered in the whole ciphertext c_1) and the case where at least a single bit from c_{m_1} is publicly leaked within u_1.

We consider the latter case that is more conservative, and assume that there exists, in the leaking part of $\mathcal{E}_1$, a bit that belongs to the message part c_{m_1}: $\exists b \in \{0,1\} : (b \in c_{m_1}) \wedge (b \in u_1)$.

For instance, let us define the upstream encryption scheme $\mathcal{E}_1$ as follows:

$$Enc_1 : \left| \begin{array}{l} \mathcal{PK} \times \mathcal{M} \longrightarrow \mathcal{C} = \mathcal{C}' \mid\mid \{0,1\} \\ \quad (pk, m) \longmapsto Enc'(pk, m) \mid\mid \sigma(m) \end{array} \right.$$

with $\sigma(m) = \sum_{i=0}^{|m|-1} b_i \ (mod\ 2)$ a checksum on all bits $b_i \in \{0,1\}$ of the plaintext m and $Enc' : \mathcal{PK} \times \mathcal{M} \rightarrow \mathcal{C}'$ the encryption algorithm of a partitioned-ciphertext IND-CPA-secure encryption scheme $\mathcal{E}'$.

Because $\mathcal{E}'$ has partitioned ciphertexts, any ciphertext c' it outputs consists of distinct random and message parts, and so is any ciphertext c_1 of the wrapping algorithm $\mathcal{E}_1$:

$$\forall (pk, m) \in \mathcal{PK} \times \mathcal{M},\ c' := Enc'(pk, m) = (c'_r, c'_m)$$
$$c_1 := c' \mid\mid \sigma(m) = \Big(c_{r_1} = c'_r, c_{m_1} = \big(c'_m, \sigma(m)\big)\Big)$$

We now study the combination in leaking-cascade of $\mathcal{E}_1$ (as the leaking upstream primitive) with an IND-CPA secure encryption scheme $\mathcal{E}_2$ (as the downstream primitive), such that the leaking part u_1 of $\mathcal{E}_1$'s ciphertext includes the bit $\sigma(m)$:

$$u_1 := c'_r \mid\mid \sigma(m) \quad \text{and} \quad v_1 := c'_m$$

Then, despite the IND-CPA security of $\mathcal{E}_2$ and the leakage of only one bit related to m, any PPT adversary $\mathcal{A}$ can easily win the IND-CPA security game of the hybrid scheme, described in Figure 1, by choosing challenge messages m^0 and m^1 such that $\sigma(m^0) = 0$ and $\sigma(m^1) = 1$, and by recovering $\sigma(m^*)$ in the challenge leaking-cascade ciphertext:

$$\begin{aligned} c^* :&= u_1^* \mid\mid c_2^* \\ &= c_r'^* \mid\mid \sigma(m^*) \mid\mid c_2^* \end{aligned}$$

Proof of the n-Hybridization in Leaking-Cascade. Let us consider n encryption schemes $(\mathcal{E}_i)_{i \in [\![1,n]\!]}$ such that all $n-1$ upstream primitives $(\mathcal{E}_i)_{i \in [\![1,n-1]\!]}$ are either non-leaking or with partitioned-ciphertext (and with the "message part" c_{m_i} of their ciphertext encapsulated by the next scheme $\mathcal{E}_{i+1}$). We want to prove that if at least one of these n primitives is IND-CPA-secure, then the combination of all of them in leaking-cascade is IND-CPA as well.

Let assume that a random scheme $\mathcal{E}_j$ ($j \in [\![1,n]\!]$) in the leaking-cascade chain is IND-CPA-secure.

Thus, relying on statement 1 above and on the work of [13] regarding the regular cascade, the combination in regular or leaking-cascade of $\mathcal{E}_j$ with the scheme $\mathcal{E}_{j+1}$ is an hybrid IND-CPA-secure primitive (that we call $\mathcal{E}^{hy}_{j,j+1}$), even if $\mathcal{E}_{j+1}$ itself appears unsecure. We similarly combine this new hybrid scheme $\mathcal{E}^{hy}_{j,j+1}$ with the following primitive $\mathcal{E}_{j+2}$, and so on, until the last scheme $\mathcal{E}_n$ of the chain.

We then use statement 2 and the results of [13] to combine the hybrid IND-CPA scheme $\mathcal{E}^{hy}_{j,n}$ with its upstream primitive $\mathcal{E}_{j-1}$. As long as the latter is either non-leaking or with partitioned-ciphertext (and with the "message part" $c_{m_{j-1}}$ of its ciphertext encapsulated by the scheme $\mathcal{E}_j$), its combination with $\mathcal{E}^{hy}_{j,n}$ also produces an IND-CPA hybrid scheme $\mathcal{E}^{hy}_{j-1,n}$. We carry out this combination step by step towards the first primitive $\mathcal{E}_1$ of the chain.

We now have a fully hybrid primitive $\mathcal{E}^{hy}_{1,n}$ yielding IND-CPA security, with only one encryption scheme underneath with that security level, which completes the proof of Theorem 1. $\square$

3.4 Functionalities of a Leaking-Cascade

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

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.

Authenticated Key Exchange with Leaking-Cascade. The leaking-cascade combination may also be used in the context of an authenticated key exchange, where the parties would be implicitly authenticated by additional KEMs instead of signatures. This approach, stated for instance by [3], appears particularly interesting in the framework of post-quantum authentication, as most current PQ signatures still suffer from a larger bandwidth than their KEM counterparts. This observation gave rise to KEM-TLS [21], a variant of TLS where signatures are replaced by a KEM-based implicit authentication. KEM-TLS however uses a classical parallel combination of its KEMs, and therefore could be improved with the use of our leaking-cascade.

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

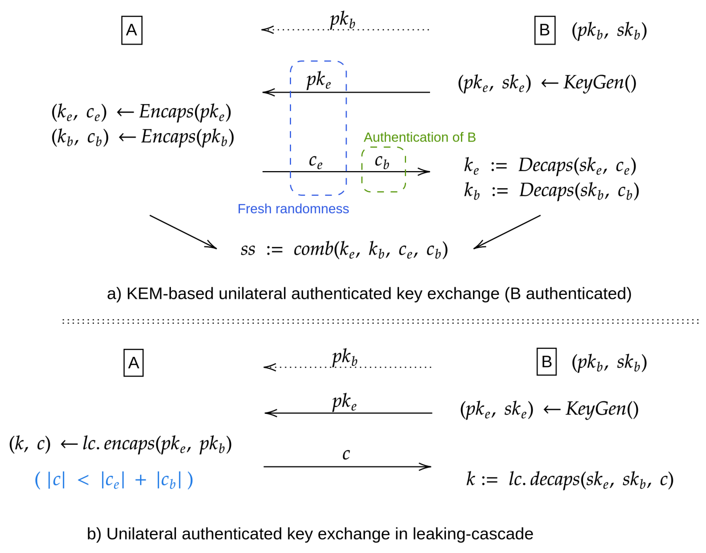
A
pk_b
B
(pk_b, sk_b)
pk_e
$(pk_e, sk_e) \leftarrow KeyGen()$
$(k_e, c_e) \leftarrow Encaps(pk_e)$
$(k_b, c_b) \leftarrow Encaps(pk_b)$
Authentication of B
c_e
c_b
$k_e := Decaps(sk_e, c_e)$
$k_b := Decaps(sk_b, c_b)$
Fresh randomness
$ss := comb(k_e, k_b, c_e, c_b)$
a) KEM-based unilateral authenticated key exchange (B authenticated)
A
pk_b
B
(pk_b, sk_b)
pk_e
$(pk_e, sk_e) \leftarrow KeyGen()$
$(k, c) \leftarrow lc.encaps(pk_e, pk_b)$
c
$(|c| < |c_e| + |c_b|)$
$k := lc.decaps(sk_e, sk_b, c)$
b) Unilateral authenticated key exchange in leaking-cascade

Fig. 5: KEM-based implicit UAKE, with authentication of party B. Figure *a* shows a standard authentication procedure, as proposed in [3]. 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.

In this paradigm, one can rely on the same algorithm for the whole leaking-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).

4 Instantiations

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

4.1 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, according to Theorem 1, the resulting hybrid PKE has IND-CPA security and the related KEM, after applying the FO transformation, is IND-CCA2 secure.

4.2 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.

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 A).

Regarding the downstream PQ algorithm, we consider two candidates both based on NTRU lattices: NTRU-HRSS [17] and the more recent algorithm BAT [9]. We also implement a variant of NTRU-HRSS proposed by [20], that we call NTRU'. This one is based on a tweak by [19] 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). This tweak therefore increases the input size of this PKE, which is beneficial for the efficiency of the leaking-cascade.

The criteria considered for these choices 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.

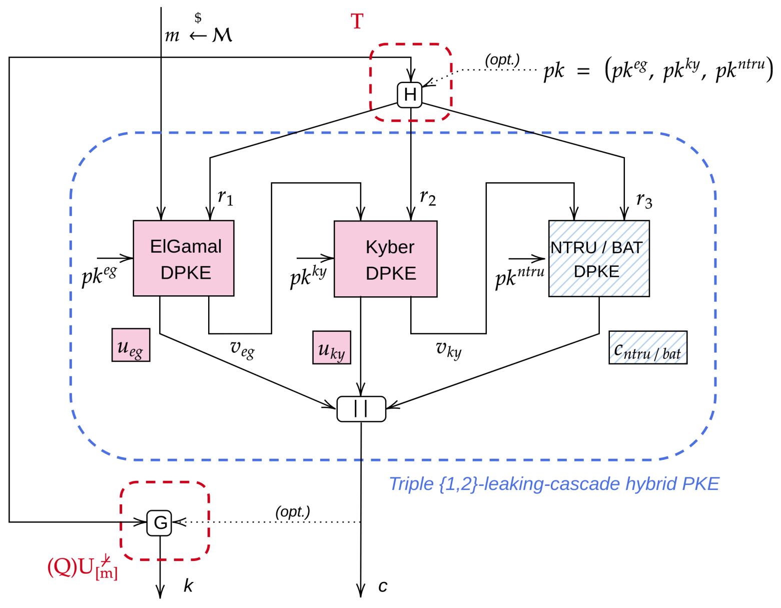

Fig. 6: Architecture of the triple hybridization in {1,2}-leaking-cascade ElGamal-Kyber-NTRU/BAT. The leaking PKEs, here both ElGamal and Kyber, are depicted in pink color. The dashed blue box represents the hybrid PKE and the dashed red boxes, the modular FO transformation.

– More importantly, the input lengths[12] of their PKEs, of 140 bytes for NTRU-HRSS, 160 bytes for BAT and 280 bytes for NTRU', 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 B.

4.3 Enhanced Variations

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

Integrated Diffie-Hellman (IDH) KEM. The IDH KEM is a leaking-cascade combination of one or several PQ PKEs with a Diffie-Hellman on Elliptic Curve (ECDH) key agreement scheme (in replacement of the classical encryption scheme).

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

The idea here is to be more efficient in terms of bandwidth, given the fact that an ECDH key agreement scheme produces a ciphertext[13] 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.

5 Performances

The leaking-cascade architecture was designed to improve the communication cost of the parallel composition, for a computational cost at least as good as the latter.

The results of our experiments show that these goals were achieved, with a bandwidth reduced over 10% – depending on the algorithms used – and a computational cost similar, and even slightly better for our leaking-cascade scheme.

5.1 Computational Cost

The leaking-cascade combiner avoids the computations of the key combination and of all but one FO transformations that are necessary with a parallel combiner.

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 combination	Leaking-cascade combination
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[14] for both parallel and leaking-cascade

[13] We model here the "ciphertext" of an ECDH key agreement scheme by the public DH element output by the sender. See appendix D for additional details.

[14] The key generation stage was not evaluated, as it is similar in both schemes.

schemes. It indeed confirms a better computational performance of the leaking-cascade scheme, mainly due to a lesser number of hashing operations resulting to the single FO transformation.

This minor 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 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.

5.2 Communication Cost

The main advantage of a PKE hybridization in leaking-cascade is the improved bandwidth of its ciphertext. The extent of this gain relies on the characteristics of the encryption schemes used in the chain, including the lengths of their inputs and ciphertext "message parts". However, we state that for given PKEs that must be hybridized, the leaking-cascade combination is the *optimal* hybridization method regarding the communication cost of its ciphertext.

We define below the concept of encryption scheme with "optimized ciphertext" that aims to exclude, in our Theorem 2 and in the related proof, artificial encryption schemes whose ciphertexts would natively include redundancy, e.g. to compensate some communication loss. In practice, to the best of our knowledge, all real-life cryptosystems have optimized ciphertexts, while potential redundancy mechanisms are implemented at a higher level in the communication protocol.

Definition 7 (Encryption Scheme with Optimized Ciphertext). *An encryption scheme $\mathcal{E}$, whose encryption algorithm* Enc *takes as input a plaintext m and yields a ciphertext c of size $|c|_\lambda$ (depending on the set of parameters) is said to have an "optimized ciphertext" if the size of its ciphertext is the minimum necessary to legitimately recover the encrypted plaintext m with a single call to the decryption algorithm* Dec*.*

Theorem 2 (Optimal Communication Cost). *Let* PKE_1 *and* PKE_2 *be two public-key cryptosystems with optimized ciphertexts, such that:*
- PKE_1 *is partitioned-ciphertext ;*
- *the message part c_{m_1} of* PKE_1*'s ciphertext is smaller than or the same size as the input of* PKE_2 *: $|c_{m_1}| \leq |m_2|$.*

Then, the combination of these two PKEs in leaking-cascade, with PKE_1 *as the leaking upstream primitive and $c_{m_1} \subseteq m_2$, constitutes the* ***optimal*** *efficient IND-CPA robust PKE combiner in terms of ciphertext bandwidth. In other terms, there exists no hybridization method of these PKEs yielding IND-CPA security, running once each underlying primitive and outputting a ciphertext shorter than with the leaking-cascade.*

Nota: A regular cascade is a particular case of leaking-cascade, occurring when $|c_1| \leq |m_2|$. When this condition is fulfilled, it is therefore the optimal combiner in terms of ciphertext bandwidth.

Proof of Theorem 2. Our proof starts by computing the communication efficiency of the leaking-cascade combination in the general case of *encryption schemes*, which implies the concept of "expansion rate" formalized below.

We then restrict the scope of our demonstration to the *efficient* combination of *PKEs with optimized ciphertexts* for an aimed IND-CPA security, and we show the ciphertext bandwidth optimality of the leaking-cascade with a proof by contradiction, by considering an hypothetical PKE combiner $\mathscr{C}$ outputting a shorter ciphertext than the leaking-cascade.

Definition 8 (Expansion Rate of an Encryption Scheme). *Let us consider an encryption scheme $\mathcal{E}$ that, for a given set of parameters depending on the security parameter λ, takes as input plaintexts of bit-length $|m_{\mathcal{E}}|_\lambda$ and outputs ciphertexts of bit-length $|c_{\mathcal{E}}|_\lambda$.*

We define the expansion rate of $\mathcal{E}$ for the security parameter λ as the ratio $\boxed{\tau_{\mathcal{E},\lambda} = \frac{|c_{\mathcal{E}}|_\lambda}{|m_{\mathcal{E}}|_\lambda}}$.

Nota: When the security parameter λ is well-defined and remains unchanged in a given context, it may be omitted in the notation of the expansion rate $\tau_{\mathcal{E}}$ and the input and output sizes $|m_{\mathcal{E}}|$ and $|c_{\mathcal{E}}|$, which are then considered as constant values. This is the case in this paper.

Communication Cost of the Leaking-Cascade. Given two encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$ with respective input and output sizes $|m_1|$, $|m_2|$, $|c_1|$, $|c_2|$, the ciphertext bandwidth $|c|_{lc}$ of their leaking-cascade combination is computed as follows.

As in Definition 4, we note u_1 the public part of $\mathcal{E}_1$'s ciphertext and v_1 the part of its ciphertext that is encapsulated by $\mathcal{E}_2$.

$$\begin{aligned} |c|_{lc} &= |u_1| + |c_2| \\ &= (|c_1| - |v_1|) + \tau_2.|m_2|, \qquad \text{with } |v_1| = |m_2| \\ \Leftrightarrow \boxed{|c|_{lc}} & \boxed{= \tau_1.|m_1| + (\tau_2 - 1).|m_2|} \end{aligned}$$

Proof by Contradiction. Let us assume that, for given public-key cryptosystems PKE$_1$ and PKE$_2$, with respective expansion rates τ_1 and τ_2, input sizes $|m_1|$ and $|m_2|$ and ciphertext sizes $|c_1|$ and $|c_2|$, there exists an efficient IND-CPA robust PKE combiner $\mathscr{C}$ (in the sense of Definition 2 and Definition 3) that has a better ciphertext communication cost than the leaking-cascade. Consequently, its ciphertext bandwidth is bounded by that of the leaking-cascade:

$$\begin{aligned} |c|_{\mathscr{C}} &< |c|_{lc} \\ &< \tau_1.|m_1| + (\tau_2 - 1).|m_2| \end{aligned}$$

We now wonder whether the data comprised in $\mathscr{C}$'s ciphertext bandwidth are sufficient to recover the original plaintext m that was encrypted by $\mathscr{C}$.

A robust PKE combiner must clearly make use of all its underlying schemes (in this case, PKE_1 and PKE_2) in order to benefit from the best security of them. Furthermore, the decryption process of a PKE combiner only uses its component PKEs' *decryption* algorithms[15]. Consequently, the original plaintext m encrypted by the combiner $\mathscr{C}$ can be recovered only if both decryptions algorithms Dec_1 and Dec_2 are run successfully.

Without loss of generality, let us analyze sequentially the decryption process of the combiner $\mathscr{C}$ by considering first the decryption algorithm Dec_2. As PKE_2 has an optimized ciphertext, Dec_2 takes as input a part of the bandwidth whose size cannot fall below $|c_2| = \tau_2.|m_2|$. Then, it yields an output of size $|m_2|$. Consequently, the data from the ciphertext bandwidth that remain available for the decryption by Dec_1 have a size bounded by:

$$\begin{aligned}
|\text{Dec}_1.input|_{\mathscr{C}} &\leq |c|_{\mathscr{C}} - |c_2| + |m_2| \\
&< |c|_{lc} - |c_2| + |m_2| \\
&< \tau_1.|m_1| + (\tau_2 - 1).|m_2| - \tau_2.|m_2| + |m_2| \\
&< |c_1|
\end{aligned}$$

Because PKE_1 also has an optimized ciphertext, its decryption algorithm Dec_1 needs an input size of at least $|c_1|$. The data available for Dec_1 are therefore too small to successfully retrieve the plaintext m_1 that was encrypted with Enc_1.

Thus, as we need both decryption algorithms to work successfully, as stated above, the failure of Dec_1 makes impossible to recover the original plaintext m that steams from m_1 and m_2 and the combiner $\mathscr{C}$ turns out to be non-functional.

This indicates that the leaking-cascade is an optimal efficient IND-CPA hybridization method for PKEs with optimized ciphertexts, which terminates the proof of Theorem 2. $\square$

Communication Cost in a Degraded Configuration. We also show in appendix C that even when the features of the algorithms PKE_1 and PKE_2 imply conditions not conducive to the implementation of the leaking-cascade (namely, when the message part $|c_{m_1}|$ of PKE_1 is bigger than the input size $|m_2|$ of PKE_2), this combination remains more efficient than the classical parallel composition, within a certain range of parameters.

[15] Indeed, if the decryption process of a PKE combiner used the *encryption* algorithm (instead of the decryption one) of one or several component PKEs, then anyone could legitimately run these primitives with the related public encryption keys. These ones would therefore bring no security to the hybrid scheme. This is a major difference with a combiner for secret-key cryptosystems, that may only use the encryption algorithms of its component schemes, even for the decryption process. This restriction specific to PKE combiners is needed in the proof of Theorem 2, which explains why the latter only shows the optimality of the leaking-cascade as a PKE combiner.

Practical Efficiency. 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.

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 %
Unilateral Authenticated Key Echange			
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 ciphertext **communication costs** of parallel and leaking-cascade combinations for different types of hybridizations and KEM-based authentications relying on ElGamal, Kyber, NTRU, NTRU' and BAT PKEs.

Even though its bandwidth 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

1. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (Apr / May 2002).
2. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (Apr 2006).
3. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634 (2017),
4. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (Aug 2003).
5. Chen, C., Danba, O., Stein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: Ntru algorithm specifications and supporting documentation (2019)

6. Dodis, Y., Katz, J.: Chosen-ciphertext security of multiple encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 188–209. Springer, Heidelberg (Feb 2005).
7. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) Advances in Cryptology. pp. 10–18. Springer Berlin Heidelberg, Berlin, Heidelberg (1985)
8. Even, S., Goldreich, O.: On the power of cascade ciphers. ACM Tras. Computer Systems, 3:108–116 (1985)
9. Fouque, P.A., Kirchner, P., Pornin, T., Yu, Y.: BAT: Small and fast KEM over NTRU lattices. Cryptology ePrint Archive, Report 2022/031 (2022),
10. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999).
11. Giacon, F., Heuer, F., Poettering, B.: KEM combiners. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 190–218. Springer, Heidelberg (Mar 2018).
12. Goldreich, O., Lustig, Y., Naor, M.: On chosen ciphertext security of multiple encryptions. Cryptology ePrint Archive, Report 2002/089 (2002),
13. Herzberg, A.: Folklore, practice and theory of robust combiners. Cryptology ePrint Archive, Report 2002/135 (2002),
14. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive, Report 2017/604 (2017),
15. Hohenberger, S., Lewko, A.B., Waters, B.: Detecting dangerous queries: A new approach for chosen ciphertext security. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 663–681. Springer, Heidelberg (Apr 2012).
16. Huguenin-Dumittan, L., Vaudenay, S.: FO-like combiners and hybrid post-quantum cryptography. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 21. LNCS, vol. 13099, pp. 225–244. Springer, Heidelberg (Dec 2021).
17. Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P.: High-speed key encapsulation from NTRU. Cryptology ePrint Archive, Report 2017/667 (2017),
18. Maurer, U.M., Massey, J.L.: Cascade ciphers: The importance of being first. Journal of Cryptology **6**, 55–61 (1993),
19. Pipher, J.H.J., Silverman, J.H.: Ntru: A new high speed public-key cryptosystem. In: draft from at CRYPTO 96 rump session (1996),
20. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 520–551. Springer, Heidelberg (Apr / May 2018).
21. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum TLS without handshake signatures. Cryptology ePrint Archive, Report 2020/534 (2020),
22. Zhang, C., Cash, D., Wang, X., Yu, X., Chow, S.S.M.: Combiners for chosen-ciphertext security. In: Dinh, T.N., Thai, M.T. (eds.) Computing and Combinatorics. pp. 257–268. Springer International Publishing, Cham (2016)
23. Zhang, R., Hanaoka, G., Shikata, J., Imai, H.: On the security of multiple encryption or CCA-security+CCA-security=CCA-security? In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 360–374. Springer, Heidelberg (Mar 2004).

A 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 4.

A.1 ElGamal

Theorem 3. *ElGamal public-key encryption algorithm [7] 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, $c_{r_{eg}}$ does not depend on the encrypted plaintext m but only on the randomness r, which completes the proof.

A.2 Kyber

Theorem 4. *Crystals Kyber public-key encryption algorithm [3] is a partitioned-ciphertext encryption scheme.*

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

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

– The modulus $q = 3329$ is a fixed public parameter;
– $\mathbf{t}$ and $A = \text{Expand}(\rho)$ are issued from the recipient's public-key $\boxed{pk := \mathbf{t} \mid\mid \rho}$ (with $\rho \xleftarrow{\$} \{0,1\}^{256}$);
– $\mathbf{e_1}$, 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. [13]), 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 $c_{r_{ky}}$ does not depend on the plaintext m: it is a function of the recipient's public key (since the matrix A directly comes from the random seed $\rho \in pk$) and of the random seed r used to deterministically produce $\mathbf{r}$, $\mathbf{e_1}$ and e_2. It thus fulfills condition 2 of Definition 6 and implies that $|c_{r_{ky}}|$ is constant over $\mathcal{M}$. According to the previous paragraph, we can deduce that $|c_{m_{ky}}|$ is constant over $\mathcal{M}$ as well, which terminates the proof of the theorem.

B NTRU-HRSS Encoding Variant for the Leaking-Cascade

B.1 Original Input Encoding of the NTRU-HRSS PKE

As indicated in section 4, the PKE of NTRU-HRSS[16] cannot recover any arbitrary input byte array. Indeed, as specified in [5], 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 [5] – 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[17]. 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.

B.2 Our Modified Encoding

However, this solution is not relevant in the more general case of a PKE, in which any 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 encrypted plaintext. Figure 9 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. With our tweak, we carry out $\gamma = 100$ times our modified encoding operation detailed in lines 5 to 7 of Figure 9, 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, whereas it brings a general scope to the PKE of NTRU-HRSS.

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

[17] 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($\mathbf{v}$: 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($\mathbf{v}$: 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. 9: Unpack_S3 and pack_S3 algorithms from the original NTRU-HRSS PKE [5] (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 $S/3 = \mathbb{Z}[X]/(3, \mathbf{\Phi}_n)$.

C Communication Cost of the Leaking-Cascade in a Degraded Configuration

We study the hybridization – as an IND-CPA robust combiner – of two encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$ such that $\mathcal{E}_1$ is "partitioned-ciphertext". For a given set of parameters, corresponding to an aimed security level, the input and output sizes of these algorithms ($|m_1|$, $|m_2|$, $|c_1|$ and $|c_2|$) are fixed.

Definition 9 (Leaking-Cascade in a Degraded Configuration). *We say that the combination in leaking-cascade of two encryption schemes $\mathcal{E}_1$ and $\mathcal{E}_2$, with $\mathcal{E}_1$ as a "partitioned-ciphertext" algorithm, occurs in a degraded configuration if the message part of the upstream primitive $\mathcal{E}_1$ is bigger than the input size of the downstream scheme $\mathcal{E}_2$: $|c_{m_1}| > |m_2|$.*

In this unfavorable case, the IND-CPA security of the leaking-cascade hybrid scheme can be ensured only if the downstream encryption scheme $\mathcal{E}_2$ runs $\alpha > 1$ times to encrypt the whole message part c_{m_1} of $\mathcal{E}_1$, with α defined as: $\alpha = \left\lceil \frac{|c_{m_1}|}{|m_2|} \right\rceil$.

In this context, the ciphertext communication cost $|c|_{lc_\alpha}$ of the leaking-cascade becomes:

$$\begin{aligned} |c|_{lc_\alpha} &= |u| + \alpha.|c_2| \\ &= (|c_1| - \alpha.|m_2|) + \alpha.|c_2| \\ \Leftrightarrow \quad & \boxed{|c|_{lc_\alpha} = \tau_1.|m_1| + \alpha.(\tau_2 - 1).|m_2|} \qquad \text{with } \alpha > 1 \end{aligned}$$

The parallel combination, where the ciphertexts of the component encryption schemes are simply concatenated together, has therefore the following ciphertext communication cost:

$$|c|_{//} = \tau_1.|m_1| + \tau_2.|m_2|$$

We can rewrite the efficiency of the leaking-cascade as:

$$\begin{aligned} |c|_{lc_\alpha} &= \tau_1.|m_1| + \alpha.(\tau_2 - 1).|m_2| \\ &= |c|_{//} + ((\alpha - 1).\tau_2 - \alpha).|m_2| \end{aligned}$$

Thus, for the leaking-cascade in this degraded mode (lc_α) to be more or equally efficient than its parallel counterpart, we need that :

$$(\alpha - 1).\tau_2 - \alpha < 0 \quad \Leftrightarrow \quad \boxed{\tau_2 < \frac{\alpha}{\alpha - 1}}$$

– $\underline{\tau_2 = 1}$: In the very particular case where the downstream encryption scheme $\mathcal{E}_2$ has a unitary expansion rate, this condition is fulfilled for all possible values of α and the leaking-cascade yields a better efficiency than the parallel combination.

– $\underline{\tau_2 > 1}$: The leaking-cascade is more efficient than the parallel composition only under some values of τ_2, depending on α. For instance, the following conditions must be met for leaking-cascade to be more efficient than the parallel combination:

$$\alpha = 2 \Rightarrow \tau_2 < 2$$
$$\alpha = 3 \Rightarrow \tau_2 < \frac{3}{2}$$

D Enhancements of the Cascade Combiner

D.1 Integrated Diffie-Hellman (IDH) KEM

As stated in subsection 4.3, 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 Curve (ECDH).

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

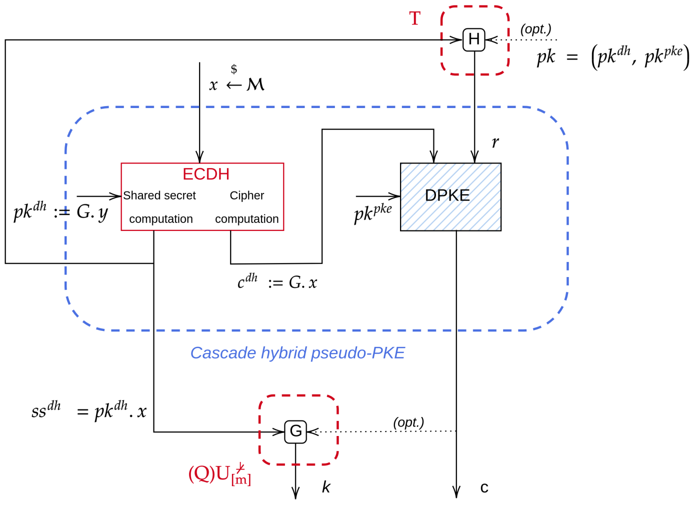

Fig. 10: Encapsulation step of an Integrated Diffie-Hellman KEM. The decapsulator's public DH element $G.y$ is seen as their ECDH public-key pk^{dh} while the encapsulator's public ECDH element is modeled as their ciphertext $c^{dh} := G.x$.

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 in order to demonstrate that the hybrid

```
IDH_KEM.Encaps_A(pk_B)
1 :  $(pk_B^{pq}, pk_B^{dh}) := Parse(pk_B)$
2 :  $x \xleftarrow{\$} \{0,1\}^n$
3 :  $ss^{dh} := H(x.pk_B^{dh})$
     FO transformation T :
4 :  $r := H(ss^{dh}\ [\ ||\ H(pk_B)])$
     Cascade PKE encryption :
5 :  $c^{dh} := X = G.x$
6 :  $c := PQ.PKE.\mathrm{Enc}(pk_B^{pq}, X, r)$
     FO transformation $(\mathrm{Q})\mathrm{U}^{\not\perp}_{[\mathrm{m}]}$:
7 :  $k := G(ss^{dh}\ [\ ||\ H(c)])$
8 :  return $(k, c)$
```

```
IDH_KEM.Decaps_B(sk_B, c, pk_B)
1 :  $sk_B^{pq}, sk_B^{dh} := Parse(sk_B)$
     Cascade PKE decryption :
2 :  $X' := PQ.PKE.\mathrm{Dec}(sk_B^{pq}, c)$
3 :  $ss^{dh'} := H(X'.sk_B^{dh})$
     FO transformation T :
4 :  $r' := H(ss^{dh'}\ [\ ||\ H(pk_B)])$
     FO transformation $(\mathrm{Q})\mathrm{U}^{\not\perp}_{[\mathrm{m}]}$:
5 :  $c' := PQ.PKE.\mathrm{Enc}(pk_B^{pq}, X', r')$
6 :  if $c' = c$ then :
7 :      $k := G(ss^{dh'}\ [\ ||\ H(c)])$
8 :  else :
9 :      $z \leftarrow \{0,1\}^n$
10 :     $k := G(z\ [\ ||\ H(c)])$
11 : return $k$
```

Fig. 11: 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.

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 as an ElGamal public-key, and thus can be considered as their ECDH public-key pk_B^{dh};
- encapsulating the sender's public DH element $G.x$ in the downstream PKE has the exact same effect as encapsulating a "real" PKE ciphertext, in the sense that an adversary needs 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 as it does upon a standard PKE. The main difference here with a standard KEM construction from a PKE is that the transformation $(\mathrm{Q})\mathrm{U}^{\not\perp}_{[m]}$ is not applied directly on the random input seed x but rather on the shared secret ss^{dh} derived from it. As this operation merely consists in hashing the seed, it makes no 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 $(\mathrm{Q})\mathrm{U}^{\not\perp}_{[m]}$ is applied not on the seed m itself but rather 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 therefore necessary to select ECDH algorithms that produce public elements and shared secret of that sizes. X25519, the key-agreement scheme based on Curve25519 (cf. [2]), appears to be an excellent candidate in that matter, with its 32 byte-long public element, 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 string of 256 bits.

D.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: $\boxed{pk^{ky} := \mathbf{t}^{ky} \mid\mid \rho}$, with $\mathbf{t}^{ky} := A.\mathbf{s} + \mathbf{e}$.

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

```
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 :
4 :  $\rho := H(pk^{dh})$   $\cancel{\rho \xleftarrow{\$} \{0,1\}^{256}}$
5 :  $\sigma \xleftarrow{\$} \{0,1\}^{256}$
6 :  $A := \mathrm{Expand}(\rho)$
7 :  $(\mathbf{s}^{ky}, \mathbf{e}) := \mathrm{Expand}(\sigma)$
8 :  $\mathbf{t}^{ky} := A.\mathbf{s}^{ky} + \mathbf{e}$
9 :  return $pk := (\mathbf{t}^{ky}, pk^{dh}), sk := (\mathbf{s}^{ky}, sk^{dh})$
```

Fig. 12: 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.

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