How Multi-Recipient KEMs can help the Deployment of Post-Quantum Cryptography

Joël Alwen[1], Matthew Campagna[1], Dominik Hartmann[2], Shuichi Katsumata[3,4], Eike Kiltz[2], Jake Massimo[1], Marta Mularczyk[1], Guillermo Pascual-Perez[5], Thomas Prest[3], and Peter Schwabe[6]

[1]Amazon Web Services
[2]Ruhr University Bochum
[3]PQShield
[4]AIST
[5]Institute of Science and Technology Austria (ISTA)
[6]Max Planck Institute for Security and Privacy, Radboud University Nijmegen

Abstract. The main purpose of this work is to raise awareness about a primitive that can provide large efficiency gains in post-quantum cryptography: multi-recipient KEMs, or mKEMs. In a nutshell, when encapsulating a key to N parties, an mKEM generates a single ciphertext that can be decapsulated by all parties. The size of an mKEM ciphertext can be significantly smaller than the sum of the sizes of N KEM ciphertexts. Moreover, individual receivers only need a small part of the mKEM ciphertext to run decapsulation.
We then propose mKyber, a very compact mKEM based on Kyber. Asymptotically, the size of an mKyber multi-recipient ciphertext is 16 times smaller than the sum of the sizes of N Kyber ciphertexts. The algorithmic description and parameters of mKyber and Kyber are very similar, which facilitates the re-use of existing security analyses, implementations, and formal verification tools that have been developed for Kyber.
Finally, we showcase some selected applications. mKEMs can be used to greatly reduce the bandwidth cost of the group key agreement protocol underlying the Message Layer Security (MLS) secure group messaging standard. Reducing bandwidth is one of the primary design considerations for MLS. More fundamentally, mKEMs reduce the cost of broadcasting private information to groups of recipients (e.g. a fleet of Cloud Hardware Security Modules).

1 Introduction

In July, 2022, NIST selected their first post-quantum standards for key agreement and (stateless) signature: the key encapsulation mechanism (KEM) Kyber [SAB+22], and the signature schemes Dilithium [LDK+22], SPHINCS$^+$ [HBD+22] and Falcon [PFH+22]. While this will greatly speed up the transition of existing systems to post-quantum cryptography (PQC), some challenges will still need to be addressed in the process.
One of the main challenges in this transition process is the overhead in communication cost. For 128 bits of classical security, the size of an ECDH public key is 32 bytes, whereas the size of a Kyber ciphertext is 768 bytes, which is 24 times larger. This means that protocols that make an extensive use of key exchange or key encapsulation will require more bandwidth when migrating to PQC; such protocols include the IETF standard MLS [BBR+23], or broadcast protocols. This additional cost may require to scale up the bandwidth capabilities of the systems deploying these protocols, a cost that not all end users will be able to shoulder.

Multi-recipient KEMs (mKEMs). Fortunately, a simple primitive can help to address some of the scalability challenges faced by PQC: multi-recipient KEMs, or mKEMs. When encapsulating one key K to N distinct encapsulation keys $(\mathsf{ek}_i)_{i\in[N]}$, the straightforward solution is to send N distinct ciphertexts ct_i – one for each ek_i. With a mKEM, one

may instead generate a single ciphertext $\vec{\mathsf{ct}}$ that can be decrypted by all recipients. For a formal definition, see Section 2.
What makes mKEMs appealing for PQC is the massive efficiency gains they can provide. Recent works [KKPP20, HKP+21, AHK+23] have shown that lattice-based mKEMs can be asymptotically (in N) one or two orders of magnitude more compact than the use of N KEMs in parallel.

mKyber: a Kyber-based mKEM. We demonstrate the potential of mKEMs by proposing mKyber, a mKEM construction based on Kyber. When encrypting the same message to a large number of recipients, mKyber can be up to 16 times more compact than Kyber, with an amortized cost of 48 bytes per recipient compared to 768 bytes per recipient for Kyber. The algorithmic description and parameters of mKyber are very similar to those of Kyber. From an implementation perspective, it means that existing implementations of Kyber can be easily repurposed to implement mKyber, even in the case of highly optimized and/or platform-specific implementations. From a security evaluation perspective, security proof techniques and formal verification tools can be adapted. See Section 3 for more details.

amKyber: an even more secure mKEM. mKyber already enjoys the standard notion of IND-CCA security. However, in some settings a stronger notion of IND-CCA security *with adaptive corruptions* may be desired. Roughly, the latter requires IND-CCA security against an attacker that can adaptively leak mKEM secret keys. This is particularly important in applications using long-term mKEM key pairs. We therefore present amKyber, adapted from [AHK+23], which satisfies the stronger security notion. The amKyber public keys are only 256 bits larger than those of mKyber and the ciphertexts are only twice larger, i.e. 8 times more compact than for Kyber. The algorithms of amKyber are built by adapting in a simple way the building blocks of mKyber: the IND-CPA secure encryption scheme and the FO transform. Therefore, amKyber is easy to implement given an mKyber implementation and security proofs and formal verification tools for mKyber can be adapted.

Application: MLS and its variants. An immediate application of (post-quantum) mKEMs is to reduce the bandwidth cost of post-quantum deployments of secure group messaging such as the Messaging Layer Security (MLS) protocol (IETF RFC 9420 [BBR+23]). Minimizing bandwidth has been a primary design goal for MLS in the interest of scalability and supporting devices connected over low bandwidth networks (e.g. 2G cell phone networks). MLS was also designed specifically to make post-quantum secure deployments as easy as possible.[1]
The MLS use case is particulary significant as MLS is proving to be an unexpectedly versatile tool with use cases already reaching well beyond its original domain of human-to-human secure messaging. MLS's scalable key agreement functionality for dynamic groups means it will serve as a key component underpinning higher-level multi-party cryptographic protocols such as Group Private Set Intersection, Privacy Preserving Federated Learning and end-to-end encrypted real-time voice/video communication. Its efficiency is also making it an attractive tool for IoT fleet command, control and coordination.
We show that mKEMs can greatly improve the scalability of MLS. Events such as removal of users can affect the efficiency of (standard) MLS, and mKEMs provide a simple way to mitigate this. In addition, the flexibility of mKEMs allows to come up with optimized variants of MLS. See Section 4 for more details.

Application: broadcast. The private broadcast functionality of mKEMs is a fundamental one. As such, post-quantum mKEMs have applications outside of MLS. For example, mKEMs allow for more efficient synchronization of fleets of (cloud based) Hardware Security Modules (HSMs). Such fleets underpin the security of major cloud providers. Privately synchronizing secret states across HSM fleets is indispensable for ensuring reliable operations and preventing loss due to hardware failure. See Section 5 for more details.

[1] For example, this consideration was an important driver for the MLS working group's switch from the initial ART sub-protocol relying on Diffie-Hellman based Non-Interactive Key Agreement to the TreeKEM design, which instead made use of generic KEMs.

2 What is an mKEM?

2.1 Syntax

A multi-recipient key-encapsulation mechanism (mKEM) allows encapsulating a single key for multiple recipients. It consists of the following algorithms.

Parameter Generation: $\mathsf{mKEM.Setup}() \to \mathsf{pp}$ returns a fresh public parameter pp.

Key Generation: The key generation algorithm $\mathsf{mKEM.Keygen}(\mathsf{pp}) \to (\mathsf{ek}, \mathsf{dk})$ takes as input a public parameter pp returns a fresh public/secret key pair $(\mathsf{ek}, \mathsf{dk})$.

Encapsulation: The (multi-recipient) encapsulation algorithm $\mathsf{mKEM.Encap}(\mathsf{pp}, (\mathsf{ek}_i)_{i\in[N]}) \to (\vec{\mathsf{ct}}, K)$ takes in a sequence (of any length $n > 0$) of public keys and outputs a (multi-recipient) ciphertext $\vec{\mathsf{ct}}$ and an encapsulated key K.

Extract: The deterministic algorithm $\mathsf{mKEM.Ext}(\mathsf{pp}, i, \vec{\mathsf{ct}}) \to \mathsf{ct}_i$ takes as input a position index i and a multi-recipient ciphertext $\vec{\mathsf{ct}}$ and returns an individual ciphertext ct_i for the i-th recipient.

Decapsulation: The decapsulation algorithm $\mathsf{mKEM.Decaps}(\mathsf{pp}, \mathsf{dk}_i, \mathsf{ct}_i) \to K/\perp$ takes as input a secret key dk_i and an individual ciphertext ct_i. If decapsulation succeeds it returns the encapsulated key K (else $\perp$).

If the size of the individual ciphertext ct_i output by $\mathsf{mKEM.Ext}$ is independent of N, then we say that the mKEM is *efficiently decomposable*.

2.2 Security

We now define IND-CCA security for mKEMs, through what is a straightforward adaptation of the standard KEM security definition, with the difference that the adversary now gets challenged on a set of keys of its choosing, instead of a single public key. Our definition distinguishes between two different flavors, depending on whether (adaptive) corruptions by the adversary are allowed. Even though no specific attacks exploiting adaptive corruptions are known, the distinction is meaningful: known proof techniques for the weaker security notion fail when corruptions are allowed due to the so-called commitment problem. This is discussed further in Section 2.3.

The security experiment is described in Fig. 1. Roughly, it illustrates a game between a challenger and a stateful adversary $\mathcal{A}$ that runs in two parts. The first one, $\mathcal{A}_1$, gets a list of N public keys of an mKEM, of which it selects a subset on which it wants to be challenged. Then, $\mathcal{A}_2$ is given a ciphertext vector $\vec{\mathsf{ct}}^*$ and a key K^*, and must decide whether K^* is a random key, independent of $\vec{\mathsf{ct}}^*$, or $\vec{\mathsf{ct}}^*$ is an encapsulation of K^* under the subset of keys chosen by $\mathcal{A}_1$. In order to succeed, the adversary can make use of a state shared between both of its parts, as well as a decryption oracle, which allows it to query for decryptions of any ciphertext under any key, provided the query is not part of the challenge. For the stronger security notion where adaptive corruptions are allowed, $\mathcal{A}$ is additionally given access to a corruption oracle, which simply returns the secret key of the queried key-pair. Finally, the adversary wins if it guesses correctly without having corrupted any of the keys from the challenge set.

Definition 1. Let mKEM be an mKEM scheme and N an integer. We say that mKEM is $\mathsf{IND\text{-}CCA}$ secure if, for all adversaries $\mathcal{A}$ running in polynomial time,

$$\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N,1}(\mathcal{A}) \to 1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N,1}(\mathcal{A}) \to 0]$$

is negligible in the security parameter.

2.3 Security with Adaptive Corruptions

Adaptive corruptions allow an adversary to decide, on the fly, what to corrupt depending on its full view (e.g. including public parameters, keys and ciphertexts). This makes it a strong, yet realistic type of adversary which is why various regular KEM and PKE schemes have been the subject of security analyses in adaptive corruptions models. For example,

Alg. 1 Experiment $\mathsf{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N,b}(\mathcal{A})$

1: $(\mathcal{A}_1, \mathcal{A}_2) \leftarrow \mathcal{A}$
2: $\mathsf{pp} \leftarrow \mathsf{mKEM.Setup}()$
3: **for** $i \in [N]$ **do**
4: $\quad(\mathsf{ek}_i, \mathsf{dk}_i) \leftarrow \mathsf{mKEM.Keygen}(\mathsf{pp})$
5: $C \leftarrow \emptyset$
6: $((i^*_j \in [N])_{j\in[n]}, st) \leftarrow \mathcal{A}_1^{\mathsf{O}}(\mathsf{pp}, (\mathsf{ek}_i)_{i\in[N]})$
7: $(\vec{\mathsf{ct}}^*, K^*_0) \leftarrow \mathsf{mKEM.Encap}(\mathsf{pp}, (\mathsf{ek}_{i^*_j})_{j\in[n]})$
8: $K^*_1 \leftarrow \{0,1\}^\kappa$
9: $b' \leftarrow \mathcal{A}_2^{\mathsf{O}}(st, \vec{\mathsf{ct}}^*, K^*_b)$
10: **return** $b = b' \wedge \{i^*_j \mid j \in [n]\} \cap C = \emptyset$

Alg. 2 Oracle $\mathsf{Dec}(i \in [N], \mathsf{ct})$

1: **if** $\mathsf{ct}^* \neq \bot$
2: $\quad$ **if** $\exists j\ i^*_j = i \wedge \mathsf{ct} = \mathsf{mKEM.Ext}(\mathsf{pp}, j, \mathsf{ct}^*)$
3: $\quad\quad$ **return** $\bot$
4: **return** $\mathsf{mKEM.Decaps}(\mathsf{pp}, \mathsf{dk}_i, \mathsf{ct})$

Alg. 3 Oracle $\mathsf{Cor}(i \in [N])$

1: $C \leftarrow C \cup \{i\}$
2: **return** dk_i

Figure 1: IND-CCA security experiments for mKEM. With (adaptive) corruptions ($\mathsf{IND\text{-}CCA}^{\mathsf{a\text{-}mu}}$), the oracle set is $\mathsf{O} := \{\mathsf{Dec}, \mathsf{Cor}\}$. Without corruptions ($\mathsf{IND\text{-}CCA}^{\mathsf{mu}}$), the oracle set is $\mathsf{O} := \{\mathsf{Dec}\}$.

their security is often analyzed in the multi-user setting, e.g. [BBM00, ABH+21]. Here, the IND-CCA adversary chooses which out of N (independently generated) key pairs to challenge. Thus we can define two multi-user security notions for an mKEM: the standard *non-adaptive* notion $\mathsf{IND\text{-}CCA}^{\mathsf{mu}}$ in which the adversary decides prior to receiving any inputs which keys it will corrupt and $\mathsf{IND\text{-}CCA}^{\mathsf{a\text{-}mu}}$, the strictly stronger *adaptive* notion in which the adversary can decide on the fly as the execution progresses which (non-challenge) secret keys to corrupt.[2] We depict both in Figure 1.

Security proofs of mKEMs and KEMs typically consider $\mathsf{IND\text{-}CCA}^{\mathsf{mu}}$, e.g. [BBM00, ABH+21, Kur02, BBS03, BBKS07, PPS14, KKPP20]. For Diffie-Hellman (DH) based schemes this allows proving tighter security bounds, i.e., the security loss of the reduction to the DH assumption does not depend on N, see e.g. in [ABH+21] (HPKE) or in [Kur02] (a DHIES-based mKEM). Note that this affects the choice of concrete parameters for the scheme. The above proof techniques (which all use random self-reducibility) *cannot* be used to prove tight $\mathsf{IND\text{-}CCA}^{\mathsf{a\text{-}mu}}$ security. To the best of our knowledge, all reductions to date proving $\mathsf{IND\text{-}CCA}^{\mathsf{a\text{-}mu}}$ of KEMs or mKEMs lose either the factor N (using the standard hybrid argument) or at least the number of corrupted keys [AHKM22].

For the case of mKyber we do not know of any proof of $\mathsf{IND\text{-}CCA}^{\mathsf{a\text{-}mu}}$ where the reduction has a non-exponential security loss in N (resulting from guessing the set of corrupted keys). We stress that, we believe it to be very unlikely that there is an actual attack on mKyber exploiting adaptivity in the multi-user setting. Instead, our current proof techniques do not work here for a technical (i.e. the so-called "commitment") problem. It is possible that new proof techniques could be used in the future to prove $\mathsf{IND\text{-}CCA}^{\mathsf{a\text{-}mu}}$ of mKyber.
We also note that in applications where key pairs live for very short time periods, which makes corruptions unlikely, $\mathsf{IND\text{-}CCA}^{\mathsf{mu}}$ can be sufficient. This is the case e.g. for TLS. In contrast, for applications like MLS we can expect keys to live for significantly longer periods of time, especially in large groups.

3 Kyber-based mKEMs

This section defines two mKEM constructions: mKyber in Section 3.1 and amKyber in Section 3.2. The latter is secure with (adaptive) corruptions but slightly less efficient. We also

[2] Outside this section we use the term IND-CCA to refer to either, for simplicity. Which one is meant will be clear from the context.

propose in Section 3.3 concrete parameters for mKyber and amKyber. Assuming there are many recipients, instantiating (a)mKyber with these parameters results in shorter ciphertexts over instantiating them with the parameters of Kyber (which would be secure, though).

3.1 The mKyber Construction

Simplifications. For conciseness, the description of Kyber in Fig. 2 is heavily simplified compared to the one in [SAB+22]. For example:

- Some calls to hash functions or PRFs are omitted or merged together;
- We assume that $\mathbf{A}$ is part of ek. For compactness, in practice ek instead contains a seed $\mathsf{seed}_{\mathbf{A}}$, which can be extended to $\mathbf{A}$ by passing it into a XOF: $\mathbf{A} \coloneqq \mathsf{XOF}(\mathsf{seed}_{\mathbf{A}})$.
- We simplify details relative to the bit representation of mathematical objects. We also omit transitions between the NTT representation and the coefficient representation.

Making A part of the public parameters. In MLWE-based KEMs such as Kyber, the matrix $\mathbf{A}$ is different for each keypair. For mKyber, the matrix $\mathbf{A}$ is the same for all keypairs in order to enable the benefits of mKEMs. See also [KKPP20].

Syntax. As explained in (previous section), the syntax of mKEMs is different than for regular KEMs. The encapsulation procedure (Algorithm 7) now takes N encapsulation keys instead of one as in a regular KEM (Algorithm 6).
Also note that for mKEMs, the encapsulation procedure outputs a multi-recipient ciphertext $\vec{\mathsf{ct}} \coloneqq \big(\mathsf{u}, (\mathsf{v}_i)_{i\in[N]}\big)$, but a user with the decryption key dk_i only needs the partial ciphertext $\mathsf{ct}_i = (\mathbf{u}, v_i)$ as input to its decapsulation procedure (Algorithm 9).

Decomposable CCA transform. Our mKEM construction is efficiently decomposable, allowing to convert a multi-recipient ciphertext $\vec{\mathsf{ct}} \coloneqq \big(\mathsf{u}, (\mathsf{v}_i)_{i\in[N]}\big)$ of size $O(N)$ into a single-recipient ciphertext $\mathsf{ct}_i = (\mathsf{u}, \mathsf{v}_i)$ of size $O(1)$, where u is independent of the encryption keys. Decomposability [KKPP20] can be exploited for further efficiency gains, see Section 4
In order to achieve decomposability in a CCA setting, we also require a CCA transform with a decomposable flavor. To this effect, we replace the Fujisaki-Okamoto transform [FO13, HHK17] used in Kyber by a decomposable variant introduced in [KKPP20].

3.2 The amKyber Construction Secure with Adaptive Corruptions

This section describes a variant of mKyber called amKyber, proposed in [AHK+23], which achieves the stronger adaptive security notion of IND-CCA$^{\mathsf{a\text{-}mu}}$ defined in Figure 1. We describe amKyber in Figs. 3 and 4.

Like mKyber (and Kyber), the amKyber mKEM is built by applying an FO transform to an underlying CPA secure mPKE (or PKE in the case of Kyber). As shown in [AHK+23], the FO transform preserves the (quantum and classic) adaptive security of the underlying mPKE resulting in an adaptively secure mKEM. Thus, en lieu of the mPKE underlying mKyber (which we shall call mKyberPKE), we use one called amKyberPKE which [AHK+23] shows to be an adaptively secure IND-CPA mPKE. amKyber is then obtained from amKyberPKE by applying the same FO as produces mKyber from mKyberPKE.

In more detail, amKyberPKE adapts to the mPKE setting, the ideas of [GW09] for building adaptively-secure broadcast encryption. At a high level, amKyberPKE runs two parallel instances of mKyberPKE. In particular, an amKyberPKE public key consists of two mKyberPKE public keys, $\mathbf{b}_{\mathsf{l}}$ (left) and $\mathbf{b}_{\mathsf{r}}$ (right) but the corresponding amKyberPKE secret key contains just one of the two matching mKyberPKE secret keys. Which one is chosen privately and at random during key generation. As such, we can use an alternative public key generation method for the other key pair which does not produce the corresponding secret key but therefore allows for a more compact representation of the resulting amKyberPKE public key. Namely, rather than encoding the two mKyberPKE public keys explicitly as $(\mathbf{b}_{\mathsf{l}}, \mathbf{b}_{\mathsf{r}})$, they are instead encoded as the (much shorter) pair $(\mathbf{b}_{\mathsf{l}}, \mathsf{seed})$ where seed is a string of 256 bits. Together they define the second public key to be $\mathbf{b}_{\mathsf{r}} \coloneqq \mathsf{HashToEk}(\mathsf{seed}) - \mathbf{b}_{\mathsf{l}}$ where HashToEk

Alg. 4 Kyber.Keygen (pp)

$\mathsf{seed} \leftarrow \{0,1\}^{256}$
$\mathsf{seed}' \leftarrow \{0,1\}^{256}$
$\mathbf{A}, \mathbf{s}, \mathbf{e} := \mathsf{PRF}(\mathsf{seed}')$ ▷ $\mathbf{A} \in R_q^{k \times k}$
$\mathbf{b} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ ▷ $\mathbf{b}, \mathbf{s}, \mathbf{e} \in R_q^k$
$\mathsf{ek} := (\mathbf{A}, \mathbf{b})$
$\mathsf{dk} := (\mathbf{s}, \mathsf{ek}, \mathsf{seed})$
return ek, dk

Alg. 5 mKyber.Keygen(pp $\ni \mathbf{A}$)

$\mathsf{seed} \leftarrow \{0,1\}^{256}$
$\mathsf{seed}' \leftarrow \{0,1\}^{256}$
$\mathbf{s}, \mathbf{e} := \mathsf{PRF}(\mathsf{seed}')$
$\mathbf{b} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
$\mathsf{ek} := \mathbf{b}$
$\mathsf{dk} := (\mathbf{s}, \mathsf{ek}, \mathsf{seed})$
return ek, dk

Alg. 6 Kyber.Encap ($\mathsf{ek} = \mathbf{b}$)

$\mathsf{msg} \leftarrow \{0,1\}^{256}$
$\mathsf{msg} := \mathsf{H}(\mathsf{msg})$
$(K', \mathsf{coin}) := \mathsf{G}(\mathsf{msg}, \mathsf{H}(\mathsf{ek}))$
$\mathbf{r}, \mathbf{e}', e'' := \mathsf{PRF}(\mathsf{coin})$
$\mathbf{u} := \mathbf{r} \cdot \mathbf{A} + \mathbf{e}'$ ▷ $\mathbf{u}, \mathbf{r}, \mathbf{e}' \in R_q^{1 \times k}$
$v := \mathbf{r} \cdot \mathbf{b} + e'' + \mathsf{Encode}(\mathsf{msg})$ ▷ $v, e'' \in R_q$
$\mathsf{u} := \mathsf{Compress}(\mathbf{u}, d_u)$
$\mathsf{v} := \mathsf{Compress}(v, d_v)$
return $\mathsf{ct} := (\mathsf{u}, \mathsf{v}), K := \mathsf{KDF}(K', \mathsf{ct})$

Alg. 7 mKyber.Encap($(\mathsf{ek}_i = \mathbf{b}_i)_{i \in [N]}$)

$\mathsf{msg} \leftarrow \{0,1\}^{\kappa}$
$\mathsf{msg} := \mathsf{H}(\mathsf{msg})$
$K' := \mathsf{H}(\mathsf{msg})$
$\mathsf{coin} := \mathsf{G}_1(\mathsf{msg})$
$\mathbf{r}, \mathbf{e}' := \mathsf{PRF}_1(\mathsf{coin})$
$\mathbf{u} := \mathbf{r} \cdot \mathbf{A} + \mathbf{e}'$
$\mathsf{u} := \mathsf{Compress}(\mathbf{u}, d_u)$
for $i \in [N]$ **do**
  $\mathsf{coin}_i := \mathsf{G}_2(\mathsf{ek}_i, \mathsf{msg})$
  $e_i'' := \mathsf{PRF}_2(\mathsf{coin}_i)$
  $v_i := \mathbf{r} \cdot \mathbf{b}_i + e_i'' + \mathsf{Encode}(\mathsf{msg})$
  $\mathsf{v}_i := \mathsf{Compress}(v_i, d_v)$
$\vec{\mathsf{ct}} := (\mathsf{u}, (\mathsf{v}_i)_{i \in [N]})$
return $\vec{\mathsf{ct}}, K := \mathsf{KDF}(K', \mathsf{ct})$

Alg. 8 Kyber.Decaps ($\mathsf{dk}, \mathsf{ct} = (\mathsf{u}, \mathsf{v})$)

$(\mathbf{s}, \mathbf{b}, \mathsf{seed}) \leftarrow \mathsf{dk}$
$\mathbf{u} := \mathsf{Decompress}(\mathsf{u}, d_u)$
$v := \mathsf{Decompress}(\mathsf{v}, d_v)$
$\mathsf{msg} := \mathsf{Decode}(v - \mathbf{u} \cdot \mathbf{s})$
$(K', \mathsf{coin}) := \mathsf{G}(\mathsf{msg}, \mathsf{H}(\mathsf{ek}))$
$\bar{\mathbf{r}}, \bar{\mathbf{e}}', \bar{e}'' := \mathsf{PRF}(\mathsf{coin})$
$\bar{\mathbf{u}} := \bar{\mathbf{r}} \cdot \mathbf{A} + \bar{\mathbf{e}}'$
$\bar{v} := \bar{\mathbf{r}} \cdot \mathbf{b} + e'' + \mathsf{Encode}(\mathsf{msg})$
$\bar{\mathsf{u}} := \mathsf{Compress}(\bar{\mathbf{u}}, d_u)$
$\bar{\mathsf{v}} := \mathsf{Compress}(\bar{v}, d_v)$
$\bar{\mathsf{ct}} := \bar{\mathsf{u}}, \bar{\mathsf{v}}$
if $\bar{\mathsf{ct}} = \mathsf{ct}$
  return $K := \mathsf{KDF}(K', \mathsf{ct})$
else
  return $K := \mathsf{KDF}(\mathsf{seed}, \mathsf{ct})$

Re-encryption

Alg. 9 mKyber.Decaps(dk_i, $\mathsf{ct}_i = (\mathsf{u}, \mathsf{v}_i)$)

$(\mathbf{s}_i, \mathbf{b}_i, \mathsf{seed}_i) \leftarrow \mathsf{dk}$
$\mathbf{u} := \mathsf{Decompress}(\mathsf{u}, d_u)$
$v_i := \mathsf{Decompress}(\mathsf{v}_i, d_v)$
$\mathsf{msg} := \mathsf{Decode}(v_i - \mathbf{u} \cdot \mathbf{s}_i)$
$K' := \mathsf{H}(\mathsf{msg})$
$\mathsf{coin} := \mathsf{G}_1(\mathsf{msg})$
$\mathbf{r}, \mathbf{e}' := \mathsf{PRF}_1(\mathsf{coin})$
$\mathsf{coin}_i := \mathsf{G}_2(\mathsf{ek}_i, \mathsf{msg})$
$e_i'' := \mathsf{PRF}_2(\mathsf{coin}_i)$
$\mathbf{u} := \mathbf{r} \cdot \mathbf{A} + \mathbf{e}'$
$v_i := \mathbf{r} \cdot \mathbf{b}_i + e_i'' + \mathsf{Encode}(\mathsf{msg})$
$\bar{\mathsf{u}} := \mathsf{Compress}(\bar{\mathbf{u}}, d_u)$
$\bar{\mathsf{v}}_i := \mathsf{Compress}(\bar{v}_i, d_v)$
$\bar{\mathsf{ct}}_i := (\bar{\mathbf{u}}, \bar{v}_i)$
if $\bar{\mathsf{ct}}_i = \mathsf{ct}_i$
  return $K := \mathsf{KDF}(K', \mathsf{ct}_i)$
else
  return $K := \mathsf{KDF}(\mathsf{seed}_i, \mathsf{ct}_i)$

Re-encryption

Figure 2: Side-by-side comparison of Kyber (Algorithms 4, 6 and 8) and mKyber (Algorithms 5, 7 and 9). The main algorithmic differences are highlighted, and we discuss them in Section 3.1.

is a public hash function mapping seeds to public keys. To generate an amKyberPKE key pair first choose a uniform random seed. Next flip a fair coin and, depending on the outcome, generate either $(\mathsf{dk}_\mathsf{l}, \mathbf{b}_\mathsf{l})$ or $(\mathsf{dk}_\mathsf{r}, \mathbf{b}_\mathsf{r})$ using mKyberPKE key generation algorithm. Finally, set remaining mKyberPKE public key such that $\mathbf{b}_\mathsf{l} := \mathsf{HashToEk}(\mathsf{seed}) - \mathbf{b}_\mathsf{r}$.

Alg. 10 amKyber.Keygen($\mathsf{pp} = \mathbf{A}$)

$(\mathsf{ek}', \mathsf{dk}') := \mathsf{mKyber.Keygen}(\mathsf{pp})$
$\mathsf{seed} \leftarrow \{0,1\}^{256}$
$\mathsf{swpEk} \leftarrow \{0,1\}$
if $\mathsf{swpEk} = 1$
  $\mathsf{ek}' := \mathsf{HashToEk}(\mathsf{seed}) - \mathsf{ek}'$
return $\mathsf{ek} := (\mathsf{ek}', \mathsf{seed}), \mathsf{dk} := (\mathsf{dk}', \mathsf{swpEk}, \mathsf{ek}', \mathsf{seed})$

Alg. 11 mKyber.Encap$\big((\mathsf{ek}_i = \mathbf{b}_i)_{i\in[N]}\big)$

$\mathsf{msg} \leftarrow \{0,1\}^\kappa$
$\mathsf{msg} := \mathsf{H}(\mathsf{msg})$
$K' := \mathsf{H}(\mathsf{msg})$
$(\mathsf{u}, *) := \mathsf{CpaEncU}(\mathsf{msg})$
for $i \in [N]$ **do**
  $\mathsf{v}_i := \mathsf{CpaEncV}(\mathsf{msg}, \mathsf{ek}_i, \mathsf{G}_2(\mathsf{msg}, \mathsf{ek}_i))$
$\vec{\mathsf{ct}} := \big(\mathsf{u}, (\mathsf{v}_i)_{i\in[N]}\big)$
return $\vec{\mathsf{ct}}, K := \mathsf{KDF}(K', \mathsf{ct})$

Alg. 12 amKyber.Encap$\big((\mathsf{ek}_i)_{i\in[N]}\big)$

$\mathsf{msg} \leftarrow \{0,1\}^\kappa$
$\mathsf{msg} := \mathsf{H}(\mathsf{msg})$
$K' := \mathsf{H}(\mathsf{msg})$
$\mathsf{msg}_\mathsf{l}, \mathsf{msg}_\mathsf{r} := \mathsf{G}_3(\mathsf{msg})$
$(\mathsf{u}_\mathsf{l}, *) := \mathsf{CpaEncU}(\mathsf{msg}_\mathsf{l})$
$\mathsf{u} := (\mathsf{u}_\mathsf{l}, \mathsf{u}_\mathsf{r})$
$(\mathsf{u}_\mathsf{r}, *) := \mathsf{CpaEncU}(\mathsf{msg}_\mathsf{r})$
for $i \in [N]$ **do**
  $(\mathbf{b}_\mathsf{l}, \mathsf{seed}) := \mathsf{ek}_i$
  $\mathbf{b}_\mathsf{r} := \mathsf{HashToEk}(\mathsf{seed}) - \mathbf{b}_\mathsf{l}$
  $\mathsf{swpC} := \mathsf{G}_4(\mathsf{msg}, \mathsf{ek}_i) \in \{0,1\}$
  if $\mathsf{swpC} = 1$
    $(\mathbf{b}_\mathsf{l}, \mathbf{b}_\mathsf{r}) \leftarrow (\mathbf{b}_\mathsf{r}, \mathbf{b}_\mathsf{l})$
  $\mathsf{v}_\mathsf{l} := \mathsf{CpaEncV}(\mathsf{msg}, \mathbf{b}_\mathsf{l}, \mathsf{G}_2(\mathsf{msg}_\mathsf{l}, \mathsf{ek}_i))$
  $\mathsf{v}_\mathsf{r} := \mathsf{CpaEncV}(\mathsf{msg}, \mathbf{b}_\mathsf{r}, \mathsf{G}_2(\mathsf{msg}_\mathsf{r}, \mathsf{ek}_i))$
  $\mathsf{v}_i := (v_\mathsf{l}, v_\mathsf{r}, \mathsf{swpC})$
$\vec{\mathsf{ct}} := \big(\mathsf{u}, (\mathsf{v}_i)_{i\in[N]}\big)$
return $\vec{\mathsf{ct}}, K := \mathsf{KDF}(K', \mathsf{ct})$)

Figure 3: Key generation of amKyber (Algorithm 10) as well as a side-by-side comparison of the encapsulation of mKyber (Algorithm 11) and amKyber (Algorithm 12). The main algorithmic differences are highlighted, and we discuss them in Section 3.2.

To encrypt msg to multiple recipients, amKyberPKE also runs mKyberPKE's encryption twice: It first generates two vectors $\mathbf{u}$ called $\mathbf{u}_\mathsf{l}$ and $\mathbf{u}_\mathsf{r}$. Then, for each recipient, it flips a fair coin (using the random oracle on msg as required by the FO transform) to decide which of the recipient's keys, $\mathbf{b}_\mathsf{l}$ or $\mathbf{b}_\mathsf{r}$, goes to the recipients left mKyber instance (i.e. with public key $\mathbf{u}_\mathsf{l}$) and which goes to the right instance. To decrypt msg, the recipient in amKyberPKE runs mKyberPKE with the secret key it knows and the corresponding invocation indicated by the sender.

Security. amKyber uses the same parameters as mKyber. The adaptive security proof of amKyber can be found in [KKPP20, AHK+23]. In particular, [KKPP20] proves mKyberPKE to be (non-adaptively) IND-CPA secure mPKE. Next, in [AHK+23] proves their transformation of mKyberPKE into amKyberPKE results in an *adaptively* IND-CPA secure mPKE.

Alg. 13 mKyber.Decaps$(\mathsf{dk}_i, \mathsf{ct}_i = (\mathsf{u}, \mathsf{v}_i))$

$(\mathbf{s}_i, \mathbf{b}_i, \mathsf{seed}_i, \mathsf{ek}') \leftarrow \mathsf{dk}$
$\mathbf{u} := \mathsf{Decompress}(\mathsf{u}, d_u)$
$v_i := \mathsf{Decompress}(\mathsf{v}_i, d_v)$
$\mathsf{msg} := \mathsf{Decode}(v_i - \mathbf{u} \cdot \mathbf{s}_i)$
$K' := \mathsf{H}(\mathsf{msg})$
$(\bar{\mathsf{u}}, *) := \mathsf{CpaEncU}(\mathsf{msg})$
$\bar{\mathsf{v}} := \mathsf{CpaEncV}(\mathsf{msg}, \mathbf{b}_i, \mathsf{G}_2(\mathsf{msg}, \mathsf{ek}_i))$
$\bar{\mathsf{ct}}_i := (\bar{\mathsf{u}}, \bar{\mathsf{v}})$
if $\bar{\mathsf{ct}}_i = \mathsf{ct}_i$
  return $K := \mathsf{KDF}(K', \mathsf{ct}_i)$
else
  return $K := \mathsf{KDF}(\mathsf{seed}_i, \mathsf{ct}_i)$

Alg. 14 amKyber.Decaps$(\mathsf{dk}_i, \mathsf{ct}_i = (\mathsf{u}, \mathsf{v}_i))$

$(\mathsf{dk}', \mathsf{swpEk}, \mathbf{b}_\mathsf{l}, \mathsf{seed}) := \mathsf{dk}_i$
$\mathbf{b}_\mathsf{r} := \mathsf{HashToEk}(\mathsf{seed}) - \mathbf{b}_\mathsf{l}$
$(\mathsf{v}_\mathsf{l}, \mathsf{v}_\mathsf{r}, \mathsf{swpC}) := \mathsf{v}_i$
$(\mathsf{u}_\mathsf{l}, \mathsf{u}_\mathsf{r}) := \mathsf{u}_i$
$\mathsf{ek}_i := (\mathbf{b}_\mathsf{l}, \mathsf{seed})$
if $\mathsf{swpEk} \,\mathrm{XOR}\, \mathsf{swpC} = 0$
  $\mathsf{side} := \mathsf{l}$
else
  $\mathsf{side} := \mathsf{r}$
$(\mathbf{s}, \mathbf{b}, \mathsf{seed}) := \mathsf{dk}'$
$\mathbf{u} := \mathsf{Decompress}(\mathsf{u}_\mathsf{side}, d_u)$
$v := \mathsf{Decompress}(\quad, d_v)$
$\mathsf{msg} := \mathsf{Decode}(v - \mathbf{u} \cdot \mathbf{s})$
$K' := \mathsf{H}(\mathsf{msg})$
$\mathsf{msg}_\mathsf{l}, \mathsf{msg}_\mathsf{r} := \mathsf{G}_3(\mathsf{msg})$
for $\mathsf{side} \in \{\mathsf{l}, \mathsf{r}\}$ **do**
  $(\bar{\mathsf{u}}_\mathsf{side}, *) := \mathsf{CpaEncU}(\mathsf{msg}_\mathsf{side})$
  $\mathsf{coin} := \mathsf{G}_2(\mathsf{msg}_\mathsf{side}, \mathsf{ek}_i)$
  $\bar{\mathsf{v}}_\mathsf{side} := \mathsf{CpaEncV}(\mathsf{msg}, \mathbf{b}_\mathsf{side}, \mathsf{coin})$
$\overline{\mathsf{swpC}} := \mathsf{G}_4(\mathsf{msg}, \mathsf{ek}_i)$
if $(\bar{\mathsf{u}}_\mathsf{l}, \bar{\mathsf{v}}_\mathsf{l}, \bar{\mathsf{u}}_\mathsf{r}, \bar{\mathsf{v}}_\mathsf{r}, \overline{\mathsf{swpC}}) = (\mathsf{u}_\mathsf{l}, \mathsf{v}_\mathsf{l}, \mathsf{u}_\mathsf{r}, \mathsf{v}_\mathsf{r}, \mathsf{swpC})$
  return $K := \mathsf{KDF}(K', \mathsf{ct}_i)$
else
  return $K := \mathsf{KDF}(\mathsf{seed}_i, \mathsf{ct}_i)$

Alg. 15 CpaEncU(msg)

$\mathbf{r}, \mathbf{e}' := \mathsf{PRF}_1(\mathsf{G}_1(\mathsf{msg}))$
$\mathbf{u} := \mathbf{r} \cdot \mathbf{A} + \mathbf{e}'$
$\mathsf{u} := \mathsf{Compress}(\mathbf{u}, d_u)$
return $(\mathsf{u}, \mathbf{r})$

Alg. 16 CpaEncV$(\mathsf{msg}, \mathsf{ek}_i = \mathbf{b}_i, \mathsf{coin}_i)$

$(*, \mathbf{r}) := \mathsf{CpaEncU}(\mathsf{msg})$
$e_i'' := \mathsf{PRF}_2(\mathsf{coin}_i)$
$v_i := \mathbf{r} \cdot \mathbf{b}_i + e_i'' + \mathsf{Encode}(\mathsf{msg})$
return $\mathsf{v}_i := \mathsf{Compress}(v_i, d_v)$

Figure 4: Side-by-side comparison of decapsulation of mKyber (Algorithm 11) and amKyber (Algorithm 12). The main algorithmic differences are highlighted, and we discuss them in Section 3.2.

The same paper also proves that the FO transform of [KKPP20] applied to amKyberPKE produces amKyber; an adaptively IND-CCA$^{\text{a-mu}}$ secure mKEM. We note that all of the above security statements hold both against classical and quantum adversaries.
Together, these results provide strong evidence for the security of amKyber design.

Efficiency. An amKyber public key has only 256 bits in addition to an mKyber public key. An amKyber ciphertext is twice larger than an mKyber ciphertext. Note that asymptotically, amKyber is still 8 times smaller than the sum of sizes of N Kyber ciphertexts.

3.3 Parameters Selection

In Table 1, we propose concrete mKyber parameters targeting the NIST security level I (at least as hard as key-recovery on AES-128). As discussed in Section 1, the parameters are largely similar to those of Kyber; we only tweaked the parameters $(d_u, d_v, |\mathsf{msg}|)$, as it allows to greatly decrease the size of $|\mathsf{v}|$. While Kyber and mKyber are largely similar, the mKEM setting impact both the efficiency analysis and the security analysis. We analyse both of them separately.

Table 1: Parameter sets of Kyber512 and mKyber512.

	Parameters								Sizes in bytes		
	q	n	k	η_1	η_2	d_u	d_v	$\lvert\mathsf{msg}\rvert$	$\lvert\mathsf{ek}\rvert$	$\lvert\mathsf{u}\rvert$	$\lvert\mathsf{v}\rvert$
Kyber512	3329	256	2	3	2	10	4	32	800	640	128
mKyber512	3329	256	2	3	2	11	3	16	768	704	48

Efficiency. Let us note $|\mathsf{x}|$ the size in bytes of an object. In the KEM regime, it is of interest for most applications to minimize the ciphertext size $|\mathsf{ct}|$, the encapsulation key size $|\mathsf{ek}|$, or some linear combination of the two. In the mKEM regime, a multi-recipient ciphertext is of the form $\vec{\mathsf{ct}} := (\mathsf{u}, (\mathsf{v}_i)_{i\in[N]})$. Therefore there is a high incentive to minimize $|\mathsf{v}_i|$, since asymptotically $|\vec{\mathsf{ct}}| \sim N \cdot |\mathsf{v}_i|$. There are a few tricks we can use to minimize $|\mathsf{v}_i|$.

1. **Shorter** msg. In Kyber, the message msg is 256 bits long across all parameter sets. However, if we note κ is the targeted bit-security level (i.e. $\kappa \in \{128, 192, 256\}$ for the NIST level I, III, V), it suffices to take a message of κ bits.
2. **Coefficient dropping.** When applying the previous idea, we only need to encode κ bits in a polynomial v_i of 256 coefficients. We only need to send κ coefficients of v_i instead of n coefficients. In effect, this divides $|\mathsf{v}_i|$ by factor $256/\kappa$, which is a factor two when $\kappa = 128$.
3. **Bit dropping.** It is customary to apply bit dropping on u and v. This is parametrized by d_u and d_v, which are the number of bits sent per coeffient of $\mathbf{u}$ and v_i, respectively. The bitsize of v_i is therefore $\kappa \cdot d_v$.

   Since our goal is to minimize $|\mathsf{v}_i|$, we reduce d_v from 4 to 3. This increases the decryption failure rate (DFR), so we increase d_u from 10 to 11 in order to keep the DFR low.

Putting these optimisations together, we manage to obtain $|\mathsf{v}_i| = 48$.

Security. Both Kyber and mKyber rely on MLWE for the security of the encapsulation key. For the ciphertext, due to the simultaneous presence of additive noise and rounding (via Compress), they both rely on a hybrid between MLWE and MLWR which we will call MLWER. The major difference is that a mKyber multi-ciphertext $\vec{\mathsf{ct}} := (\mathsf{u}, (\mathsf{v}_i)_{i\in[N]})$ contains a number of MLWER samples that is affine in N. This new parameter has varying impacts on the existing methods to solve MLWE, MLWR and MLWER. There are roughly three families of such methods:

1. **Lattices.** Methods based on (primal or dual) lattice reduction are usually the most relevant for most parametrizations of lattice-based schemes. For these methods, having a large number of samples provide results little to no advantage.

2. **Algebraic.** Algebraic methods, such as Arora-Ge and its variants, rely on linearization. They require a large number of samples: $(k \cdot n)^D$, where D is the size of the support of the errors (both due to additive noise and to rounding). While the required number of samples is too large in the KEM regime, mKEM ciphertexts contain a large number of MLWER samples. The relevance of Arora-Ge therefore needs to be re-assessed.

   Fortunately, the ciphertexts in mKyber are very noisy. Indeed, each v_i undergoes heavy bit dropping on each coefficient, see Table 1. This makes the required number of samples much larger than 2^κ, which rules out the Arora-Ge attack in its current form.

3. **Combinatorial.** Combinatorial methods, such as BKW, combine lattice attacks and guessing. Like Arora-Ge, BKW and its variants need a large number of samples and therefore need to be considered for mKyber. Fortunately, they are also very sensitive to the size of the errors' supports. Due to the heavy rounding performed on the v_i, BKW requires an intractable number of samples and is in effect inoperative against the parameters in Table 1.

A fully detailed discussion on the security analysis of mKEMs against known attacks can be found in [HKP+21, Appendix G]. The insights and trade-offs discussed in this section are summarized in Table 2.

Table 2: Impact of parameters on security and performance metrics. Terminology: ↗ (resp. =, resp. ↘) indicates that increasing this parameter has a positive (resp. essentially neutral, resp. negative) impact on the considered metric.

	Communication cost			Correctness	Ciphertext security		
	ek	u	v_i		Arora-Ge	BKW	Lattice
N	=	=	=	=	↘	↘	=
q	↘	=	=	↗	=	=	↘
n	↘	↘	=	↘	↗	↗	↗
k	↘	↘	=	↘	↗	↗	↗
η_1	=	=	=	↘	↗	↗	↗
η_2	=	=	=	↘	↗	↗	↗
d_u	=	↘	=	↘	↗	↗	↗
d_v	=	=	↘	↘	↗	↗	↗

4 Application: MLS and its Variants

MLS and TreeKEM. Messaging Layer Security (MLS), is a protocol for end-to-end security. MLS has been standardized by IETF under RFC 9420 [BBR+23] in March 2023, and enjoys the support of several industry actors (Google, Amazon, Cisco, etc.).
The notion of continuous group key agreement (CGKA) was put forward to capture the notion of secure group management which lies at the core of secure group messaging and other protocols. The main function of a CGKA is to ensure the secure distribution of a shared key K inside a group of N users connected to an untrusted server. A CGKA must be able to support the addition or removal of users to the group, as well as properties such as forward secrecy and post-compromise security.
Inside MLS, the sub-protocol TreeKEM[3] performs the function of a CGKA. In TreeKEM, the N users inside a group are positioned at the leaves of a binary tree called the ratchet tree, see Fig. 5a. To each node i of the ratchet tree is associated an encapsulation keypair $(\mathsf{ek}_i, \mathsf{dk}_i)$. The ratchet tree maintains the following tree invariant [BBR+23, §4.2]:

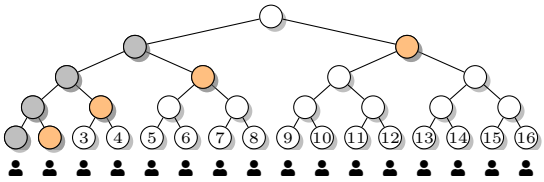
(TI) "*The private key for a node in the tree is known to a member of the group if and only if the node's subtree contains that member's leaf.*"

[3]While the term "TreeKEM" itself does not appear in the MLS RFC [BBR+23], it is implemented through the concept of "ratchet trees". The term TreeKEM is commonly used as shorthand in research articles to refer to this particular part of MLS. We follow the same practice here.

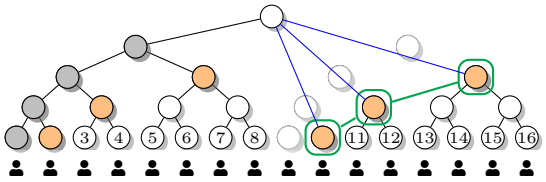
A critical operation in MLS/TreeKEM consists of a user refreshing their cryptographic keys inside the ratchet tree. They do so via the following steps:

1. Generate a node secret s_0 for their leaf node;
2. For each node i in their direct path, i.e. the path from their leaf node to the root, starting from the leaf node, use the node secret s_i as follows:
   (a) Pass s_i into a PRF to derive its parent node's secret $s_{\mathsf{parent}(i)} = s_{i+1}$ (except when i is the root).
   (b) Pass s_i through a PRF with a different input than in Item 2a and use the output to derive a KEM key pair $(\mathsf{ek}_i, \mathsf{dk}_i)$.
   (c) Compute a ciphertext ct_i encrypting s_{i+1} under the encapsulation key of the sibling node of i using the KEM-DEM paradigm and concretely HPKE, defined in RFC 9180 [BBLW22]. (Except for the leaf node)
3. Broadcast a *commit message* containing all the $(\mathsf{ek}_i, \mathsf{ct}_i)$ for i on the path of the user.

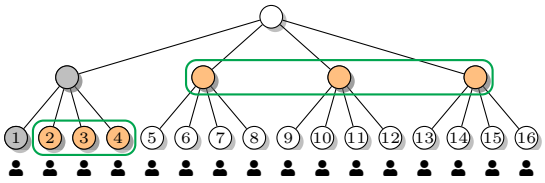
One can show that this preserves the tree invariant (TI). This method enjoys great communication complexity; a commit message contains $\lceil \log N \rceil$ public keys and as many ciphertexts.



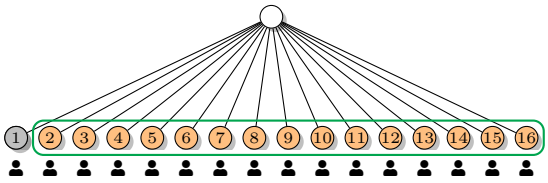
(a) TreeKEM, complete ratchet tree



(b) TreeKEM after a user has been removed and their direct path blanked out. The three boxed ciphertexts encrypt the same value: the node secret of the root.



(c) m-ary TreeKEM [KKPP20], full ratchet tree



(d) Chained CmPKE [HKP+21]

Figure 5: Illustration of the refresh operation for four scenarii (involving three distinct CGKAs). In each scenario, the leftmost user refreshes their cryptographic keys by broadcasting a commit message. Each figure highlights the nodes for which the commit message contains a public key ( ) or a ciphertext ( ). When k ciphertexts encrypt the same message, they are regrouped in a same box ( ); when this is the case, we may use a mKEM multi-ciphertext instead of k KEM ciphertexts.

Blanking. When a user j is removed from a group in TreeKEM, in order to preserve the tree invariant (TI), all the nodes in their path are blanked out, meaning that these nodes are now considered empty. A blank node becomes populated again if: (i) a user i refreshes their key by broadcasting a commit message and (ii) the user i has the blank node in their path. Note that, due to the tree invariant (TI), a user may not generate a node secret for a node that is not in their co-path. By blanking up to $\lceil \log N \rceil$ nodes, removing users may disrupt the topology of the ratchet tree. This is illustrated by Fig. 5b: the ninth user (from the left) has been removed from the group and their direct path blanked out. When the leftmost user sends a commit message, they now need to send 6 ciphertexts instead of 4, since the node secret of the root needs to be encrypted to 3 nodes instead of 1. In large groups, blank nodes can have a significant adverse effect on the efficiency of TreeKEM.
Our first application of mKEMs is TreeKEM. When implemented with a mKEM instead of a KEM, the efficiency of TreeKEM is more resilient to topology changes provoked by removing users. For the example of Fig. 5b, switching from Kyber to mKyber decreases the (m)KEM-related overhead from 12320 to 6176 bytes, a 50% improvement.

m-ary TreeKEM. A second application of mKEMs can be found by leveraging their flexibility to explore the design space of TreeKEM for better trade-offs. Indeed, instead of a binary tree, one could instantiate TreeKEM with an m-ary tree, for $m > 2$. An example is provided in Fig. 5c for $m = 4$. In a commit message for a full ratchet tree, this reduces the number of public keys to $\lceil \log_m N \rceil$ but increases the number of ciphertexts to $(m-1)\lceil \log_m N \rceil$, so it's far from obvious that it would be a good trade-off in general.
Fortunately, for each layer of the ratchet tree, the $m-1$ ciphertexts in this layer all encrypt the same message. This means these $m-1$ KEM ciphertexts can be replaced by a single mKEM ciphertext for $m-1$ recipients, which size may be much shorter in practice. For 16 users, switching from (binary) TreeKEM with Kyber to 4-ary TreeKEM with mKyber decreases the (m)KEM-related bandwidth overhead from 6272 to 3232 bytes. A more detailed presentation of m-ary TreeKEM is found in [KKPP20]

Chained CmPKE. A final application of mKEMs can be found by taking the m-ary TreeKEM idea to the extreme: by setting $m = N$, we get a flat tree. This is the basis of the Chained CmPKE protocol, illustrated in Fig. 5d; for this example, the bandwidth overhead related to mKyber is $1472 + (N-1) \cdot 48 = 2192$ bytes, a 65% gain compared to the standard TreeKEM with Kyber.
Note that the bandwidth overhead is linear in N. Fortunately, we can also exploit the fact that mKyber is efficiently decomposable. The uploaded commit message contains a multi-recipient ciphertext that is a N-uple $(\mathsf{u}, (\mathsf{v}_i)_i)$, but each recipient only needs to download $(\mathsf{u}, \mathsf{v}_i)$ to decrypt the message. This allows further savings in the overall protocol. More details can be found in [HKP+21].

Table 3: Comparison of the bandwidth costs when using Kyber vs mKyber, for the four scenarii in Fig. 5. For clarity, these numbers ignore (m)KEM-independent values that may be included in the commit message[4].

		Cost with Kyber			**Cost with** mKyber			
Reference	Description	ek	ct	Overhead	ek	u	v_i	Overhead
Fig. 5a	TreeKEM (full tree)	4	4	6272	4	4	4	6080
Fig. 5b	TreeKEM (blanked)	4	6	12320	4	4	6	**6176**
Fig. 5c	4-ary TreeKEM	2	6	6208	2	2	6	**3232**
Fig. 5d	Chained CmPKE	1	15	7808	1	1	15	**2192**

5 Application: Broadcast Scenario

The broadcast scenario, in which one sender transmits the same keying material simultaneously to multiple receivers with authentic static public-keys (discussed within NIST SP 800-56A-rev2) remains an important use-case for the cloud. In particular, this is the case for vendors that network a collection of hosts in a high-availability setting that require the synchronization of cryptographic state/keys across hosts.

A common example of such an application is within a fleet of cloud Hardware Security Modules (HMSs) that provides a key management system to generate, manage and distribute key material for encryption and authentication of user data across cloud computing or large cluster environments. In order for the fleet to synchronize state across all other members, it must first establish a key transport mechanism. To achieve this, SP 800-56A-rev2 relaxes the prohibition against the reuse of an ephemeral Diffie-Hellman key pair in broadcast scenarios, such that the key transport sender can use the same ephemeral key pair when establishing key-wrapping keys with the multiple key-transport receivers.

In such broadcast scenarios, key agreement is often performed by first collecting all static public-keys of members in the group (HSMs) into a public-key repository, which can then

[4] In addition to the (m)KEM-related overhead, commit messages also contain additional data such as a signature, a hash, a constant-size header, etc. See [AHKM22, Figure 8] for an estimate.

be distributed among the fleet so that individual key exchanges can be performed by users as needed. As such, the asymmetric cryptography used in key establishment across large groups can bottleneck fleet performance, particularly in settings in which group members are dynamic and members are changing frequently, or in systems that rotate key establishment keys frequently for forward security requirements. Such issues are exacerbated further when considering the requirement for quantum-resistance. Post-quantum KEMs generally have much larger public keys and ciphertexts than their classical counter parts e.g., 800B for a Kyber512 public key vs. 33B for a (compressed) ECDH P-256 public key.

Using an mKEM instead of a KEM mitigates this bottleneck in multiple ways. First, it drastically decreases bandwidth and computation requirements from the sender, asymptotically 16 times with mKyber instead of Kyber. Second, since the goal is to establish a *single* group key, the sender equipped with a regular KEM has to use the KEM-DEM paradigm to wrap the group key. On the other hand, an mKEM already produces a single key. This means that to send a key to N members with Kyber, the sender generates N Kyber ciphertexts and N symmetric encryptions, while with mKyber it generates one mKyber ciphertext and no symmetric encryptions. In other words, we take advantage of the fact that Kyber already consists of an underlying *encryption* scheme where we can choose the key to be the same. This reduces sender and receiver computation and bandwidth cost.

6 Further reading

Due to space constraints, this paper only gives a high-level view of mKEMs and their applications. We provide further references here.

Classical mKEMs. The possibility of saving bandwidth when encrypting to multiple recipients has first been studied in [BBM00, Kur02, BBS03], via multi-recipient constructions based on El Gamal and Cramer-Shoup, which rely on classical assumptions. The term mKEM has been coined by Smart [Sma05] and studied further in works [HK07, BF07, MH13].

Post-quantum mKEMs. Comparatively, the study of post-quantum mKEMs is much more recent. A LPN-based construction was proposed in [CLQY18] but later proven insecure due in [KKPP20] to a misuse of the Fujisaki-Okamoto transform. The first secure constructions for post-quantum mKEMs have been proposed in [KKPP20] and further developed in [HKP+21, AHK+23]. These three papers propose multi-recipient adaptations of the LPR/Lindner-Peikert framework [LPR10, LP11], which underlied many of the candidates to the 2017 NIST PQC call: Kyber (future ML-KEM), FrodoKEM, Saber, etc.[5]

Applications. The works of [KKPP20, HKP+21, AHKM22] apply mKEMs in the context of secure messaging, particularly the MLS protocol [BBR+23]. The present work proposes a natural application in the context of broadcast scenarios (Section 5), and we expect further applications to be found.

References

[ABH+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE standard. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 87–116. Springer, Heidelberg, October 2021.

[AHK+23] Joël Alwen, Dominik Hartmann, Eike Kiltz, Marta Mularczyk, and Peter Schwabe. Post-quantum multi-recipient public key encryption. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 1108–1122. ACM, 2023.

[5] [KKPP20] also proposed mKEMs based on SIDH/SIKE and CSIDH; the recent cryptanalytic advances on CSIDH [BS20, Pei20] and SIDH [CD23, MMP+23, Rob23] make these insecure.

[AHKM22] Joël Alwen, Dominik Hartmann, Eike Kiltz, and Marta Mularczyk. Server-aided continuous group key agreement. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 69–82. ACM Press, November 2022.

[BBKS07] Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Transactions on Information Theory*, 53(11):3927–3943, 2007.

[BBLW22] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180, February 2022.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.

[BBR+23] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[BBS03] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 85–99. Springer, Heidelberg, January 2003.

[BF07] Manuel Barbosa and Pooya Farshim. Randomness reuse: Extensions and improvements. In *IMA International Conference on Cryptography and Coding*, pages 257–276. Springer, 2007.

[BS20] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 493–522. Springer, Heidelberg, May 2020.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Heidelberg, April 2023.

[CLQY18] Haitao Cheng, Xiangxue Li, Haifeng Qian, and Di Yan. Cca secure multi-recipient kem from lpn. In *ICICS*, pages 513–529. Springer, 2018.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 171–188. Springer, Heidelberg, April 2009.

[HBD+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017.

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 553–571. Springer, Heidelberg, August 2007.

[HKP+21] Keitaro Hashimoto, Shuichi Katsumata, Eamonn Postlethwaite, Thomas Prest, and Bas Westerbaan. A concrete treatment of efficient continuous group key agreement via multi-recipient PKEs. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1441–1462. ACM Press, November 2021.

[KKPP20] Shuichi Katsumata, Kris Kwiatkowski, Federico Pintore, and Thomas Prest. Scalable ciphertext compression techniques for post-quantum KEMs and their applications. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 289–320. Springer, Heidelberg, December 2020.

[Kur02] Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In David Naccache and Pascal Paillier, editors, *PKC 2002*, volume 2274 of *LNCS*, pages 48–63. Springer, Heidelberg, February 2002.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Heidelberg, February 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

[MH13] Takahiro Matsuda and Goichiro Hanaoka. Key encapsulation mechanisms from extractable hash proof systems, revisited. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 332–351. Springer, Heidelberg, February / March 2013.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 448–471. Springer, Heidelberg, April 2023.

[Pei20] Chris Peikert. He gives C-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 463–492. Springer, Heidelberg, May 2020.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[PPS14] Alexandre Pinto, Bertram Poettering, and Jacob C. N. Schuldt. Multi-recipient encryption, revisited. In Shiho Moriai, Trent Jaeger, and Kouichi Sakurai, editors, *ASIACCS 14*, pages 229–238. ACM Press, June 2014.

[Rob23] Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 472–503. Springer, Heidelberg, April 2023.

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Sma05] Nigel P. Smart. Efficient key encapsulation to multiple parties. In Carlo Blundo and Stelvio Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 208–219. Springer, Heidelberg, September 2005.