Scalable Ciphertext Compression Techniques for Post-Quantum KEMs and their Applications

Shuichi Katsumata[1], Kris Kwiatkowski[2], Federico Pintore[3], Thomas Prest[2]

[1] National Institute of Advanced Industrial Science and Technology (AIST), JP
[2] PQShield, UK
[3] Mathematical Institute, University of Oxford, UK.

Abstract. A *multi-recipient* key encapsulation mechanism, or mKEM, provides a scalable solution to securely communicating to a large group, and offers savings in both bandwidth and computational cost compared to the trivial solution of communicating with each member individually. All prior works on mKEM are only limited to classical assumptions and, although some generic constructions are known, they all require specific properties that are not shared by most post-quantum schemes. In this work, we first provide a simple and efficient generic construction of mKEM that can be instantiated from versatile assumptions, including post-quantum ones. We then study these mKEM instantiations at a practical level using 8 post-quantum KEMs (which are lattice and isogeny-based NIST candidates), and CSIDH, and show that compared to the trivial solution, our mKEM offers savings of at least one order of magnitude in the bandwidth, and make encryption time shorter by a factor ranging from 1.92 to 35. Additionally, we show that by combining mKEM with the TreeKEM protocol used by MLS – an IETF draft for secure group messaging – we obtain significant bandwidth savings.

1 Introduction

Secure communication within a system of several users is becoming indispensable in our everyday lives. One leading example is the recent trend in secure group messaging (Zoom, Signal, WhatsApp, and so on) to handle large groups – up to 50 000 users according to the IETF draft of the Message Layer Security (MLS) architecture [38, Section 3.1]. The scenario is that users in a system, each holding their public and secret key, frequently exchange messages with a group of users. More than often, the solution adopted is the trivial approach of individually encrypting the same message M using the public keys associated with the respective recipients in the group.[4] However, this trivial approach makes the required *bandwidth* and *computational costs* grow by a factor N (where N is the number of recipients), compared to sending a message to a single recipient. Therefore, as the number of recipients increases, this trivial solution has poor scalability.

[4] To be more precise, it is common to rely on the KEM/DEM framework [19, 22] to lower the reliance on the more inefficient public key cryptography.

An additional motivation for lowering the bandwidth and computational costs is the current phase of gradual transition towards *post-quantum* cryptography — a type of cryptography that is known to be resilient against quantum adversaries. Most, if not all, post-quantum secure schemes are known to incur bandwidth and/or computational overheads compared to classical schemes. For example, all key encapsulation mechanisms (KEMs) still considered for standardization by NIST require an order of magnitude more bandwidth than ECDH [9] at a comparable classical security level. Therefore, lowering the cost of communication with multiple recipients even when the number of recipients N is only moderately large, say $N \geq 10$, will already be of value.

Multi-Recipient Key Encapsulation Mechanism (mKEM), coined by Smart [40][5], is a primitive designed with the above motivations in mind. On a high level, an mKEM is like a standard KEM that securely sends *the same* session key K to a group of recipients. Subsequently, the sender transmits *a single* ciphertext to all the recipients by encrypting the message M using K as a secret key for a secret-key encryption scheme. The latter procedure corresponds to the standard DEM. The main requirement that makes mKEM appealing is that the bandwidth and computational resources required to send the session key K are less than those required when individually encrypting K using the recipients' public keys. To be precise, we can trivially construct an mKEM from any public-key encryption (PKE) scheme by encrypting the same session key K with respect to all the recipients' public keys. However, this trivial construction will be as inefficient as the aforementioned trivial solution (modulo the efficient DEM component), and therefore, the main goal for mKEM is to offer a more efficient alternative.

Due to its practically appealing and theoretically interesting nature, the study of mKEM has attracted much attention, e.g., [8,24,26,33,35,42]. Also, similar variants of mKEM, such as *multi-message* multi-recipient *public-key encryption* [11–13, 33], have been considered prior to mKEM with similar motivations in mind, and have illustrated the importance of investigating the multi-recipient settings. As a consequence, by now many exciting results regarding mKEMs have appeared. However, we like to point out *three* unsatisfactory issues remaining with burdening the current state of affairs. First, to the best of our knowledge, all the literature on mKEMs is based on classical assumptions (e.g., Diffie-Hellman type assumptions) which are believed to not endure quantum adversaries. We are aware of one recent work [17] that claims the construction of an IND-CCA secure mKEM from the learning parity with noise (LPN) assumption, which is believed to be quantumly secure. However, while going over their results, we noticed that their scheme is insecure since there is a trivial break in their claimed IND-CCA security. In particular, the ciphertexts are easily malleable. Second, earlier works such as [8, 24, 35] provide a somewhat generic construction of mKEM from a (single-recipient) PKE, but require the underlying PKE to satisfy rather specific properties that seems somewhat tailored to classical Diffie-Hellman type assumptions. For instance, [8] requires a notion of *weak reproducibility*, which in-

[5] We note that very similar variants of mKEM have been considered prior to this work [11–13, 33]. More details follow.

formally states that there is an efficient procedure to re-randomize a ciphertext under a certain public key to a ciphertext under another public key. Unfortunately, such properties are not known to exist for post-quantum assumptions, such as lattice-based assumptions. Therefore, we still do not have a truly general framework for constructing mKEMs from standard building blocks. Here, "standard" building blocks mean blocks that are potentially instantiable from many hardness assumptions.

Summarizing thus far, the first question we are interested in this work is:

> *(Theoretical Question) Are there any simple and efficient generic constructions of* mKEM *that can be based on versatile assumptions, including post-quantum assumptions?*

The third issue, which is orthogonal to the above concerns, is that all previous works on mKEM do not come with any implementations. Notably, most literature only points out the efficiency gain in a rather theoretical manner and does not provide comparisons with the trivial solution (i.e., running KEM in parallel). Since these gains depend on the concrete mKEM implementation and also on the choice of KEM used in the trivial solution, the benefit of using an mKEM is unclear without proper comparison. Considering the practical oriented nature of mKEM, we believe understanding the concrete gain of using an mKEM instead of using the trivial solution would help in illustrating the practical relevance of this primitive and in providing insight on when to use an mKEM.

Therefore, the second question we are interested in this work is:

> *(Practical Question) What is the concrete gain of using an* mKEM *compared to the trivial solution? What are the concrete applications of* mKEM*s?*

1.1 Our Contributions and Techniques

Theoretical Contribution. We provide a new simple and efficient generic construction of an IND-CCA secure multi-recipient KEM (mKEM) from any IND-CPA secure multi-recipient PKE (mPKE).[6] The construction is proven secure in the classical *and* quantum random oracle model ((Q)ROM). Here, mPKE is a variant of mKEM where a user can encrypt any same message M (rather than a random session key K) to multiple recipients. We then show that IND-CPA secure mPKEs can be constructed very easily from most assumptions known to imply standard PKEs (including classical Diffie-Hellman type assumptions). The construction of an IND-CPA secure mPKE is in most cases a simple modification of a standard IND-CPA secure PKE to the multi-recipient setting. Concretely, we show how to construct mPKEs based on lattices and isogenies. Compared to previous works [8, 24, 35] which provide some types of generic constructions of mKEM, ours require an mPKE whereas they only require a single-recipient PKE.

[6] As standard in practice, we consider indistinguishability under chosen ciphertext attacks (IND-CCA) to be the default security requirement on our resulting scheme.

However, we only require very natural properties from the underlying mPKE, such as IND-CPA. Considering that our mPKE can be instantiated with diverse assumptions (including but not limited to post-quantum assumptions) in a very natural way from standard PKEs, we believe our generic construction to be more versatile and handy than previous ones. We point out that our mKEM achieves both implicit and explicit rejection.

Moreover, we introduce a new notion of *recipient anonymity* which we believe to be of independent interest. The notion captures the fact that the ciphertext does not leak the set of intended group members or recipients. We provide a mild additional property for the underlying IND-CPA secure mPKE, under which our above generic construction naturally implies a recipient-anonymous IND-CCA secure mKEM. Our lattice and isogeny-based instantiations satisfy the extra property without any modification. An overview of our generic construction is provided in the following section.

Practical Contribution. An immediate consequence of our theoretical contribution is that it opens the door to a large number of post-quantum instantiations of mKEM. A natural next step is to study these mKEM instantiations at a practical level and compare them to the trivial solution of running standard KEMs in parallel. Doing this work is precisely one of our practical contributions. As it turns out, at least 9 post-quantum schemes are compatible with our construction of mKEM: 7 lattice-based NIST candidates, the only isogeny-based NIST candidate SIKE, and the CSIDH scheme. We performed a systematic study of the bandwidth efficiency and found that for all of these schemes, our mKEM variants are more compact than the trivial solution with the original schemes by *at least one order of magnitude* (for a clearly defined metric). In addition, for a subset of these 9 schemes (CSIDH, FrodoKEM, Kyber, SIKE), we implemented their mKEM counterparts and compared their performance (cycle count). We found our mKEM variants to be (asymptotically) faster than the trivial solution with original schemes by factors ranging from 1.92 to more than 35.

Additionally, we show that we can use the mKEM primitive for the TreeKEM protocol obtaining significant bandwidth savings. To give some context, the importance of TreeKEM could be best understood by looking at its parent protocol, MLS [10,38], a IETF draft for secure (group) messaging. MLS has gained considerable industrial traction and has attracted a fair amount of academic scrutiny. TreeKEM constitutes the cryptographic backbone of MLS, as well as its main bottleneck in bandwidth and computational efficiency. Indeed, given N users, it requires each of them to compute and send $O(\log N)$ ciphertexts at regular intervals. We highlight a simple but powerful interplay between TreeKEM and mKEM, and show that by applying our technique we can reduce communication cost by a factor between 1.8 and 4.2 compared to using standard KEMs.

Our Techniques: Generic Construction of IND-CCA secure mKEM. On a high level, our generic construction can be seen as a generalization of the Fujisaki-Okamoto (FO) transform [23]. The FO transform (roughly) converts any IND-CPA secure PKE into an IND-CCA secure KEM. There are several variants of

the FO transform and most of the variants are secure in the ROM [18,22,25,37] and/or QROM [15,25,29–32,39,41,43]. The high-level construction is as follows: to encrypt, we sample a random message $\mathsf{M} \leftarrow \mathcal{M}$ and derive randomness for the underlying encryption algorithm of the PKE by hashing M with a hash function G modeled as a (Q)RO. That is, $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M}; \mathsf{G}(\mathsf{M}))$. The session key is then set as $\mathsf{K} := \mathsf{H}(\mathsf{M})$, where H is another hash function modeled as a (Q)RO. To decrypt, we first decrypt $\mathsf{M}' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$ and then only accept $\mathsf{K} = \mathsf{H}(\mathsf{M}')$ if M' re-encrypts back to ct, that is, we check $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M}'; \mathsf{G}(\mathsf{M}'))$. Although the actual proof is rather complicated, intuitively, it achieves $\mathsf{IND\text{-}CCA}$ security since the adversary must have queried G to have constructed a valid ciphertext ct. Therefore, in the ROM, to answer a decapsulation-oracle query, the simulator runs through all the messages that have been queried to G to check if any of them re-encrypts to ct. Since the simulator no longer requires sk to simulate the decapsulation oracle, we can invoke the $\mathsf{IND\text{-}CPA}$ security of the underlying PKE.

Our idea is to generalize the FO transform to the mPKE/mKEM setting. At first glance, this may seem to not work. Indeed, an mPKE typically comes with a *multi*-encryption algorithm with the following syntax: $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M}; \mathsf{r}) \rightarrow \mathbf{ct}$, where $\mathbf{ct}$ is targeted to the set of N recipients with public keys $(\mathsf{pk}_i)_{i\in[N]}$. There is also an extraction algorithm mExt which takes as input an index $i \in [N]$ and $\mathbf{ct}$, and outputs the ciphertext component ct_i targeted to the i-th recipient, say R_i, holding pk_i. Recipient R_i can then run the decryption algorithm on ct_i using its secret key sk_i. The reason why the FO transform cannot be directly applied to mPKE becomes clear. Assume $\mathbf{r} = \mathsf{G}(\mathsf{M})$ and that recipient R_i decrypted to M. Then, to check validity of ct_i, R_i must re-encrypt the *entire* ciphertext $\mathbf{ct}$ by running $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M}; \mathsf{r})$. Therefore, the decapsulation time will depend on N, which is highly undesirable.

To get around this issue, in this work we consider a slight variant of mPKE with a *decomposable* flavor. Informally, a decomposable multi-encryption algorithm mEnc takes randomness of the form $\mathsf{r} = (\mathsf{r}_0, \mathsf{r}_1, \cdots, \mathsf{r}_N)$ as input, and creates a public-key-*independent* ciphertext $\mathsf{ct}_0 \leftarrow \mathsf{mEnc}^{\mathsf{i}}(\mathsf{r}_0)$ and public-key-*dependent* ciphertexts $\widehat{\mathsf{ct}}_i \leftarrow \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i)$. The resulting ciphertext for recipient R_i is then $\mathsf{ct}_i = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$. We view this as a natural formalization of mPKE as it is satisfied by all the mPKE constructions that we are aware of. Moreover, this feature is desirable in practice as it allows to parallelize part of the encryption algorithm. Now, to perform the FO transform, we derive $\mathsf{r}_0 = \mathsf{G}(\mathsf{M})$ and $\mathsf{r}_i = \mathsf{G}(\mathsf{pk}_i, \mathsf{M})$. It is evident that R_i can re-encrypt and check the validity of its ciphertext. Notably, the decapsulation time is now independent of N. With this new formalization, the proof in the (classical) ROM follows in a straightforward manner (with minor modification) from the standard FO transform [25].

However, the security proof of our mKEM in the *quantum* ROM (QROM) requires more work. Prior proof strategies in the QROM for standard $\mathsf{IND\text{-}CCA}$ secure KEMs based on the FO transform – which fix the description of the QROM at the outset of the game [15,25,29–31,39,41] – seem to be an ill fit for mPKE. This is because in the multi-recipient setting, the decapsulation oracle is required

to output a different (implicit) rejection value for each of the users when the ciphertext is invalid, and to output the same session key K when the ciphertext is valid. Due to this discrepancy between invalid and valid ciphertexts (i.e., the former requires to output different random values, whereas the latter requires to output the same random value), previous proof techniques that always output random values fail. Note that in the single-user setting, regardless of the ciphertext being valid or invalid, the decapsulation oracle could output random values without being detected by the adversary, and hence, this obstacle was absent. To overcome this, we use the recently introduced *compressed oracles* technique [43]. This allows the simulator to perform *lazy sampling* and to check the validity of the ciphertext submitted to the decapsulation oracle without interfering with the adversary's state. Although the high-level structure of the proof is similar to the classical case, much subtle care is required in the QROM case as the simulator must not disturb the adversary's state. We note that Zhandry [43] showed security of one variant of the FO transform which converts a *perfectly* correct IND-CPA secure PKE to an IND-CCA secure PKE.

2 Preliminaries

2.1 Hard Problems for Lattices

For any natural number d and q, let R_q denote the ring $\mathbb{Z}[X]/(q, X^d + 1)$. The learning with errors (LWE) problem is defined below.

Definition 1 (Learning with Errors (LWE)). *Let d, q, n_1, n_2, n_3 be natural numbers, and D_{s} and D_{e} be distributions over R_q. We say that the advantage of algorithm $\mathcal{A}$ in solving the (decisional) $\mathsf{LWE}_{n_1,n_2,n_3}$ problem over the ring R_q is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{LWE}}_{n_1,n_2,n_3}(\mathcal{A}) := \big|\ &\Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{S} \leftarrow D_{\mathsf{s}}^{n_2 \times n_3}, \mathbf{E} \leftarrow D_{\mathsf{e}}^{n_1 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{AS} + \mathbf{E})] \\ &- \Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{B} \leftarrow R_q^{n_1 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{B})]\big| .\end{aligned}$$

We say the $\mathsf{LWE}_{n_1,n_2,n_3}$ problem is hard if, for any (possibly quantum) efficient adversary $\mathcal{A}$, its advantage is negligible.

We also consider a variant of the LWE problem, called learning with rounding (LWR) problem [7], where the least significant bits are removed. The benefit of this variant is that we no longer require to sample the noise, as it is removed. Below the function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$, where $q > p \geq 2$, is defined as $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \mod p$. The definition of the LWR problem follows.

Definition 2 (Learning with Rounding (LWR)). *Let d, p, q, n_1, n_2, n_3 be natural numbers such that $q > p$, and D_{s} a distributions over R_q. We say that the advantage of algorithm $\mathcal{A}$ in solving the (decisional) $\mathsf{LWR}_{n_1,n_2,n_3}$ problem over the rings R_p and R_q is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{LWR}}_{n_1,n_2,n_3}(\mathcal{A}) := \big|\ &\Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{S} \leftarrow D_{\mathsf{s}}^{n_2 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \lfloor \mathbf{AS} \rceil_p)] \\ &- \Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{B} \leftarrow R_p^{n_1 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{B})]\big| .\end{aligned}$$

We say the $\mathsf{LWR}_{n_1,n_2,n_3}$ problem is hard if, for any (possibly quantum) efficient adversary $\mathcal{A}$, its advantage is negligible.

2.2 Hard Problems for Isogenies

In the following sections we propose two different isogeny-based schemes: one stemming from the SIDH key exchange [21] and the other from the CSIDH key exchange [16]. Both key exchanges share common mathematical tools, but several technical differences make them, and their descendants, substantially different. As a consequence, schemes in the SIDH family rely on hardness assumptions different from those used for schemes in the CSIDH family. Our schemes make no exception, as they use distinct security assumptions.

SIDH-based assumption. Let p be an odd prime of the form $2^{e_2}3^{e_3} - 1$, with $e_2, e_3 \in \mathbb{N}$ and $2^{e_2} \approx 3^{e_3}$. For a supersingular elliptic curve E over $\mathbb{F}_{p^2}$ we will denote by $B_2 = \{P_2, Q_2\}$ and $B_3 = \{P_3, Q_3\}$ bases for $E[2^{e_2}]$ and $E[3^{e_3}]$, respectively. Under the hypothesis that $|E(\mathbb{F}_{p^2})| = (2^{e_2}3^{e_3})^2$, both torsion subgroups $E[2^{e_2}]$ and $E[3^{e_3}]$ are contained in $E(\mathbb{F}_{p^2})$. Given the curve E and $s \in \mathbb{Z}_{2^{e_2}}$, by $\mathsf{pk}_2(s)$ we denote the tuple $(E/\langle R_2 = P_2 + [s]Q_2\rangle, \phi_{\langle R_2\rangle}(P_3), \phi_{\langle R_2\rangle}(Q_3))$, where $\phi_{\langle R_2\rangle}$ is the isogeny from E having kernel $\langle R_2\rangle$. Analogously, for $r \in \mathbb{Z}_{3^{e_3}}$ we define $\mathsf{pk}_3(r)$ as $(E/\langle R_3 = P_3 + [r]Q_3\rangle, \phi_{\langle R_3\rangle}(P_2), \phi_{\langle R_3\rangle}(Q_2))$.

The security of our scheme relies on a decisional variant, named SSDDH [21], of the SSCDH assumption. The latter is used by one of NIST second-round candidate KEMs, called SIKE [28], which is deduced from the key exchange SIDH.

Definition 3 (Supersingular Decisional Diffie-Hellman (SSDDH)). *Let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$ such that $|E(\mathbb{F}_{p^2})| = (2^{e_2}3^{e_3})^2$. We say that the advantage of algorithm $\mathcal{A}$ in solving the $\mathsf{SSDDH}_{p,E,B_2,B_3}$ problem is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SSDDH}}_{p,E,B_2,B_3}(\mathcal{A}) := |\ &\Pr[s \leftarrow \mathbb{Z}_{2^{e_2}}, r \leftarrow \mathbb{Z}_{3^{e_3}} : \\ &\qquad 1 \leftarrow \mathcal{A}(\mathsf{pk}_2(s), \mathsf{pk}_3(r), E/\langle P_2 + [s]Q_2, P_3 + [r]Q_3\rangle)] \\ &- \Pr[(s, s') \leftarrow (\mathbb{Z}_{2^{e_2}})^2, (r, r') \leftarrow (\mathbb{Z}_{3^{e_3}})^2 : \\ &\qquad 1 \leftarrow \mathcal{A}(\mathsf{pk}_2(s), \mathsf{pk}_3(r), E/\langle P_2 + [s']Q_2, P_3 + [r']Q_3\rangle)]|\,.\end{aligned}$$

We say the $\mathsf{SSCDH}_{p,E,B_2,B_3}$ problem is hard if, for any (possibly quantum) efficient adversary $\mathcal{A}$, its advantage is negligible.

CSIDH-based assumption. The CSIDH key exchange works with supersingular elliptic curves and isogenies as well, but they are defined over a prime field $\mathbb{F}_p$. Despite offering weaker security guarantees than SIDH, CSIDH enjoys a simpler design based on the action of a group G on a set of curves. The simplicity of its design makes it easy to use CSIDH for constructing cryptographic primitives. Details on the CSIDH assumption we use are provided in the full version.

3 Multi-Recipient PKE and KEM

3.1 Decomposable Multi-Recipient Public Key Encryption

Definition 4 (Decomposable Multi-Recipient Public Key Encryption). *A* (single-message) decomposable multi-recipient public key encryption (mPKE) *over a message space* $\mathcal{M}$ *and ciphertext spaces* $\mathcal{C}$ *and* $\mathcal{C}_{\mathsf{single}}$ *consists of the following five algorithms* $\mathsf{mPKE} = (\mathsf{mSetup}, \mathsf{mGen}, \mathsf{mEnc}, \mathsf{mExt}, \mathsf{mDec})$:

- $\mathsf{mSetup}(1^\kappa) \to \mathsf{pp}$: *The setup algorithm on input the security parameter* 1^κ *outputs a public parameter* pp.
- $\mathsf{mGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: *The key generation algorithm on input a public parameter* pp *outputs a pair of public key and secret key* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_1, \cdots, \mathsf{r}_N) \to \mathbf{ct} = (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_i)_{i\in[N]})$: *The (decomposable) encryption algorithm running with randomness* $(\mathsf{r}_0, \mathsf{r}_1, \cdots, \mathsf{r}_N)$, *splits into a pair of algorithms* $(\mathsf{mEnc}^{\mathsf{i}}, \mathsf{mEnc}^{\mathsf{d}})$:
  - $\mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0) \to \mathsf{ct}_0$: *On input a public parameter* pp *and randomness* r_0, *it outputs a (public key* Independent*) ciphertext* ct_0.
  - $\mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i) \to \widehat{\mathsf{ct}}_i$: *On input a public parameter* pp, *a public key* pk_i, *a message* $\mathsf{M} \in \mathcal{M}$, *and randomness* $(\mathsf{r}_0, \mathsf{r}_i)$, *it outputs a (public key* Dependent*) ciphertext* $\widehat{\mathsf{ct}}_i$.
- $\mathsf{mExt}(i, \mathbf{ct}) \to \mathsf{ct}_i = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ *or* $\perp$: *The deterministic extraction algorithm on input an index* $i \in \mathbb{N}$ *and a (multi-recipient) ciphertext* $\mathbf{ct} \in \mathcal{C}$, *outputs either a (single-recipient) ciphertext* $\mathsf{ct}_i = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i) \in \mathcal{C}_{\mathsf{single}}$ *or a special symbol* $\perp_{\mathsf{Ext}}$ *indicating extraction failure.*
- $\mathsf{mDec}(\mathsf{sk}, \mathsf{ct}_i) \to \mathsf{M}$ *or* $\perp$: *The deterministic decryption algorithm on input a secret key* sk *and a ciphertext* $\mathsf{ct}_i \in \mathcal{C}_{\mathsf{single}}$, *outputs either* $\mathsf{M} \in \mathcal{M}$ *or a special symbol* $\perp \notin \mathcal{M}$.

Although we can consider *non-decomposable* multi-recipient PKEs, we only focus on decomposable schemes as they are compatible with the Fujisaki-Okamoto (FO) transform [23]. Informally, the FO transform relies on the recipient being able to recover the encryption randomness from the ciphertext and to check validity of the ciphertext by re-encrypting with the recovered randomness. Therefore, in the multi-recipient setting, if we do not impose decomposable encryption, then the recipient may require all the public keys that were used in constructing $\mathbf{ct}$ to be able to re-encrypt. However, this is clearly undesirable since the decryption time may now depend on the number of public keys used to encrypt, and furthermore, the size of the ciphertext will grow by appending all the public keys used. Therefore, in this paper, when we say mPKE, we always assume it is decomposable. We require the following properties from a mPKE.

Definition 5 (Correctness). *A* mPKE *is* δ*-correct if*

$$\delta \geq \mathbb{E}\left[\max_{\mathsf{M}\in\mathcal{M}} \Pr\left[\begin{array}{c} \mathsf{ct}_0 \leftarrow \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}), \widehat{\mathsf{ct}} \leftarrow \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}, \mathsf{M}) : \\ \mathsf{M} \neq \mathsf{mDec}(\mathsf{sk}, (\mathsf{ct}_0, \widehat{\mathsf{ct}})) \end{array}\right]\right], \tag{1}$$

where the expectation is taken over $\mathsf{pp} \leftarrow \mathsf{mSetup}(1^\kappa)$ *and* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{mGen}(\mathsf{pp})$.

We also define the notion of *well-spreadness* [23] which states informally that the ciphertext has high min-entropy.

Definition 6 (γ-Spreadness). *Let* mPKE *be a decomposable multi-recipient* PKE *with message space* $\mathcal{M}$ *and ciphertext spaces* $\mathcal{C}$ *and* $\mathcal{C}_{\mathsf{single}}$*. For all* $\mathsf{pp} \in \mathsf{Setup}(1^\kappa)$*, and* $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{pp})$*, define*

$$\gamma(\mathsf{pp}, \mathsf{pk}) := -\log_2 \left(\max_{\mathsf{ct} \in \mathcal{C}_{\mathsf{single}}, \mathsf{M} \in \mathcal{M}} \Pr_{r_0, \mathsf{r}} \left[\mathsf{ct} = \left(\mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; r_0), \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}, \mathsf{M}; r_0, \mathsf{r}) \right) \right] \right).$$

We call mPKE γ*-spread if* $\mathbb{E}[\gamma(\mathsf{pp}, \mathsf{pk})] \geq \gamma$*, where the expectation is taken over* $\mathsf{pp} \leftarrow \mathsf{mSetup}(1^\kappa)$ *and* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{mGen}(\mathsf{pp})$*.*

Finally, we define the notion of indistinguishability of chosen plaintext attacks ($\mathsf{IND\text{-}CPA}$) for mPKE.

Definition 7 ($\mathsf{IND\text{-}CPA}$). *Let* mPKE *be a decomposable multi-recipient* PKE *with message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*. We define* $\mathsf{IND\text{-}CPA}$ *by a game illustrated in Fig. 1 and say the (possibly quantum) adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ wins *if the game outputs* 1*. We define the advantage of* $\mathcal{A}$ *against* $\mathsf{IND\text{-}CPA}$ *security of* mPKE *parameterized by* $N \in \mathbb{N}$ *as* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|$*.*

Remark 1 (Insider corruption). We point out that insider corruptions for mPKE are not considered [8, 40]. This is because if an adversary obtains a secret key corresponding to any of the public keys used to encrypt, then it can trivially recover the encrypted message.

Remark 2 (Inefficient mPKE *from any standard (single-recipient)* PKE*).* Our definition of mPKE captures the trivial solution of sending different ciphertexts obtained with a standard single-recipient PKE to multiple recipients. That is, independently encrypting the same message to all recipients using their respective public keys. In the above syntax of mPKE, this amounts to setting $\mathsf{mEnc}^{\mathsf{i}}$ as a null function and setting r_0 as an empty string. Also, mExt will simply pick the relevant ciphertext component for the particular recipient. Therefore, in the context of ciphertext compression, the goal is to obtain a mPKE with better efficiency/ciphertext-size compared to this trivial method.

Remark 3 (Number of recipients). In general, the number of recipients $N = \mathsf{poly}(\kappa)$ can be chosen arbitrary by the sender (or adversary). Some schemes may require an upper bound on N since the concrete provably-secure parameters may have a dependance on N, e.g., the reduction loss degrades by a factor of $1/N$. Our proposal does not require such an upper bound since N only shows up in a statistical manner, and so we can handle large N, say $N = 2^{15}$, without having any large impact on the concrete parameter choice.

3.2 Multi-Recipient Key Encapsulation Mechanism

Definition 8 (Multi-Recipient Key Encapsulation Mechanism). *A* (single-message) multi-recipient key encapsulation mechanism (mKEM) *over a key space* $\mathcal{K}$ *and ciphertext space* $\mathcal{C}$ *consists of the following five algorithms* $\mathsf{mKEM} = (\mathsf{mSetup}, \mathsf{mGen}, \mathsf{mEncaps}, \mathsf{mExt}, \mathsf{mDecaps})$:

GAME IND-CPA

$\mathsf{pp} \leftarrow \mathsf{mSetup}(1^\kappa)$
for $i \in [N]$ **do**
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mGen}(\mathsf{pp})$
$(\mathsf{M}_0^*, \mathsf{M}_1^*, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]})$
$b \leftarrow \{0,1\}$
$\mathbf{ct}^* \leftarrow \mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M}_b^*)$
$b' \leftarrow \mathcal{A}_2(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathbf{ct}^*, \mathsf{state})$
return $[b = b']$

GAME IND-CCA

$\mathsf{pp} \leftarrow \mathsf{mSetup}(1^\kappa)$
for $i \in [N]$ **do**
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mGen}(\mathsf{pp})$
$(\mathsf{K}_0^*, \mathbf{ct}^*) \leftarrow \mathsf{mEncaps}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]})$
$\mathsf{K}_1^* \leftarrow \mathcal{K}$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{D}}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathbf{ct}^*, \mathsf{K}_b^*)$
return $[b = b']$

Decapsulation Oracle $\mathcal{D}(i, \mathsf{ct})$

$\mathsf{ct}_i^* := \mathsf{mExt}(i, \mathbf{ct}^*)$
if $\mathsf{ct} = \mathsf{ct}_i^*$ **then**
  return $\perp$
$\mathsf{K} := \mathsf{mDecaps}(\mathsf{sk}_i, \mathsf{ct})$
return K

Fig. 1: IND-CPA of mPKE and IND-CCA of mKEM.

- $\mathsf{mSetup}(1^\kappa) \to \mathsf{pp}$: *The setup algorithm on input the security parameter* 1^κ *outputs a public parameter* pp.
- $\mathsf{mGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: *The key generation algorithm on input a public parameter* pp *outputs a pair of public key and secret key* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{mEncaps}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}) \to (\mathsf{K}, \mathbf{ct})$: *The encapsulation algorithm on input a public parameter* pp*, and* N*public keys* $(\mathsf{pk}_i)_{i\in[N]}$*, outputs a key* K *and a ciphertext* $\mathbf{ct}$.
- $\mathsf{mExt}(i, \mathbf{ct}) \to \mathsf{ct}_i$: *The deterministic extraction algorithm on input an index* $i \in \mathbb{N}$ *and a ciphertext* $\mathbf{ct}$*, outputs either* ct_i *or a special symbol* $\perp_{\mathsf{Ext}}$ *indicating extraction failure.*
- $\mathsf{mDecaps}(\mathsf{sk}, \mathsf{ct}_i) \to \mathsf{K}$ *or* $\perp$: *The deterministic decryption algorithm on input a secret key* sk *and a ciphertext* ct_i*, outputs either* $\mathsf{K} \in \mathcal{K}$ *or a special symbol* $\perp \notin \mathcal{K}$.

Definition 9 (Correctness). *A* mKEM *is* δ_N*-correct if*

$$\delta_N \geq \Pr\left[(\mathsf{K}, \mathbf{ct}) \leftarrow \mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}), (\mathsf{ct}_i \leftarrow \mathsf{mExt}(i, \mathbf{ct}))_{i\in[N]} \right. \\ \left. : \exists i \in [N] \text{ s.t. } \mathsf{K} \neq \mathsf{mDec}(\mathsf{sk}, \mathsf{ct}_i)\right],$$

where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{mSetup}$ *and* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mGen}(\mathsf{pp})$ *for all* $i \in [N]$.

We define the notion of indistinguishability of chosen ciphertext attacks (IND-CCA) for mKEM.

Definition 10 (IND-CCA). *Let* mKEM *be a multi-recipient* KEM*. We define* IND-CCA *by a game illustrated in Fig. 1 and say the (possibly quantum) adversary* $\mathcal{A}$ *(making only classical decapsulation queries to* $\mathcal{D}$*)* wins *if the game outputs* 1*. We define the advantage of* $\mathcal{A}$ *against* IND-CCA *security of* mKEM *parameterized by* $N \in \mathbb{N}$ *as* $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|$.

3.3 Recipient Anonymity for mPKE and mKEM

In many practical scenarios, it is often convenient to have an additional guarantee of recipient anonymity, which stipulates that the ciphertext does not leak any information about the set of intended recipients. Informally, we say mPKE (mKEM) is IND-Anon-CPA (IND-Anon-CCA) if there exists a fake encryption (encapsulation) algorithm $\overline{\mathsf{mEnc}}$ ($\overline{\mathsf{mEncaps}}$), which takes as input only the number of recipients and outputs a fake ciphertext indistinguishable from an honestly generated ciphertext. The definition is formally provided in the full version.

4 FO Transform: (IND-CPA mPKE) $\Rightarrow$ (IND-CCA mKEM)

4.1 Generic Construction via FO Transform

We provide a generic transformation of an IND-CPA secure mPKE to an IND-CCA secure mKEM following the (generalized) Fujisaki-Okamoto transform. This is illustrated in Fig. 2. The scheme provides *implicit* rejection as opposed to *explicit* rejection, where in the latter type, the decapsulation algorithm outputs a special symbol $\perp$ to explicitly indicate decapsulation failure. We discuss later how to tweak our scheme to get explicit rejection with no additional cost. In Fig. 2, $\mathsf{G}_1, \mathsf{G}_2, \mathsf{H}, \mathsf{H}'$ are hash functions modeled as random oracles in the security proof. They can be simulated by a single random oracle by using appropriate domain separation. Finally, we include an ℓ-bit seed to perform implicit rejection by viewing $\mathsf{H}'(\mathsf{seed}, \cdot)$ as a pseudorandom function in the (Q)ROM.

$\underline{\mathsf{mSetup}(1^\kappa)}$
1: $\mathsf{pp} \leftarrow \mathsf{mSetup}^{\mathsf{p}}(1^\kappa)$
2: **return** pp

$\underline{\mathsf{mGen}(\mathsf{pp})}$
1: $(\mathsf{pk}, \mathsf{sk}^{\mathsf{p}}) \leftarrow \mathsf{mGen}^{\mathsf{p}}(\mathsf{pp})$
2: $\mathsf{seed} \leftarrow \{0,1\}^\ell$
3: $\mathsf{sk} := (\mathsf{sk}^{\mathsf{p}}, \mathsf{seed})$
4: **return** $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{mExt}(i, \mathbf{ct})}$
1: $\mathsf{ct}_i \leftarrow \mathsf{mExt}^{\mathsf{p}}(i, \mathbf{ct})$
2: **return** ct_i

$\underline{\mathsf{mEncaps}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]})}$
1: $\mathsf{M} \leftarrow \mathcal{M}$
2: $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$
3: **for** $i \in [N]$ **do**
4: $\quad \widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$
5: $\mathsf{K} := \mathsf{H}(\mathsf{M})$
6: **return** $(\mathsf{K}, \mathbf{ct} := (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_i)_{i\in[N]}))$

$\underline{\mathsf{mDecaps}(\mathsf{sk}, \mathsf{ct})}$
1: $\mathsf{sk} := (\mathsf{sk}^{\mathsf{p}}, \mathsf{seed})$
2: $\mathsf{M} := \mathsf{mDec}(\mathsf{sk}^{\mathsf{p}}, \mathsf{ct})$
3: **if** $\mathsf{M} = \perp$ **then**
4: $\quad$ **return** $\mathsf{K} := \mathsf{H}'(\mathsf{seed}, \mathsf{ct})$
5: $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$
6: $\widehat{\mathsf{ct}} := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}, \mathsf{M}))$
7: **if** $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ **then**
8: $\quad$ **return** $\mathsf{K} := \mathsf{H}'(\mathsf{seed}, \mathsf{ct})$
9: **else**
10: $\quad$ **return** $\mathsf{K} := \mathsf{H}(\mathsf{M})$

Fig. 2: An IND-CCA secure mKEM from a decomposable IND-CPA secure $\mathsf{mPKE} = (\mathsf{mSetup}^{\mathsf{p}}, \mathsf{mGen}^{\mathsf{p}}, \mathsf{mEnc} = (\mathsf{mEnc}^{\mathsf{i}}, \mathsf{mEnc}^{\mathsf{d}}), \mathsf{mExt}^{\mathsf{p}}, \mathsf{mDec})$. We include the superscript $^{\mathsf{p}}$ to make the code more readable.

The following theorem classically and quantumly reduce the IND-CCA security of mKEM to the IND-CPA security of mPKE, where the classical reduction is tight. The proof for each theorem is provided in the subsequent sections.

Theorem 1 (Classical: IND-CPA mPKE $\Rightarrow$ IND-CCA mKEM). *Assume* mPKE *with message space $\mathcal{M}$ is δ-correct and γ-spread. Then, for any classical PPT* IND-CCA *adversary $\mathcal{A}$ issuing at most $q_{\mathcal{D}}$ queries to the decapsulation oracle $\mathcal{D}$, a total of at most q_{G} queries to G_1 and G_2, and at most $q_{\mathsf{H}}, q'_{\mathsf{H}}$ queries to H and H', there exists a classical PPT adversary $\mathcal{B}_{\mathsf{IND}}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}}) + (2q_{\mathsf{G}} + q_{\mathcal{D}} + 2) \cdot \delta + q_{\mathcal{D}} \cdot 2^{-\gamma} + \frac{(q_{\mathsf{G}} + q_{\mathsf{H}})}{|\mathcal{M}|} + q'_{\mathsf{H}} \cdot N \cdot 2^{-\ell}.$$

where the running time of $\mathcal{B}_{\mathsf{IND}}$ is about that of $\mathcal{A}$, and ℓ is the number of bits of the seed composing a private key.

Theorem 2 (Quantum: IND-CPA mPKE $\Rightarrow$ IND-CCA mKEM). *Assume* mPKE *with message space $\mathcal{M}$ is δ-correct and γ-spread. Then, for any quantum PT* IND-CCA *adversary $\mathcal{A}$ issuing at most $q_{\mathcal{D}}$ classical queries to the decapsulation oracle $\mathcal{D}$, a total of at most q_{G} quantum queries to G_1 and G_2, and at most $q_{\mathsf{H}}, q'_{\mathsf{H}}$ quantum queries to H and H', there exists a quantum PT adversary $\mathcal{B}_{\mathsf{IND}}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N}(\mathcal{A}) \leq \sqrt{2 \cdot (q_{\mathsf{G}} + 1) \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}})} + \frac{4(q_{\mathsf{G}} + 1)}{\sqrt{|\mathcal{M}|}} + 12 \cdot (q_{\mathsf{G}} + q_{\mathcal{D}} + 1)^2 \cdot \delta + q_{\mathcal{D}} \cdot (9\sqrt{2^{-\gamma}} + 2^{\mu-2}) + q'_{\mathsf{H}} \cdot N \cdot 2^{\frac{-\ell+1}{2}},$$

where the running time of $\mathcal{B}_{\mathsf{IND}}$ is about that of $\mathcal{A}$, ℓ is the number of bits of the seed composing a private key, and $\mu = \max_{(\mathsf{r}_0,\mathsf{r}) \in \mathcal{R}}\{|\mathsf{r}_0|, |\mathsf{r}|\}$ where $\mathcal{R}$ is the randomness space of mPKE.

Remark 4 (Implicit vs explicit rejection). In our construction in Fig. 2, we use *implicit* rejection. That is, mDecaps does not explicitly output $\perp$ to indicate that the input ciphertext was invalid. This may be suitable in practice when we do not want to let the adversary know that decapsulation failed. However, we note that our proof is agnostic to this choice, and in particular, the same proof can be shown in case we want *explicit rejection*, where mDecaps outputs $\perp$ in case either $\mathsf{M} = \perp$ or ct is not the same as the reencrypted ciphertext $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$. Concretely, we obtain an IND-CCA secure mKEM with explicit rejection by simply outputting $\perp$ rather than outputting $\mathsf{H}'(\mathsf{seed}, \mathsf{ct})$ in Fig. 2. We emphasize that this tweak cannot be made in general since the security proofs may hinge on the fact that the adversary does not learn decapsulation failures (see [15, 39]).

4.2 Proof for Classical Case

Proof (Proof of Thm. 1). Let $\mathcal{A}$ be a classical PPT adversary against the IND-CCA security of mKEM. We upper bound its advantage by considering the following game sequence. We denote by E_i the event $\mathcal{A}$ wins in Game_i.

- Game_1: This is the real IND-CCA security game: $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N}(\mathcal{A}) = |\Pr[\mathsf{E}_1] - 1/2|$.

- Game_2: In this game, we replace the computation of $\mathsf{H}'(\mathsf{seed}_i, \cdot)$ by a random function $\widehat{\mathsf{H}}'_i(\cdot)$ in case $\mathsf{M} = \perp$ or $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ occurs when answering the decapsulation oracle with input $i \in [N]$. Here, $\widehat{\mathsf{H}}'_i(\cdot)$ is a random function that cannot be accessed by the adversary. Since this modification remains unnoticed by the adversary unless $\mathsf{H}'(\mathsf{seed}, \cdot)$ is queried for any $\mathsf{seed} \in \{\mathsf{seed}_i\}_{i\in[N]}$, we have $|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| \leq \frac{q'_{\mathsf{H}} \cdot N}{2^\ell}$.

- Game_3: In this game, we enforce that no decryption failure occurs. Namely, we modify the random oracle so that the output is distributed randomly over the space of randomness that leads to no decryption failures. By the correctness of mPKE, we have $|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| \leq (q_{\mathsf{G}} + q_{\mathcal{D}} + 1) \cdot \delta$.

Game_4 : Decap. Oracle $\mathcal{D}(i, \mathsf{ct} \neq \mathsf{ct}^*_i)$

$\mathsf{sk}_i := (\mathsf{sk}^{\mathsf{p}}_i, \mathsf{seed}_i)$
$\mathsf{M} := \mathsf{mDec}(\mathsf{sk}^{\mathsf{p}}_i, \mathsf{ct})$
if $\mathsf{M} \notin \mathcal{L}_{\mathsf{G}}$ **then**
  return $\mathsf{K} := \widehat{\mathsf{H}}'_i(\mathsf{ct})$
if $\mathsf{M} = \perp$ **then**
  return $\mathsf{K} := \widehat{\mathsf{H}}'_i(\mathsf{ct})$
$\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$
$\widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$
if $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ **then**
  return $\mathsf{K} := \widehat{\mathsf{H}}'_i(\mathsf{ct})$
else
  return $\mathsf{K} := \mathsf{H}(\mathsf{M})$

Game_5 : Decap. Oracle $\mathcal{D}(i, \mathsf{ct} \neq \mathsf{ct}^*_i)$

for $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ **do**
  $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$
  $\widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$
  if $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ **then**
    return $\mathsf{K} := \mathsf{H}(\mathsf{M})$
return $\mathsf{K} := \widehat{\mathsf{H}}_i(\mathsf{ct})$

Fig. 3: Decapsulation oracles of Game_4 and Game_5. We enforce ct is not $\mathsf{ct}^*_i := \mathsf{mExt}(i, \mathbf{ct}^*)$ at the input level for simplicity.

(The next Game_4, Game_5 and Game_6 aim to get rid of the secret keys sk_i to answer $\mathcal{A}$'s decapsulation oracle queries.)

- Game_4: In this game, we add an additional check when answering the decapsulation oracle query. This is illustrated in Fig. 3 where the red underline indicates the modification. Here, $\mathcal{L}_{\mathsf{G}}$ is a list that stores the random oracle queries made to G_1 and G_2.We have $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ if either G_1 was queried on M or G_2 was queried on $(\mathsf{pk}, \mathsf{M})$ for any pk. The only difference occurs when $\mathcal{A}$ queries a ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ such that $\mathsf{M} := \mathsf{mDec}(\mathsf{sk}^{\mathsf{p}}_i, \mathsf{ct})$ has not

been queried to the random oracles G_1 and G_2 but $\mathsf{ct}_0 = \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$ and $\widehat{\mathsf{ct}}_i = \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$. Since $\mathsf{G}_1(\mathsf{M})$ and $\mathsf{G}_2(\mathsf{pk}_i, \mathsf{M})$ are information theoretically hidden from $\mathcal{A}$, we can use γ-spreadness of mPKE to conclude $|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_4]| \leq q_{\mathcal{D}} \cdot 2^{-\gamma}$.

- Game_5: In this game, we further modify the way a decapsulation-oracle query is answered. This is illustrated in Fig. 3, where notice that we no longer require the secret keys sk_i to answer the queries. If the decapsulation oracle in Game_4 outputs $\mathsf{K} := \mathsf{H}(\mathsf{M})$, then $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ and $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ holds. Therefore, the decapsulation oracle in Game_5 outputs K as well. On the other hand, assume the decapsulation oracle in Game_5 outputs $\mathsf{K} := \mathsf{H}(\mathsf{M})$ for some $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ such that $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ where $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$ and $\widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$. Then, since we have no correctness error (due to Game_3), ct must decrypt to M. Hence, this implies that the decapsulation oracle Game_4 outputs the same K as well. Combining the arguments together, we get $\Pr[\mathsf{E}_4] = \Pr[\mathsf{E}_5]$.

- Game_6: In this game, we undo the change we made in Game_3 and alter the output of the random oracles G_1 and G_2 to be over all the randomness space. Due to the same argument as before, we have $|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_6]| \leq (q_{\mathsf{G}} + 1) \cdot \delta_N$.

(The following final Game_7 *aims to get rid of* M^* *in the challenge ciphertext.)*

- Game_7: In this game, we sample the random message $\mathsf{M}^* \leftarrow \mathcal{M}$ to be used to generate the challenge ciphertext at the beginning. We then define Query as the event that $\mathcal{A}$ queries the random oracles $\mathsf{H}(\cdot)$, $\mathsf{G}_1(\cdot)$, or $\mathsf{G}_2(\star, \cdot)$ on input M^*, where $\star$ denotes an arbitrary element. When Query occurs, we abort the game and force $\mathcal{A}$ to output a random bit. We show in the full version that $|\Pr[\mathsf{E}_6] - \Pr[\mathsf{E}_7]| \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}}) + \frac{(q_{\mathsf{G}}+q_{\mathsf{H}})}{|\mathcal{M}|}$ for some classical PPT adversary $\mathcal{B}_{\mathsf{IND}}$ with similar runtime as $\mathcal{A}$.

In Game_7, the view of the adversary is independent of the challenge bit b. Therefore, we have $\Pr[\mathsf{E}_7] = \frac{1}{2}$. This concludes the proof.

4.3 Proof for Quantum Case

The proof structure for the quantum case follows very closely the classical case. Minimal background on quantum computation is provided in the full version, and we refer for more details to other works, such as [6,20,27,43]. The main difference between our proof and prior proofs for IND-CCA secure KEM in the QROM, e.g., [15,25,29–31,39,41], is that we use the *lazy sampling* with *compressed* quantum oracles introduced in [43]. This allows the simulator to check the validity of the ciphertext submitted to the decapsulation oracle without interfering with the adversary's state. Specifically, other than how we specify and interact with the random oracle, the proof structure is essentially the same as the classical case. We refer to the full version for the full proof.

4.4 Adding Recipient Anonymity

The construction provided in Sec. 4.1 immediately give rise to a *recipient anonymous* mKEM if we additionally assume the underlying IND-CPA secure mPKE is

IND-Anon-CPA secure. In particular, we define the fake encapsulation algorithm $\overline{\mathsf{mEncaps}}$ (see Sec. 3.3) as: sample $\mathsf{K} \leftarrow \mathcal{K}$, run $\mathbf{ct} \leftarrow \overline{\mathsf{mEnc}}(\mathsf{pp}, N)$, and output $(\mathsf{K}, \mathbf{ct})$, where $\overline{\mathsf{mEnc}}$ is the fake encryption algorithm of the underlying mPKE (see Sec. 3.3). The only modification to the proofs of Thms. 1 and 2 is that we add an additional game at the end where we invoke the IND-Anon-CPA security game. Since, by the end of both proofs, the key K^* are distributed uniformly random, it remains to guarantee that $\mathbf{ct}^*$ is distributed independently of the public keys $(\mathsf{pk}_i)_{i\in[N]}$. We omit the full proof as it directly reduces from the IND-Anon-CPA security game.

5 Multi-Recipient KEM from Post-Quantum Assumptions

We provide two types of IND-CCA secure mKEM instantiations: one scheme based on lattices, and two schemes based on isogenies (in the SIDH and CSIDH setting). Specifically, we provide two types of IND-CPA secure mPKEs and use Thms. 1 and 2 to generically convert them into IND-CCA secure mKEMs in the ROM and QROM, respectively. As we see in Sec. 6, both types of instantiations are designed to fit with many of the NIST round 2 candidate (single-recipient) PKE/KEMs.

5.1 Multi-Recipient KEM from Lattices

In this section, we show that the lattice-based (single-recipient) PKE based on the Lindner-Peikert framework [34] provides a natural mPKE with the required properties. Since we are able to reuse a large part of the ciphertext for lattice-based schemes, we get a notable efficiency gain compared to the trivial mPKE/mKEM which runs PKE/KEM independently for each recipient (as discussed in Rem. 2).

The mPKE scheme based on the Lindner-Peikert framework [34] is provided in Fig. 4. Here, Encode (resp. Decode) is an efficiently computable bijective function that maps elements from the message space (resp. $R_q^{\bar{m}\times m}$) to $R_q^{\bar{m}\times m}$ (resp. message space). The details of Encode and Decode are scheme specific and not significant for this section. We show the mPKE scheme in Fig. 4 has all the properties required for applying the "multi-recipient" Fujisaki-Okamoto transform (Thms. 1 and 2). First, it is straightforward to see that we can easily set the parameters as to have δ-correctness and γ-spreadness for exponentially small δ and $2^{-\gamma}$. Moreover, practical schemes such as NIST candidates also allow for exponentially small δ and $2^{-\gamma}$. It remains to show that the Linder-Peikert framework provides not only a secure PKE but also a secure mPKE.

IND-(Anon-)CPA *Security.* It is straightforward to see that IND-CPA security follows naturally from the LWE assumption. The proof of the following lemma is given in the full version for completeness.

Lemma 1. *Assume* mPKE *as shown in Fig. 4. Then, for any (classical/quantum)* IND-CPA *adversary* $\mathcal{A}$*, there exist (classical/quantum) adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{LWE}}_{n,n,Nm}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{LWE}}_{(n+Nm),n,\bar{m}}(\mathcal{B}_2).$$

Algorithm 1 $\mathsf{mSetup}(1^\kappa)$

Input: Security parameter 1^κ
Output: Public parameter pp

1: $\mathbf{A} \leftarrow R_q^{n \times n}$
2: **return** $\mathsf{pp} := \mathbf{A}$

Algorithm 2 $\mathsf{mGen}(\mathsf{pp})$

Input: Public parameter $\mathsf{pp} = \mathbf{A}$
Output: Public key pk, a secret key sk

1: $\mathbf{S} \leftarrow D_{\mathsf{s}}^{n \times m}$
2: $\mathbf{E} \leftarrow D_{\mathsf{e}}^{n \times m}$
3: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$ ▷ $\mathbf{B} \in R_q^{n \times m}$
4: **return** $\mathsf{pk} := \mathbf{B}, \mathsf{sk} := \mathbf{S}$

Algorithm 3 $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i \in [N]}, \mathsf{M})$

Input: Public parameter $\mathsf{pp} = \mathbf{A}$, set of public keys $(\mathsf{pk}_i = \mathbf{B}_i)_{i \in [N]}$, message M
Output: Ciphertext $\mathbf{ct} = (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_i)_{i \in [N]})$

1: $\mathsf{r}_0 := (\mathbf{R}, \mathbf{E}') \leftarrow D_{\mathsf{s}}^{\bar{m} \times n} \times D_{\mathsf{e}}^{\bar{m} \times n}$
2: $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0)$
3: **for** $i \in [N]$ **do**
4: $\quad \mathsf{r}_i := \mathbf{E}_i'' \leftarrow D_{\mathsf{e}}^{\bar{m} \times m}$
5: $\quad \widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i)$
6: **return** $\mathbf{ct} := (\mathsf{ct}_0, \widehat{\mathsf{ct}}_1, \ldots, \widehat{\mathsf{ct}}_N)$

Algorithm 4 $\mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i)$

Input: Public parameter $\mathsf{pp} = \mathbf{A}$, public key $\mathsf{pk}_i = \mathbf{B}_i$, message M, randomness $\mathsf{r}_0 = (\mathbf{R}, \mathbf{E}')$ and $\mathsf{r}_i = \mathbf{E}_i''$
Output: (Public key dependent) ciphertext $\widehat{\mathsf{ct}}_i$

1: $\mathbf{V}_i \leftarrow \mathbf{RB}_i + \mathbf{E}_i'' + \mathsf{Encode}(\mathsf{M})$ ▷ $\mathbf{V}_i \in R_q^{\bar{m} \times m}$
2: **return** $\widehat{\mathsf{ct}}_i := \mathbf{V}_i$

Algorithm 5 $\mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0)$

Input: Public parameter $\mathsf{pp} = \mathbf{A}$, randomness $\mathsf{r}_0 = (\mathbf{R}, \mathbf{E}')$
Output: (Public key independent) ciphertext ct_0

1: $\mathbf{U} \leftarrow \mathbf{RA} + \mathbf{E}'$ ▷ $\mathbf{U} \in R_q^{\bar{m} \times n}$
2: **return** $\mathsf{ct}_0 := \mathbf{U}$

Algorithm 6 $\mathsf{mDec}(\mathsf{sk}, \mathsf{ct})$

Input: Secret key $\mathsf{sk} = \mathbf{S}$, ciphertext $\mathsf{ct} = (\mathbf{U}, \mathbf{V})$
Output: Message M

1: $\mathbf{M} \leftarrow \mathbf{V} - \mathbf{US}$ ▷ $\mathbf{M} \in R_q^{\bar{m} \times m}$
2: **return** $\mathsf{M} := \mathsf{Decode}(\mathbf{M})$

Fig. 4: Lattice-based mPKE via the Lindner-Peikert framework [34]. mExt with input index i is defined by picking the relevant components $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ from $\mathbf{ct}$.

Moreover, as a simple consequence of the proof of the above lemma, we have $\mathsf{IND\text{-}Anon\text{-}CPA}$ for free. In particular, the fake encryption algorithm $\overline{\mathsf{mEnc}}$ simply outputs a random element in $R_q^{\bar{m} \times n} \times (R_q^{\bar{m} \times m})^N$.

Remark 5 (Using LWR *instead of* LWE*).* The mPKE presented in Fig. 4 readily generalizes to the LWR setting. The only difference is that instead of adding the noise terms (i.e., $\mathbf{E}, \mathbf{E}', \mathbf{E}_i''$), we round. For instance, the public key pk will be $\lfloor \mathbf{AS} \rceil_p \in R_p^{n \times m}$ rather than $\mathbf{AS} + \mathbf{E} \in R_q^{n \times m}$. It is easy to show that mPKE has γ-spreadness, is δ-correct and $\mathsf{IND\text{-}CPA}$ secure assuming the LWR assumption.

5.2 Multi-Recipient KEMs from Isogenies

Retracing the steps that lead to the hashed version of ElGamal encryption from the Diffie-Hellman key exchange, public-key encryption schemes can be deduced from both SIDH [21] and CSIDH. Building on such encryption schemes, we present two isogeny-based IND-CPA secure mPKEs. Both of them satisfy the generic properties required in Thms. 1 and 2 for obtaining an IND-CCA secure mKEM. Since a unified presentation of the two schemes would be rather convoluted, for the sake of readability we differentiate their explanations. We note that both schemes require a family of universal hash functions $\mathcal{H} = \{\mathsf{H}_k : \mathcal{X} \subset \mathbb{F} \to \{0,1\}^w\}_{k \in K}$ indexed by a finite set K, where $\mathbb{F}$ denotes a finite field. The scheme based on SIDH is detailed below, while, due to space limitation, the CSIDH-based mPKE is provided in the full version.

Isogeny-based mPKE via SIDH. The mPKE deduced from SIDH is provided in Fig. 5. We highlight that the public parameter pp output by mSetup on input a security parameter 1^κ consists of: a prime p of the form $2^{e_2}3^{e_3}-1$; a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ and such that $|E(\mathbb{F}_{p^2})| = (2^{e_2}3^{e_3})^2$; bases $B_2 = \{P_2, Q_2\}$ and $B_3 = \{P_3, Q_3\}$ for $E[2^{e_2}]$ and $E[3^{e_3}]$, respectively; a hash function H uniformly sampled from a family of universal hash functions $\mathcal{H} = \{\mathsf{H}_k : \mathcal{X} \subset \mathbb{F}_{p^2} \to \{0,1\}^w\}_{k \in K}$. Here $\mathcal{X}$ is the set of all supersingular j-invariants in $\mathbb{F}_{p^2}$, for which holds $|\mathcal{X}| = p/12+\epsilon$, with $\epsilon \in \{0,1,2\}$ [21]. Furthermore, Encode (resp. Decode) is an efficiently computable bijective function from the message space (resp. $\{0,1\}^w$) to $\{0,1\}^w$ (resp. message space). The details of Encode and Decode are not significant for this section, since they are scheme specific.

The perfect correctness of the SIDH-based public-key encryption scheme from which our mPKE is deduced implies that the latter has δ-correctness, with $\delta = 0$. In addition, for a given security parameter 1^κ, the prime $p = 2^{e_2}3^{e_3} - 1$ in the public parameter $\mathsf{pp} \leftarrow \mathsf{mGen}(1^\kappa)$ is fixed [28]. The first component of each element in $\mathcal{C}_{\mathsf{single}}$ contains a curve 2^{e_2}-isogenous to E. We denote by W the set $\{j(E/\langle P_2 + [r]Q_2\rangle) | r \in \mathbb{Z}_{2^{e_2}}\}$ of all such curves. Since $p/12 + \epsilon \gg |W|$, one expects that the number of pairs of distinct coefficients $r, \tilde{r} \in \mathbb{Z}_{2^{e_2}}$ such that $j(E/\langle P_2 + [r]Q_2\rangle) = j(E/\langle P_2 + [\tilde{r}]Q_2\rangle)$ is very small [1]. Hence, we can assume that $|W| = 2^{e_2}$ and deduce $\gamma(\mathsf{pp}, \mathsf{pk}) \geq e_2$. This value is independent of the public key pk and E, B_2, B_3 in pp,therefore the mPKE scheme has γ-spreadness with $\gamma = e_2$. We observe that $1/2^{e_2} \approx 1/\sqrt{p}$, which is negligible in the security parameter κ ($e_2 \geq \kappa$ for any set of SIDH parameters [28]).

IND-(Anon-)CPA *Security.* The IND-CPA security of the SIDH-based mPKE follows from the SSDDH assumption and the Leftover Hash Lemma. The proof of the following lemma is given in the full version for completeness.

Lemma 2. *Assume* mPKE *as shown in Fig. 5. Then, for any (classical/quantum)* IND-CPA *adversary* $\mathcal{A}$*, there exists a (classical/quantum) adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{A}) \leq N \cdot \left(\mathsf{Adv}^{\mathsf{SSDDH}}_{p,E,B_2,B_3}(\mathcal{B}) + \frac{1}{2}\sqrt{2^w/p}\right). \tag{2}$$

Algorithm 7 $\mathsf{mSetup}(1^\kappa)$

Input: Security parameter 1^κ
Output: Public parameter pp

1: Select $e_2, e_3, E, B_2 = \{P_2, Q_2\}, B_3 = \{P_3, Q_3\}$
2: $\mathsf{H} \leftarrow \mathcal{H}$
3: **return** $\mathsf{pp} := (E, \{(e_j, B_j)\}_{j=2,3}, \mathsf{H})$

Algorithm 8 $\mathsf{mGen}(\mathsf{pp})$

Input: Public parameter $\mathsf{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, \mathsf{H})$
Output: Public key pk, a secret key sk

1: $(P_2, Q_2) \leftarrow B_2, (P_3, Q_3) \leftarrow B_3$
2: $s \leftarrow \mathbb{Z}_{3^{e_3}}$
3: $R_3 \leftarrow P_3 + [s]Q_3$
4: $E_3 \leftarrow E/\langle R_3 \rangle$
5: $U_2 \leftarrow \phi_{\langle R_3 \rangle}(P_2), V_2 \leftarrow \phi_{\langle R_3 \rangle}(Q_2)$
6: **return** $\mathsf{pk} := (E_3, U_2, V_2), \mathsf{sk} := s$

Algorithm 9 $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M})$

Input: Public parameter $\mathsf{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, \mathsf{H})$, set of public keys $(\mathsf{pk}_i = (E_3^{(i)}, U_2^{(i)}, V_2^{(i)}))_{i\in[N]}$, message M
Output: Ciphertext $\mathbf{ct} = (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_i)_{i\in[N]})$

1: $\mathsf{r}_0 := r \leftarrow \mathbb{Z}_{2^{e_2}}$
2: $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0)$
3: **for** $i \in [N]$ **do**
4: $\quad \widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0)$
5: **return** $\mathbf{ct} := (\mathsf{ct}_0, \widehat{\mathsf{ct}}_1, \ldots, \widehat{\mathsf{ct}}_N)$

Algorithm 10 $\mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0)$

Input: Public parameter $\mathsf{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, \mathsf{H})$, public key $\mathsf{pk}_i = (E_3^{(i)}, U_2^{(i)}, V_2^{(i)})$, message M, randomness $\mathsf{r}_0 = r$
Output: (Public key dependent) ciphertext $\widehat{\mathsf{ct}}_i$

1: $T_i \leftarrow U_2^{(i)} + [r]V_2^{(i)}$
2: $J_i \leftarrow \mathsf{jInvariant}(E_3^{(i)}/\langle T_i \rangle)$
3: $F_i \leftarrow \mathsf{H}(J_i) \oplus \mathsf{Encode}(\mathsf{M})$
4: **return** $\widehat{\mathsf{ct}}_i := F_i$

Algorithm 11 $\mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0)$

Input: Public parameter $\mathsf{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, \mathsf{H})$, randomness $\mathsf{r}_0 = r$
Output: (Public key independent) ciphertext ct_0

1: $(P_2, Q_2) \leftarrow B_2, (P_3, Q_3) \leftarrow B_3$
2: $R_2 \leftarrow P_2 + [r]Q_2$
3: $E_2 \leftarrow E/\langle R_2 \rangle$
4: $U_3 \leftarrow \phi_{\langle R_2 \rangle}(P_3), V_3 \leftarrow \phi_{\langle R_2 \rangle}(Q_3)$
5: **return** $\mathsf{ct}_0 := (E_2, U_3, V_3)$

Algorithm 12 $\mathsf{mDec}(\mathsf{sk}, \mathsf{ct})$

Input: Public parameter $\mathsf{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, \mathsf{H})$, secret key $\mathsf{sk} = s$, ciphertext $\mathsf{ct} = (E_2, U_3, V_3, F)$
Output: Message M

1: $R' \leftarrow U_3 + [s]V_3$
2: $E' \leftarrow E_2/\langle R' \rangle$
3: $J' \leftarrow \mathsf{jInvariant}(E')$
4: $\mathbf{M} \leftarrow F \oplus \mathsf{H}(J')$
5: **return** $\mathsf{M} := \mathsf{Decode}(\mathbf{M})$

Fig. 5: SIDH-based mPKE via hashed ElGamal [21]. mExt with input index i is defined by picking the relevant components $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ from $\mathbf{ct}$. Note that $\mathsf{mEnc}^{\mathsf{d}}$ does not require any randomness r_i for $i \in [N]$.

We note that in concrete instantiations, $\log_2 p$ assumes one of the values $434, 503, 610$, while the corresponding w is 128,192 or 256, respectively [28]. Therefore the quantity $(1/2)\sqrt{2^w/p}$ is bounded by 2^{152} for each pair (p, w) and it can be safely discarded in the right term of Eq. (2). Moreover, as a simple consequence of the concrete proof of the above lemma, we have IND-Anon-CPA for free. In particular, the fake encryption algorithm $\overline{\mathsf{mEnc}}$ simply outputs a tuple composed by a ciphertext ct_0 and N uniformly random elements in $\{0,1\}^w$.

Isogeny-based mPKE via CSIDH. Since the high level structure of our CSIDH-based mPKE is similar to our SIDH-based mPKE, we refer the full details to full version. We consider the action of a cyclic group G on a set of supersingular elliptic curves. However, it can be easily adapted to the case where the structure of G is unknown.

6 Instantiating mKEM with NIST Candidates and CSIDH

In this section, we concretely instantiate the generic mKEM framework laid out in previous sections. We take the PKEs underlying 8 existing lattice-based and isogeny-based NIST KEMs (as well as CSIDH). We first modify them into efficient mPKEs (following Sec. 5) and then into mKEMs via our generic transformation (Thms. 1 and 2). We note that we did not consider the corresponding mKEM for the CISDH mPKE, for reasons explained later. We compare these mKEMs to the trivial solution that uses (single-recipient) KEMs in parallel, and show that our mKEMs provide efficiency gains, both in communication and computation, of an order of magnitude.

Until the end of this document, we denote by $|\mathsf{x}|$ the bytesize of an object x, where x may be any cryptographic object (a public key, a ciphertext, etc.)

6.1 Comparison Methodology

Our goal is to provide an accurate assessment of the gains provided by various mKEM instantiations. A natural way to do that is to compare the performances of these mKEMs (with N recipients) with N instantiations of the original (single-recipient) KEMs. This comparison can be done via two metrics:

(C1) Communication cost. How much data does the encryptor broadcast when using mKEM with N recipients, and how does it compare to N instances of the original KEM (one per recipient)?

(C2) Computational cost. How many cycles does one instance of mKEM with N recipients cost, and how does it compare to N instances of KEM?

For (C1), we measure the ratio:

$$\frac{\text{Data broadcast when using } N \text{ instances of the original KEM}}{\text{Data broadcast when using mKEM with } N \text{ recipients}}. \tag{3}$$

With mKEM the encryptor broadcasts a single multi-ciphertext of size $|\mathsf{ct}_0| + \sum_{i\in[N]} |\widehat{\mathsf{ct}}_i| = |\mathsf{ct}_0| + N|\widehat{\mathsf{ct}}_i|$, whereas with N instances of KEM he broadcasts

N ciphertexts $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ – except for NewHope, see footnote 4 – for a total size $N|\mathsf{ct}_0| + N|\widehat{\mathsf{ct}}_i|$. Therefore, the ratio converges to a value independent of N when N tends to infinity. Specifically, the value (3) is:

$$\frac{N|\mathsf{ct}_0| + N|\widehat{\mathsf{ct}}_i|}{|\mathsf{ct}_0| + N|\widehat{\mathsf{ct}}_i|} \underset{N\to\infty}{\longrightarrow} 1 + \frac{|\mathsf{ct}_0|}{|\widehat{\mathsf{ct}}_i|}. \tag{4}$$

Let $k_{\mathsf{comm}} = 1 + \frac{|\mathsf{ct}_0|}{|\widehat{\mathsf{ct}}_i|}$. This value measures asymptotically "how much more compact" mKEM is compared to the original KEM, and serves as our metric for (C1). Similarly, the following value serves as our metric for (C2):

$$k_{\mathsf{cycles}} = \lim_{N\to\infty} \frac{\text{Cycles spent to run } N \text{ instances of the original KEM}}{\text{Cycles spent to run mKEM with } N \text{ recipients}} \tag{5}$$

We note that k_{cycles} is far less absolute than k_{comm} as a metric, since the number of cycles depend on the implementation of a scheme, the architecture of the target platform, etc. However, it is a useful indicator of the efficiency gain that one can expect by using mKEM. All cycles measurements in this section are performed on a processor i7-8665U (Whiskey Lake) @ 1.90GHz, with Turbo Boost disabled.

6.2 Instantiation with Lattice-based NIST Candidates

In this section, we provide concrete instantiations of the high-level scheme described in Sec. 5.1. Our efforts are facilitated by the fact that 7 lattice-based NIST candidate KEMs are deduced from PKEs that follow the Lindner-Peikert framework:

- Kyber;
- FrodoKEM;
- LAC;
- NewHope;
- Round5;
- Saber;
- ThreeBears.

Full specifications of these 7 schemes are available at [36]. Out of these, FrodoKEM, Kyber, LAC and NewHope follow the most closely the Lindner-Peikert framework, since they are based on LWE, Module-LWE, Ring-LWE and Ring-LWE, respectively. Round5 and Saber are based on variants of LWR. This implies a few changes on Fig. 4, since the addition of noise error is replaced in some instances by rounding. See Rem. 5 for a short discussion on this change. Finally, ThreeBears is based on an extremely recent variant called Module Integer-LWE. In addition, each scheme has different parameters and uses different tweaks. A widespread trick is for $\widehat{\mathsf{ct}}_i$ to drop the least significant bits of $\mathbf{V}_i$, since the message M is encoded in the most significant bits. This reduces the size of a (multi-)ciphertext. Note that bit dropping is more beneficial to mKEMs than to KEMs as it reduces $|\widehat{\mathsf{ct}}_i|$, hence a larger bandwidth impact for mKEMs – see (4).

These 7 KEMs and the PKEs they are based on serve as the bases for our mKEM constructions. We tweaked them in order to fit the frameworks described in Fig. 4 (IND-CPA mPKE) and Fig. 2 (conversion into an IND-CCA mKEM). Note that our tweaks break compatibility with the specifications of the aforementioned schemes, for two reasons. First, we fix the public matrix $\mathbf{A}$ in order to fit Fig. 4 (see Rem. 6 below). Second, the transform of Fig. 2 is completely

different from the ones used in the 7 aforementioned KEMs, which themselves differ from each other. As a consequence, comparing our mKEMs to these KEMs is not an entirely apples-to-apples comparison, as the 7 KEMs we cited claim some additional properties such as contributivity or security in specific threat models (see Rem. 6). For our mKEMs, we do not claim to achieve any security notion besides those proven in this document.

Remark 6 (Reusing the public matrix). A difference between Fig. 4 and the aforementioned NIST schemes is that the latter use PKEs for which the matrix $\mathbf{A}$ is made part of the public key pk. That is, each user has its $\mathbf{A}$ rather than sharing it. The main argument for this choice is to hinder *all-for-the-price-of-one attacks* [2, Section 3]. The associated threat model considers an attacker that has enough cryptanalytic capabilities to break *one* hard instance of a lattice problem, but not much more. This is an arguably specific security model, one that implicitly considers that the parameter set of the scheme may not be cryptographically secure. In order to enable our mKEM instantiations, we instead make $\mathbf{A}$ part of the public parameter pp, as per Fig. 4. This can be done with minimal changes to the PKEs used by the original KEMs, and has no impact on their concrete security analysis.

Communication costs. Tab. 1 provides a comparison of NIST KEMs with their mKEM variants. Sending N ciphertexts costs $N \cdot |\mathsf{ct}|$ bytes for a NIST KEM, whereas using its mKEM counterpart costs $|\mathsf{ct}_0| + N \cdot |\widehat{\mathsf{ct}}_i|$. The gain in bandwidth k_{comm} is of one order of magnitude (sometimes two). Schemes based on module lattices (Saber, Kyber, ThreeBears) and standard lattices (FrodoKEM) see the most dramatic gains (as high as a factor 169 times for FrodoKEM).

Computational costs. Due to time constraints, we only implemented mKEM on two lattice-based schemes: FrodoKEM and Kyber. Nevertheless, we believe these examples already showcase the efficiency gain provided by our techniques. Starting from reference implementations available on Github[8][9], we tweaked them to obtain mKEMs. As shown by Tab. 2, our mKEM variants perform (multi-)encapsulation between one and two orders of magnitude faster than their original KEM counterparts. We provide additional experiments in the full version and show that the target platform can play an important role in the performance gain.

6.3 Instantiation with Isogeny-Based schemes

In this section, we focus on isogeny-based instantiations of mKEM and mPKE. Concerning SIKE, we obtain an mKEM from the mPKE of Fig. 5, and we compare it with the trivial solution consisting in N instances of SIKE. For CSIDH,

[4] Unlike other lattice-based KEMs, the CCA variant of NewHope adds a hash to the ciphertext. So in this particular case $|\mathsf{ct}| = |\mathsf{ct}_0| + |\widehat{\mathsf{ct}}_i| + \{32, 64\}$.

[8] https://github.com/Microsoft/PQCrypto-LWEKE

[9] https://github.com/pq-crystals/kyber/

Table 1: Bandwidth impact of our solution on various schemes. Sizes are in bytes.

Scheme	$\lvert\mathsf{ct}_0\rvert$	$\lvert\widehat{\mathsf{ct}}_i\rvert$	$\lvert\mathsf{ct}\rvert$	k_{comm}
FrodoKEM-640	9 600	120	9 720	**81**
FrodoKEM-976	15 616	128	15 744	**123**
FrodoKEM-1344	21 504	128	21 632	**169**
Kyber-512	640	96	736	**7.67**
Kyber-768	960	128	1 088	**8.5**
Kyber-1024	1 408	160	1 568	**9.8**
LAC-128	512	200	712	**3.56**
LAC-192	1024	164	1188	**7.24**
LAC-256	1024	400	1424	**3.56**
NewHope-512-CCA-KEM[7]	896	192	1 120	**5.83**
NewHope-1048-CCA-KEM	1 792	384	2 208	**5.75**
Round5 `R5ND_1KEMb`	429	110	539	**4.9**
Round5 `R5ND_3KEMb`	756	74	830	**11.22**
Round5 `R5ND_5KEMb`	940	142	1 082	**7.62**
LightSaber	640	96	736	**7.67**
Saber	960	128	1 088	**8.5**
FireSaber	1 280	192	1 472	**7.67**
BabyBear	780	137	917	**6.69**
MamaBear	1 170	137	1 307	**9.54**
PapaBear	1 560	137	1 697	**12.38**

Table 2: Encapsulation times of FrodoKEM and Kyber vs their mKEM variants. Times are in cycles and are normalized by the number of recipients (here, 1000).

Scheme	Trivial KEM	Our mKEM	k_{cycles}
FrodoKEM-640	4 948 835	251 405	**19.68**
FrodoKEM-976	10 413 149	387 733	**26.86**
FrodoKEM-1344	18 583 122	519 973	**35.74**
Kyber-512	181 297	42 647	**4.25**
Kyber-768	279 210	52 471	**5.32**
Kyber-1024	414 774	61 808	**6.71**

we compare our mPKE in the full version with N instances of the CSIDH-based hashed ElGamal. Since CSIDH is a key-exchange, we simply construct a trivial IND-CPA secure PKE from it (rather than constructing an IND-CCA secure KEM) and compare it with our mPKE from Sec. 5.2. To obtain proof-of-concept implementation of mPKE for CSIDH and mKEM for SIKE, we have modified implementation available in the NOBS library[10].

[10] https://github.com/henrydcase/nobs

Communication cost. Our construction provides the most significant gain when used with SIKE/p434. In this case our mKEM variant can be over 20 times more efficient.

Table 3: Bandwidth impact of our mKEM on isogeny schemes. Sizes are in bytes.

Scheme	$\lvert\mathsf{ct}_0\rvert$	$\lvert\widehat{\mathsf{ct}}_i\rvert$	$\lvert\mathsf{ct}\rvert$	k_{comm}
SIKE/p434	330	16	346	**21.63**
SIKE/p503	378	24	402	**16.75**
SIKE/p751	564	32	596	**18.63**
SIKE/p434_compressed	196	16	209	**13.25**
SIKE/p503_compressed	224	24	248	**10.33**
SIKE/p751_compressed	331	32	363	**11.34**
cSIDH PKE/p512	64	16	80	**5**

Computational costs. In SIKE and CSIDH-based hashed ElGamal, the computational cost is dominated by isogeny computations. In both schemes, encapsulation/encryption requires the computation of two smooth-degree isogenies. Assuming SIKE key compression is not used, we can assume that both computations have a similar cost C. When running SIKE/CSIDH-based hashed ElGamal for N recipients, the total computation cost is roughly $2 \cdot N \cdot C$. By applying our mKEM/mPKE this cost reduces to $(N + 1) \cdot C$. So, the expectation is that our approach will be roughly two times faster. The results from the benchmarking in Tab. 4 confirms the expected speed-up. It is worth noticing that the gain from using mKEM is expected to be bigger when using SIKE with key compression. That is because computing $\lvert\mathsf{ct}_0\rvert$ is a slower operation than computing $\lvert\widehat{\mathsf{ct}}_i\rvert$.

Table 4: Encapsulation times of SIKE vs its mKEM variant and encryption times of CSIDH-based hashed ElGamal vs its mPKE variant. Times are in cycles and are normalized by the number of recipients (here, 100).

Scheme	Trivial KEM	Our mKEM	k_{cycles}
SIKE/p434	1 657 655 212	759 202 275	**2.18**
SIKE/p503	2 301 014 376	1 037 469 650	**2.22**
SIKE/p751	6 900 791 605	3 150 069 659	**2.19**
cSIDH/p512	37 455 411 429	19 438 021 692	**1.92**

7 Application to Secure Group Messaging

In this section, we show how our mKEM can be used to optimize the *TreeKEM* protocol [3, 5, 14] used within secure group messagings. The resulting protocol has a lower communication cost than the standard version of TreeKEM [5, 14].

7.1 Syntax and Notations for Group Messaging

We first introduce group messaging-related notions. We observe that group messaging is an extensive topic; we keep our presentation minimal and introduce notions that are strictly required for our argument. More in-depth discussions on group messaging can be found in e.g. [3, 5, 10, 14].

Continuous group key agreement (CGKA), which generalizes the notion of continuous key agreement (CKA, see [4]), forms the backbone of secure *group* messaging (SGM) protocols. Informally, one can think of CGKA as a group key exchange where the group members dynamically change and the (group) session keys need to be re-established in each epoch to maintain strong security. Once a session key is established for a given epoch, a user can then use the key to securely communicate with the group members. Therefore, a SGM protocol can be described as a continuum of running CGKA and exchanging secured messages.

Definition 11 (Continuous Group Key Agreement. [5]). *A continuous group key agreement* $\mathsf{CGKA} = (\mathsf{Init}, \mathsf{Create}, \mathsf{Add}, \mathsf{Remove}, \mathsf{Update}, \mathsf{Process})$ *consists of the following algorithms:*

- ***Initialization.*** Init *takes an ID* ID *and outputs an initial state* state.
- ***Group creation.*** Create *takes a state* state, *a list of IDs* $(\mathsf{ID}_i)_{i\in[N]}$ *and outputs a new state* state' *and a control message* W.
- ***Add.*** Add *takes a state* state, *an ID* ID *and outputs a new state* state' *and control messages* W, T.
- ***Remove.*** Remove *takes a state* state, *an ID* ID *and outputs a new state* state' *and a control message* T.
- ***Update.*** Update *takes a state* state *and outputs a new state* state' *and a control message* T.
- ***Process.*** $\mathsf{Process}$ *takes a state* state *and outputs a new state* state' *and an update secret* I.

Above, Update allows a user to update the session key on behalf of the whole group (it is run on every epoch to maintain strong security), and $\mathsf{Process}$ allows each group member to process the updated session key. Four properties are required from a CGKA: correctness, privacy, forward privacy (FS), and post-compromise security (PCS). At a high level, FS states that if any group member is compromised at some point, then all previous session keys remain hidden from the attacker; and PCS states that after every compromised group member performs an update, the session key becomes secret again. As the precise definitions are not relevant to our work, we refer to [5, Section 3.2] for more details.

In the following, we focus on TreeKEM; a specific instantiation of CGKA that forms the building block of the SGM protocol MLS [10]. It was first described in [14] and various improvements have been proposed in [3,5]. TreeKEM is at the heart of the MLS protocol [10], and is arguably one of MLS' main efficiency bottlenecks due to the large number of public key material sent. To be more concrete, our efforts are directed at optimizing the Update algorithm of TreeKEM; this algorithm constitutes an efficiency bottleneck (in computation and communication) of TreeKEM as it is performed on a regular basis (in contrast to Create, Add and Remove, which are performed upon punctual events). In effect, improving the efficiency of Update will improve the efficiency of TreeKEM (and hence the MLS protocol) on a similar scale. Details on TreeKEM follows.

Dendrologic notations. In a (binary or m-ary) tree T, a *leaf* is a node with no child, an *internal node* is a node that is not a leaf, and the root root is the unique node that has no parent. By synecdoche, we may abusively refer to a node by its label; for example in Fig. 6, "1" denotes the bottom left node.

Let u be a node in a tree T. Its siblings, $\mathsf{siblings}(u)$, is the set of nodes $v \neq u$ in T with the same parent as u. Its *path*, $\mathsf{path}(u)$, is the set of nodes between u and root, including u but excluding root. Its *co-path*, $\mathsf{copath}(u)$, is the set of siblings of nodes in its path: $\mathsf{copath}(u) = \bigcup_{v \in \mathsf{path}(u)} \mathsf{siblings}(v)$. For example, in Fig. 6, the only sibling of "1" is "2", its path is the set of red nodes ( ), and its co-path is the set of green nodes ( ).

TreeKEM. In TreeKEM, a (binary or m-ary) tree T is constructed with the N group members as its leaves. As an example, Fig. 6 illustrates the tree T associated to a group of 16 users (numbered from 1 to 16). Let PRG be a pseudorandom generator. Then, to each node i is associated a secret seed seed_i and a keypair $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{mGen}(\mathsf{pp}; \mathsf{PRG}(\mathsf{seed}_i)_{\mathsf{L}})$, where $\mathsf{PRG}(\cdot)_{\mathsf{L}}$ (resp. $\mathsf{PRG}(\cdot)_{\mathsf{R}}$) denotes the left (resp. right) half output of the PRG. In particular, mGen is run on randomness $\mathsf{PRG}(\mathsf{seed}_i)_{\mathsf{L}}$. The root does not need a keypair, but its seed will in effect be the group secret I (i.e., session key). The *TreeKEM invariant* states that a group member u knows seed_i if and only if $i \in \mathsf{path}(u)$. When a user u performs an update (via Update), he does the following:

(U1) Generate a new secret seed seed_u for u.
(U2) For each $i \in \mathsf{path}(u)$, update its keypair: $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{mGen}(\mathsf{pp}; \mathsf{PRG}(\mathsf{seed}_i)_{\mathsf{L}})$, and compute a new secret seed for its parent: $\mathsf{seed}_{\mathsf{parent}(i)} = \mathsf{PRG}(\mathsf{seed}_i)_{\mathsf{R}}$.
(U3) For each $i \in \mathsf{path}(u)$, compute the ciphertext

$$\mathbf{ct}_i \leftarrow \mathsf{mEncaps}(\mathsf{pp}, (\mathsf{pk}_j)_{j \in \mathsf{siblings}(i)}; \mathsf{seed}_{\mathsf{parent}(i)}). \tag{6}$$

Note that $\mathsf{mEncaps}$ is derandomized here. For our construction in Fig. 2, this is equivalent to setting the random message $\mathsf{M}_i = \mathsf{PRG}(\mathsf{seed}_{\mathsf{parent}(i)})$.
(U4) Send the update package $(\mathsf{pk}_i, \mathbf{ct}_i)_{i \in \mathsf{path}(u)}$ to the server, which dispatches it to the other group members (this is known as *server-side fan-out*).

Upon receiving the update package, a user v processes it (via $\mathsf{Process}$) as follows:

(P1) Update each pk_i he received.
(P2) Compute the closest common ancestor w of u and v, then recover seed_w by decapsulating the adequate $\mathbf{ct}_i$.
(P3) Recover the secret seeds of all remaining common ancestors of u and v by computing $\mathsf{seed}_{\mathsf{parent}(i)} = \mathsf{PRG}(\mathsf{seed}_i)_{\mathsf{R}}$. The update secret is $I = \mathsf{seed}_{\mathsf{root}}$

This description is more generic than previous ones [3, 5, 10, 14] in the following sense. All existing instantiations of TreeKEM take T to be a binary tree, in which case there is no need for a mKEM as a single-recipient KEM suffices. Note that while our description uses mKEM as a building block, it is easily adapted to work with an mPKE. Fig. 6 illustrates the "classical" instantiation of TreeKEM. Each update contains at most $\lceil \log_2(N) \rceil$ public keys and as many ciphertexts, so its bytesize is at most:

$$\lceil \log_2(N) \rceil \cdot \left(|\mathsf{pk}| + |\mathsf{ct}_0| + |\widehat{\mathsf{ct}}_i| \right) \tag{7}$$

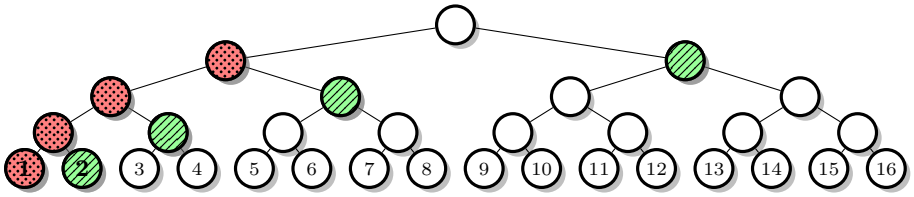


Fig. 6: TreeKEM

***m*-ary TreeKEM.** We now show how to obtain significant efficiency gains by instantiating TreeKEM with an m-ary tree combined with mKEM. As mentioned in [14], TreeKEM can be instantiated with an m-ary tree instead of binary; see Fig. 7 for an example where "1" issues a package update. At first, it is not obvious that this is more efficient than the instantiation of Fig. 6, since in our example the update package now contains 2 public keys (one for each node ( )) in the path) and 6 ciphertexts (one for each node ( )) in the co-path).

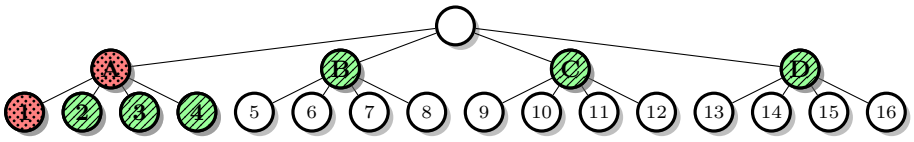


Fig. 7: 4-ary TreeKEM

We make the following observation: when a user u issues an update, the update package may encapsulate several times the same information. Precisely,

for each $i \in \mathsf{path}(u)$, the update package encapsulates $\mathsf{seed}_{\mathsf{parent}(i)}$ under the key pk_j for each $j \in \mathsf{siblings}(i)$. In the example of Fig. 7, this means that an update package issued by 1 encapsulates seed_A under $\mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4$, and $\mathsf{seed}_{\mathsf{root}}$ under $\mathsf{pk}_B, \mathsf{pk}_C, \mathsf{pk}_D$. The bandwidth gain happens exactly here: since the same value seed_A is encapsulated under $\mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4$, one can use mKEM to perform this (multi-)encapsulation. And similarly at each level of the tree. Hence the total size of an update package is at most: $\lceil \log_m(N) \rceil \cdot \left(|\mathsf{pk}| + |\mathsf{ct}_0| + (m-1) \cdot |\widehat{\mathsf{ct}}_i| \right)$. One can see that this generalizes (7) to any integer $m > 2$. It is clear that whenever $|\mathsf{pk}| + |\mathsf{ct}_0| \gg |\widehat{\mathsf{ct}}_i|$, it is advantageous efficiency-wise to take $m > 2$. This is illustrated in the next section.

7.2 Concrete Instantiations of m-ary TreeKEM

We now illustrate the substantial communication gains that can be obtained in practice with the method described above. A good rule of thumb is to take $m - 1 \approx \frac{|\mathsf{pk}| + |\mathsf{ct}_0|}{|\widehat{\mathsf{ct}}_i|}$. According to (7), the bytesize of an update package for binary TreeKEM will then be approximately $\lceil \log_2(N) \rceil \cdot m \cdot |\widehat{\mathsf{ct}}_i|$. On the other hand, the bytesize – given by (**??**) – for our proposal is about $\lceil \log_m(N) \rceil \cdot 2(m-1) \cdot |\widehat{\mathsf{ct}}_i|$. Compared to the standard TreeKEM, our proposal improves communication cost by a factor equal to the ratio of the two values, which is approximately:

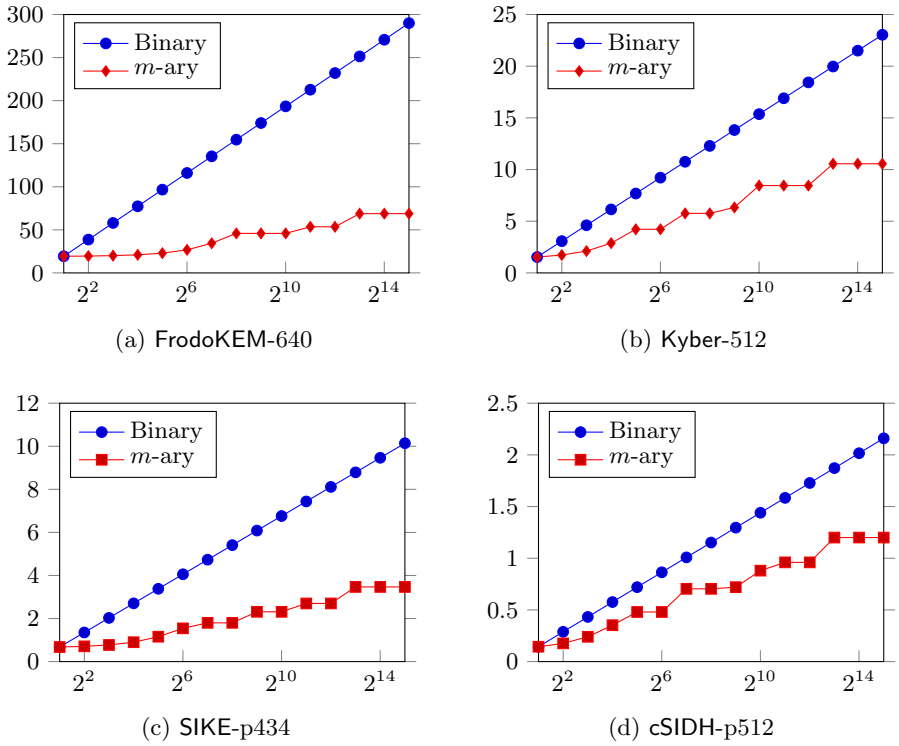
$$\begin{aligned} \frac{\lceil \log_2(N) \rceil \cdot m \cdot |\widehat{\mathsf{ct}}_i|}{\lceil \log_m(N) \rceil \cdot 2(m-1) \cdot |\widehat{\mathsf{ct}}_i|} &\underset{N \to \infty}{\longrightarrow} \frac{m}{2(m-1)} \cdot \log_2(m) \\ &= O(\log m). \end{aligned}$$

Our solution provides a gain $O(\log m)$ compared to TreeKEM. A concrete comparison is provided by Fig. 8, which compares the bytesize of an update package for binary TreeKEM - using FrodoKEM, Kyber, SIKE or cSIDH as a (single-recipient) KEM/PKE - and m-ary TreeKEM - using the mKEM/mPKE obtained from FrodoKEM, Kyber, SIKE or cSIDH, respectively. For the schemes considered, our proposal improves the communication cost for large groups by a factor between 1.8 and 4.2.

Acknowledgement. Shuichi Katsumata was supported by JST CREST Grant Number JPMJCR19F6 and JSPS KAKENHI Grant Number JP19H01109. Kris Kwiatkowski and Thomas Prest were supported by the Innovate UK Research Grant 104423 (PQ Cybersecurity). The authors would like to thank Takashi Yamakawa for helpful discussions on QROM.

References

1. G. Adj, D. Cervantes-Vázquez, J.-J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. *SAC 2018*, pp. 322–343.
2. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. *USENIX Security 2016*, pp. 327–343.



(a) FrodoKEM-640 (b) Kyber-512

(c) SIKE-p434 (d) cSIDH-p512

Fig. 8: Comparing the classic "binary" TreeKEM with m-ary TreeKEM, when instantiated with four schemes: FrodoKEM, Kyber, SIKE and cSIDH. In each case, the x-axis represent the number N of group members (from 2 to 2^{15}) and the y-axis represent the maximal size of an update package in kilobytes. The arity m depends on the scheme and the group size N, and is omitted for readability.

3. J. Alwen, M. Capretto, M. Cueto, C. Kamath, K. Klein, G. Pascual-Perez, K. Pietrzak, and M. Walter. Keep the dirt: Tainted treekem, an efficient and provably secure continuous group key agreement protocol. Cryptology ePrint Archive, Report 2019/1489.
4. J. Alwen, S. Coretti, and Y. Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. *EUROCRYPT 2019*, pp. 129–158.
5. J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. Cryptology ePrint Archive, Report 2019/1189.
6. A. Ambainis, M. Hamburg, and D. Unruh. Quantum security proofs using semi-classical oracles. *CRYPTO 2019*, pp. 269–295.
7. A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. *EUROCRYPT 2012*, pp. 719–737.
8. M. Barbosa and P. Farshim. Randomness reuse: Extensions and improvements. In *IMA International Conference on Cryptography and Coding*, pp. 257–276. Springer.
9. E. Barker, L. Chen, A. Roginsky, M. Smid, E. Barker, L. Chen, A. Roginsky, and M. Smid. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. In *Technical Report; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2006. 2012*, page 15158. https://doi.org/10.6028/NIST.SP.800-56Ar3.
10. R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-09, Internet Engineering Task Force, 2020. Work in Progress.
11. O. Baudron, D. Pointcheval, and J. Stern. Extended notions of security for multicast public key cryptosystems. *ICALP 2000*, pp. 499–511.
12. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. *EUROCRYPT 2000*, pp. 259–274.
13. M. Bellare, A. Boldyreva, and J. Staddon. Randomness re-use in multi-recipient encryption schemeas. *PKC 2003*, pp. 85–99.
14. K. Bhargavan, R. Barnes, and E. Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, 2018.
15. N. Bindel, M. Hamburg, K. Hövelmanns, A. Hülsing, and E. Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In *TCC 2019*, pp. 61–90.
16. W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: An efficient post-quantum commutative group action. *ASIACRYPT 2018*, pp. 395–427.
17. H. Cheng, X. Li, H. Qian, and D. Yan. Cca secure multi-recipient kem from lpn. In *ICICS*, pp. 513–529. Springer.
18. J.-S. Coron, H. Handschuh, M. Joye, P. Paillier, D. Pointcheval, and C. Tymen. GEM: A generic chosen-ciphertext secure encryption method. *CT-RSA 2002*, pp. 263–276.
19. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226.
20. J. Czajkowski, C. Majenz, C. Schaffner, and S. Zur. Quantum lazy sampling and game-playing proofs for quantum indifferentiability. Cryptology ePrint Archive, Report 2019/428.
21. L. De Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Journal of Mathematical Cryptology*, pp. 209–247.

22. A. W. Dent. A designer's guide to kems. In *IMA International Conference on Cryptography and Coding*, pp. 133–151. Springer.
23. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *CRYPTO'99*, pp. 537–554.
24. H. Hiwatari, K. Tanaka, T. Asano, and K. Sakumoto. Multi-recipient public-key encryption from simulators in security proofs. *ACISP 09*, pp. 293–308.
25. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. *TCC 2017*, pp. 341–371.
26. D. Hofheinz and E. Kiltz. Secure hybrid encryption from weakened key encapsulation. *CRYPTO 2007*, pp. 553–571.
27. K. Hövelmanns, E. Kiltz, S. Schäge, and D. Unruh. Generic authenticated key exchange in the quantum random oracle model. In *PKC 2020*, pp. 389–422.
28. D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, D. Urbanik, and G. Pereira. SIKE. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
29. H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. *CRYPTO 2018*, pp. 96–125.
30. H. Jiang, Z. Zhang, and Z. Ma. Key encapsulation mechanism with explicit rejection in the quantum random oracle model. *PKC 2019*, pp. 618–645.
31. H. Jiang, Z. Zhang, and Z. Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. In *PQCRYPTO 2019*, pp. 227–248.
32. V. Kuchta, A. Sakzad, D. Stehlé, R. Steinfeld, and S.-F. Sun. Measure-rewind-measure: tighter quantum random oracle model proofs for one-way to hiding and cca security. In *EUROCRYPT 2020*, pp. 703–728. Springer.
33. K. Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. *PKC 2002*, pp. 48–63.
34. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. *CT-RSA 2011*, pp. 319–339.
35. T. Matsuda and G. Hanaoka. Key encapsulation mechanisms from extractable hash proof systems, revisited. *PKC 2013*, pp. 332–351.
36. NIST. Post-quantum cryptography - round 2 submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
37. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. *CT-RSA 2001*, pp. 159–175.
38. E. Omara, B. Beurdouche, E. Rescorla, S. Inguva, A. Kwon, and A. Duric. The Messaging Layer Security (MLS) Architecture. Internet-Draft draft-ietf-mls-architecture-04, Internet Engineering Task Force, 2020. Work in Progress.
39. T. Saito, K. Xagawa, and T. Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. *EUROCRYPT 2018*, pp. 520–551.
40. N. P. Smart. Efficient key encapsulation to multiple parties. *SCN 04*, pp. 208–219.
41. E. E. Targhi and D. Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. *TCC 2016-B*, pp. 192–216.
42. Z. Yang. On constructing practical multi-recipient key-encapsulation with short ciphertext and public key. *SCN*, 8(18):4191–4202.
43. M. Zhandry. How to record quantum queries, and applications to quantum indifferentiability. *CRYPTO 2019*, pp. 239–268.