

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

Shuichi Katsumata^{1,2}, Kris Kwiatkowski², Federico Pintore³, Thomas Prest²

¹ National Institute of Advanced Industrial Science and Technology (AIST)

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 [46, 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.¹ 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.

¹ To be more precise, it is common to rely on the KEM/DEM framework [20, 23] 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 [49]², 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, 27, 29, 37, 42, 52]. Also, similar variants of mKEM, such as *multi-message* multi-recipient *public-key encryption* [11–13, 37], 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, 27, 42] 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-

² We note that very similar variants of mKEM have been considered prior to this work [11–13, 37]. 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 **mKEMs**?*

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**).³ 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, 27, 42] which provide some types of generic constructions of **mKEM**, ours require an **mpKE** whereas they only require a single-recipient **PKE**.

³ 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 $\mathsf{IND-CPA}$. 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 PKE s, 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 $\mathsf{IND-CPA}$ secure mPKE , under which our above generic construction naturally implies a recipient-anonymous $\mathsf{IND-CCA}$ 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 KEM s 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, 46], 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 KEM s.

Our Techniques: Generic Construction of $\mathsf{IND-CCA}$ secure mKEM . On a high level, our generic construction can be seen as a generalization of the Fujisaki-Okamoto (FO) transform [25]. The FO transform (roughly) converts any $\mathsf{IND-CPA}$ secure PKE into an $\mathsf{IND-CCA}$ secure KEM . There are several variants of

the FO transform and most of the variants are secure in the ROM [18, 23, 28, 45] and/or QROM [15, 28, 33–36, 48, 50, 53]. The high-level construction is as follows: to encrypt, we sample a random message $M \leftarrow \mathcal{M}$ 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, $ct \leftarrow \text{PKE.Enc}(pk, M; G(M))$. The session key is then set as $K := H(M)$, where H is another hash function modeled as a (Q)RO. To decrypt, we first decrypt $M' \leftarrow \text{PKE.Dec}(sk, ct)$ and then only accept $K = H(M')$ if M' re-encrypts back to ct , that is, we check $ct = \text{PKE.Enc}(pk, M'; G(M'))$. Although the actual proof is rather complicated, intuitively, it achieves IND-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 IND-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: $\text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, M; r) \rightarrow \text{ct}$, where ct is targeted to the set of N recipients with public keys $(\text{pk}_i)_{i \in [N]}$. There is also an extraction algorithm mExt which takes as input an index $i \in [N]$ and 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 $r = G(M)$ and that recipient R_i decrypted to M . Then, to check validity of ct_i , R_i must re-encrypt the *entire* ciphertext ct by running $\text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, M; r)$. 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 $r = (r_0, r_1, \dots, r_N)$ as input, and creates a public-key-independent ciphertext $\text{ct}_0 \leftarrow \text{mEnc}^i(r_0)$ and public-key-dependent ciphertexts $\hat{\text{ct}}_i \leftarrow \text{mEnc}^d(\text{pk}_i, M; r_0, r_i)$. The resulting ciphertext for recipient R_i is then $\text{ct}_i = (\text{ct}_0, \hat{\text{ct}}_i)$. 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 $r_0 = G(M)$ and $r_i = G(\text{pk}_i, M)$. 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 [28].

However, the security proof of our mKEM in the *quantum* ROM (QROM) requires more work. Prior proof strategies in the QROM for standard IND-CCA secure KEMs based on the FO transform – which fix the description of the QROM at the outset of the game [15, 28, 33–35, 48, 50] – 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 [53]. 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 [53] 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) $\text{LWE}_{n_1, n_2, n_3}$ problem over the ring R_q is*

$$\text{Adv}_{n_1, n_2, n_3}^{\text{LWE}}(\mathcal{A}) := \left| \Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{S} \leftarrow D_s^{n_2 \times n_3}, \mathbf{E} \leftarrow D_e^{n_1 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{AS} + \mathbf{E})] - \Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{B} \leftarrow R_q^{n_1 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{B})] \right|.$$

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

Roughly, when $d = 1$ and $n_1 = n_2 \geq 1$ (with an appropriate choice of distributions D_s, D_e), the above corresponds to the standard LWE [47], and when $d \geq 1$ and $n_1 = n_2 = 1$, the above corresponds to the ring LWE (Ring-LWE) problem [40, 41]. We can parametrize d, n_1, n_2, n_3 to achieve a tradeoff between space and security, where the general case can be casted as the module LWE (Module-LWE) problem [38]. In practice, distributions D_s and D_e are chosen for instance from the discrete Gaussian distribution or from the uniform distribution (with possibly a fixed weight) over some small interval.

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 $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$, where $q > p \geq 2$, is defined as $\lfloor x \rfloor_p = \lfloor (p/q) \cdot x \rfloor \bmod p$. 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) $\text{LWR}_{n_1, n_2, n_3}$ problem over the rings R_p and R_q is

$$\text{Adv}_{n_1, n_2, n_3}^{\text{LWR}}(\mathcal{A}) := \left| \Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{S} \leftarrow D_s^{n_2 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \lfloor \mathbf{AS} \rfloor_p)] \right. \\ \left. - \Pr[\mathbf{A} \leftarrow R_q^{n_1 \times n_2}, \mathbf{B} \leftarrow R_p^{n_1 \times n_3} : 1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{B})] \right|.$$

We say the $\text{LWR}_{n_1, n_2, n_3}$ 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 [22] 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 $\text{pk}_2(s)$ 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 $\text{pk}_3(r)$ 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 [22], of the SSCDH assumption. The latter is used by one of NIST second-round candidate KEMs, called SIKE [32], 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 $\text{SSDDH}_{p, E, B_2, B_3}$ problem is

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

We say the $\text{SSCDH}_{p, E, B_2, B_3}$ 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 App. A.

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 $\mathcal{C}_{\text{single}}$ consists of the following five algorithms $\text{mPKE} = (\text{mSetup}, \text{mGen}, \text{mEnc}, \text{mExt}, \text{mDec})$:

- $\text{mSetup}(1^\kappa) \rightarrow \text{pp}$: The setup algorithm on input the security parameter 1^κ outputs a public parameter pp .
- $\text{mGen}(\text{pp}) \rightarrow (\text{pk}, \text{sk})$: The key generation algorithm on input a public parameter pp outputs a pair of public key and secret key (pk, sk) .
- $\text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{M}; r_0, r_1, \dots, r_N) \rightarrow \text{ct} = (\text{ct}_0, (\hat{\text{ct}}_i)_{i \in [N]})$: The (decomposable) encryption algorithm running with randomness (r_0, r_1, \dots, r_N) , splits into a pair of algorithms $(\text{mEnc}^i, \text{mEnc}^d)$:
 - $\text{mEnc}^i(\text{pp}; r_0) \rightarrow \text{ct}_0$: On input a public parameter pp and randomness r_0 , it outputs a (public key Independent) ciphertext ct_0 .
 - $\text{mEnc}^d(\text{pp}, \text{pk}_i, \text{M}; r_0, r_i) \rightarrow \hat{\text{ct}}_i$: On input a public parameter pp , a public key pk_i , a message $\text{M} \in \mathcal{M}$, and randomness (r_0, r_i) , it outputs a (public key Dependent) ciphertext $\hat{\text{ct}}_i$.
- $\text{mExt}(i, \text{ct}) \rightarrow \text{ct}_i = (\text{ct}_0, \hat{\text{ct}}_i)$ or \perp : The deterministic extraction algorithm on input an index $i \in \mathbb{N}$ and a (multi-recipient) ciphertext $\text{ct} \in \mathcal{C}$, outputs either a (single-recipient) ciphertext $\text{ct}_i = (\text{ct}_0, \hat{\text{ct}}_i) \in \mathcal{C}_{\text{single}}$ or a special symbol \perp_{Ext} indicating extraction failure.
- $\text{mDec}(\text{sk}, \text{ct}_i) \rightarrow \text{M}$ or \perp : The deterministic decryption algorithm on input a secret key sk and a ciphertext $\text{ct}_i \in \mathcal{C}_{\text{single}}$, outputs either $\text{M} \in \mathcal{M}$ 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 [25]. 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 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_N -correct if

$$\delta_N \geq \mathbb{E} \left[\max_{M \in \mathcal{M}} \Pr \left[\begin{array}{l} \mathbf{ct} \leftarrow \text{mEnc}(\mathbf{pp}, (\mathbf{pk}_i)_{i \in [N]}, M), \\ (\mathbf{ct}_i \leftarrow \text{mExt}(i, \mathbf{ct}))_{i \in [N]} : \\ \exists i \in [N] \text{ s.t. } M \neq \text{mDec}(\mathbf{sk}, \mathbf{ct}_i) \end{array} \right] \right], \quad (1)$$

where the expectation is taken over $\mathbf{pp} \leftarrow \text{mSetup}(1^\kappa)$ and $(\mathbf{pk}_i, \mathbf{sk}_i) \leftarrow \text{mGen}(\mathbf{pp})$ for all $i \in [N]$.

In above, when δ_N is the same for all $N \in \mathbb{N}$, we drop the subscript. We use similar conventions for the subsequent terms. We also define the notion of *well-spreadness* [25] 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 $\mathcal{C}_{\text{single}}$. For all $\mathbf{pp} \in \text{Setup}(1^\kappa)$, and $(\mathbf{pk}, \mathbf{sk}) \in \text{Gen}(\mathbf{pp})$, define

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

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

Finally, we define the notion of indistinguishability of chosen plaintext attacks (IND-CPA) for **mPKE**.

Definition 7 (IND-CPA). Let **mPKE** be a decomposable multi-recipient PKE with message space \mathcal{M} and ciphertext space \mathcal{C} . We define IND-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 IND-CPA security of **mPKE** parameterized by $N \in \mathbb{N}$ as $\text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|$.

Remark 1 (Insider corruption). We point out that insider corruptions for **mPKE** are not considered [8, 49]. 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 mEnc^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 $N = \text{poly}(\kappa)$ 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.

<u>GAME IND-CPA</u>	<u>GAME IND-CCA</u>
1: $\text{pp} \leftarrow \text{mSetup}(1^\kappa)$	1: $\text{pp} \leftarrow \text{mSetup}(1^\kappa)$
2: for $i \in [N]$ do	2: for $i \in [N]$ do
3: $(\text{pk}_i, \text{sk}_i) \leftarrow \text{mGen}(\text{pp})$	3: $(\text{pk}_i, \text{sk}_i) \leftarrow \text{mGen}(\text{pp})$
4: $(\text{M}_0^*, \text{M}_1^*, \text{state}) \leftarrow \mathcal{A}_1(\text{pp}, (\text{pk}_i)_{i \in [N]})$	4: $(\text{K}_0^*, \text{ct}^*) \leftarrow \text{mEncaps}(\text{pp}, (\text{pk}_i)_{i \in [N]})$
5: $b \leftarrow \{0, 1\}$	5: $\text{K}_1^* \leftarrow \mathcal{K}$
6: $\text{ct}^* \leftarrow \text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{M}_b^*)$	6: $b \leftarrow \{0, 1\}$
7: $b' \leftarrow \mathcal{A}_2(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{ct}^*, \text{state})$	7: $b' \leftarrow \mathcal{A}^{\mathcal{D}}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{ct}^*, \text{K}_b^*)$
8: return $[b = b']$	8: return $[b = b']$

<u>Decapsulation Oracle $\mathcal{D}(i, \text{ct})$</u>
1: $\text{ct}_i^* := \text{mExt}(i, \text{ct}^*)$
2: if $\text{ct} = \text{ct}_i^*$ then
3: return \perp
4: $\text{K} := \text{mDecaps}(\text{sk}_i, \text{ct})$
5: return K

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

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 $\text{mKEM} = (\text{mSetup}, \text{mGen}, \text{mEncaps}, \text{mExt}, \text{mDecaps})$:

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

- $\text{mDecaps}(\text{sk}, \text{ct}_i) \rightarrow K \text{ or } \perp$: The deterministic decryption algorithm on input a secret key sk and a ciphertext ct_i , outputs either $K \in \mathcal{K}$ or a special symbol $\perp \notin \mathcal{K}$.

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

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

where the probability is taken over $\text{pp} \leftarrow \text{mSetup}$ and $(\text{pk}_i, \text{sk}_i) \leftarrow \text{mGen}(\text{pp})$ 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 $\text{Adv}_{\text{mKEM}, N}^{\text{IND-CCA}}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|$.

We note that similarly to the remark made in Rem. 2, the goal is to obtain a mKEM with better efficiency/ciphertext-size compared to the trivial method of running a standard single-recipient KEM in parallel.

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 $\overline{\text{mEnc}}$ ($\overline{\text{mEncaps}}$), 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 App. A.3.

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, G_1, G_2, H, 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

<u>mSetup(1^κ)</u>	<u>mGen(pp)</u>	<u>mExt(i, ct)</u>
1: $\text{pp} \leftarrow \text{mSetup}^{\text{P}}(1^\kappa)$	1: $(\text{pk}, \text{sk}^{\text{P}}) \leftarrow \text{mGen}^{\text{P}}(\text{pp})$	1: $\text{ct}_i \leftarrow \text{mExt}^{\text{P}}(i, \text{ct})$
2: return pp	2: $\text{seed} \leftarrow \{0, 1\}^\ell$	2: return ct_i
	3: $\text{sk} := (\text{sk}^{\text{P}}, \text{seed})$	
	4: return (pk, sk)	
<u>mEncaps(pp, $(\text{pk}_i)_{i \in [N]}$)</u>	<u>mDecaps(sk, ct)</u>	
1: $M \leftarrow \mathcal{M}$	1: $\text{sk} := (\text{sk}^{\text{P}}, \text{seed})$	
2: $\text{ct}_0 := \text{mEnc}^{\text{i}}(\text{pp}; G_1(M))$	2: $M := \text{mDec}(\text{sk}^{\text{P}}, \text{ct})$	
3: for $i \in [N]$ do	3: if $M = \perp$ then	
4: $\hat{\text{ct}}_i := \text{mEnc}^{\text{d}}(\text{pp}, \text{pk}_i, M;$	4: return $K := H'(\text{seed}, \text{ct})$	
$G_1(M), G_2(\text{pk}_i, M))$	5: $\text{ct}_0 := \text{mEnc}^{\text{i}}(\text{pp}; G_1(M))$	
5: $K := H(M)$	6: $\hat{\text{ct}} := \text{mEnc}^{\text{d}}(\text{pp}, \text{pk}, M; G_1(M), G_2(\text{pk}, M))$	
6: return $(K, \text{ct} := (\text{ct}_0, (\hat{\text{ct}}_i)_{i \in [N]}))$	7: if $\text{ct} \neq (\text{ct}_0, \hat{\text{ct}})$ then	
	8: return $K := H'(\text{seed}, \text{ct})$	
	9: else	
	10: return $K := H(M)$	

Fig. 2: An IND-CCA secure mKEM from a decomposable IND-CPA secure mPKE = (mSetup^P, mGen^P, mEnc = (mEncⁱ, mEnc^d), mExt^P, mDec). We include the superscript ^P to make the code more readable.

separation. Finally, we include an ℓ -bit seed to perform implicit rejection by viewing $H'(\text{seed}, \cdot)$ as a pseudorandom function in the (Q)ROM.

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_N -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 $q_{\mathcal{G}}$ queries to G_1 and G_2 , and at most $q_{\mathcal{H}}, q'_{\mathcal{H}}$ queries to H and H' , there exists a classical PPT adversary \mathcal{B}_{IND} such that*

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

where the running time of \mathcal{B}_{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_N -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 $q_{\mathcal{G}}$ quantum queries to G_1 and G_2 , and at most $q_{\mathcal{H}}, q'_{\mathcal{H}}$*

quantum queries to H and H' , there exists a quantum PT adversary \mathcal{B}_{IND} such that

$$\begin{aligned} \text{Adv}_{\text{mKEM}, N}^{\text{IND-CCA}}(\mathcal{A}) &\leq 2 \cdot \sqrt{q_G \cdot \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + \frac{8q_G^2 \cdot (N+1)}{|\mathcal{M}|}} \\ &\quad + 8(q_G + q_D + 1)^2 \cdot \delta_N + q_D \cdot (8\sqrt{2^{-\gamma}} + 2^{-\gamma}) + q'_H \cdot N \cdot 2^{-\frac{\ell+1}{2}}, \end{aligned}$$

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

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 $M = \perp$ or ct is not the same as the reencrypted ciphertext $(\text{ct}_0, \widehat{\text{ct}}_i)$. Concretely, we obtain an IND-CCA secure mKEM with explicit rejection by simply outputting \perp rather than outputting $H'(\text{seed}, \text{ct})$ 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, 48]).

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 .

- **Game₁**: This is the real IND-CCA security game: $\text{Adv}_{\text{mKEM}, N}^{\text{IND-CCA}}(\mathcal{A}) = |\Pr[E_1] - 1/2|$.

- **Game₂**: In this game, we replace the computation of $H'(\text{seed}_i, \cdot)$ by a random function $\widehat{H}'_i(\cdot)$ in case $M = \perp$ or $\text{ct} \neq (\text{ct}_0, \widehat{\text{ct}})$ occurs when answering the decapsulation oracle with input $i \in [N]$. Here, $\widehat{H}'_i(\cdot)$ is a random function that cannot be accessed by the adversary. Since this modification remains unnoticed by the adversary unless $H'(\text{seed}, \cdot)$ is queried for any $\text{seed} \in \{\text{seed}_i\}_{i \in [N]}$, we have $|\Pr[E_1] - \Pr[E_2]| \leq \frac{q'_H \cdot N}{2^\ell}$.

- **Game₃**: 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 $|\Pr[E_2] - \Pr[E_3]| \leq (q_G + q_D + 1) \cdot \delta_N$.

(The next **Game₄**, **Game₅** and **Game₆** aim to get rid of the secret keys sk_i to answer \mathcal{A} 's decapsulation oracle queries.)

- **Game₄**: 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}_G is a list that stores the random oracle

<u>Game₄ : Decap. Oracle $\mathcal{D}(i, \text{ct} \neq \text{ct}_i^*)$</u>	<u>Game₅ : Decap. Oracle $\mathcal{D}(i, \text{ct} \neq \text{ct}_i^*)$</u>
1: $\text{sk}_i := (\text{sk}_i^p, \text{seed}_i)$	1: for $M \in \mathcal{L}_G$ do
2: $M := \text{mDec}(\text{sk}_i^p, \text{ct})$	2: $\text{ct}_0 := \text{mEnc}^i(\text{pp}; G_1(M))$
3: if $M \notin \mathcal{L}_G$ then	3: $\hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, \text{pk}_i, M;$
4: <u>return $K := \hat{H}'_i(\text{ct})$</u>	$G_1(M), G_2(\text{pk}_i, M))$
5: if $M = \perp$ then	4: if $\text{ct} = (\text{ct}_0, \hat{\text{ct}}_i)$ then
6: return $K := \hat{H}'_i(\text{ct})$	5: return $K := H(M)$
7: $\text{ct}_0 := \text{mEnc}^i(\text{pp}; G_1(M))$	6: return $K := \hat{H}_i(\text{ct})$
8: $\hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, \text{pk}_i, M; G_1(M), G_2(\text{pk}_i, M))$	
9: if $\text{ct} \neq (\text{ct}_0, \hat{\text{ct}}_i)$ then	
10: return $K := \hat{H}'_i(\text{ct})$	
11: else	
12: return $K := H(M)$	

Fig. 3: Decapsulation oracles of Game₄ and Game₅. We enforce ct is not $\text{ct}_i^* := \text{mExt}(i, \text{ct}^*)$ at the input level for simplicity.

queries made to G_1 and G_2 . We have $M \in \mathcal{L}_G$ if either G_1 was queried on M or G_2 was queried on (pk, M) for any pk . The only difference occurs when \mathcal{A} queries a ciphertext $\text{ct} = (\text{ct}_0, \hat{\text{ct}}_i)$ such that $M := \text{mDec}(\text{sk}_i^p, \text{ct})$ has not been queried to the random oracles G_1 and G_2 but $\text{ct}_0 = \text{mEnc}^i(\text{pp}; G_1(M))$ and $\hat{\text{ct}}_i = \text{mEnc}^d(\text{pp}, \text{pk}_i, M; G_1(M), G_2(\text{pk}_i, M))$. Since $G_1(M)$ and $G_2(\text{pk}_i, M)$ are information theoretically hidden from \mathcal{A} , we can use γ -spreadness of mPKE to conclude $|\Pr[E_3] - \Pr[E_4]| \leq q_{\mathcal{D}} \cdot 2^{-\gamma}$.

- **Game₅**: 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 Game₄ outputs $K := H(M)$, then $M \in \mathcal{L}_G$ and $\text{ct} = (\text{ct}_0, \hat{\text{ct}}_i)$ holds. Therefore, the decapsulation oracle in Game₅ outputs K as well. On the other hand, assume the decapsulation oracle in Game₅ outputs $K := H(M)$ for some $M \in \mathcal{L}_G$ such that $\text{ct} = (\text{ct}_0, \hat{\text{ct}}_i)$ where $\text{ct}_0 := \text{mEnc}^i(\text{pp}; G_1(M))$ and $\hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, \text{pk}_i, M; G_1(M), G_2(\text{pk}_i, M))$. Then, since we have no correctness error (due to Game₃), ct must decrypt to M . Hence, this implies that the decapsulation oracle Game₄ outputs the same K as well. Combining the arguments together, we get $\Pr[E_4] = \Pr[E_5]$.

- **Game₆**: In this game, we undo the change we made in Game₃ 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 $|\Pr[E_5] - \Pr[E_6]| \leq (q_G + 1) \cdot \delta_N$.

(The following final Game₇ aims to get rid of M^* in the challenge ciphertext.)

- **Game₇**: In this game, we sample the random message $M^* \leftarrow \mathcal{M}$ 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 $H(\cdot)$, $G_1(\cdot)$, or $G_2(\star, \cdot)$ 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 Lem. 4 that we have

$|\Pr[E_6] - \Pr[E_7]| \leq 2 \cdot \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + \frac{(q_G + q_H)}{|\mathcal{M}|}$ for some classical PPT adversary \mathcal{B}_{IND} with similar runtime as \mathcal{A} . The proof of Lem. 4 is given in App. B.1.

In Game_7 , the view of the adversary is independent of the challenge bit b . Therefore, we have $\Pr[E_7] = \frac{1}{2}$. 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 App. D, and we refer for more details to other works, such as [6, 21, 31, 53]. The main difference between our proof and prior proofs for IND-CCA secure KEM in the QROM, e.g., [15, 28, 33–35, 48, 50], is that we use the *lazy sampling* with *compressed* quantum oracles introduced in [53]. 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 App. B.2 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 mEncaps (see Sec. 3.3) as: sample $K \leftarrow \mathcal{K}$, run $\text{ct} \leftarrow \text{mEnc}(\text{pp}, N)$, and output (K, ct) , where 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 ct^* is distributed independently of the public keys $(\text{pk}_i)_{i \in [N]}$. 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 [39] 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 [39] 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_N -correctness and γ -spreadness for exponentially small δ_N and $2^{-\gamma}$. Moreover, practical schemes such as NIST candidates also allow for exponentially small δ_N 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 App. C.1 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*

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

Moreover, as a simple consequence of the proof of the above lemma, we have IND-Anon-CPA for free. In particular, the fake encryption algorithm **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 $\lfloor \mathbf{AS} \rfloor_p \in R_p^{n \times m}$ rather than $\mathbf{AS} + \mathbf{E} \in R_q^{n \times m}$. It is easy to show that mPKE has γ -spreadness, is δ_N -correct and IND-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 [22] 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 $\mathcal{H} = \{H_k : \mathcal{X} \subset \mathbb{F} \rightarrow \{0, 1\}^w\}_{k \in K}$ 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 App. C.

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

Input: Security parameter 1^κ
Output: Public parameter pp
1: $\mathbf{A} \leftarrow R_q^{n \times n}$
2: **return** $\text{pp} := \mathbf{A}$

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

Input: Public parameter $\text{pp} = \mathbf{A}$
Output: Public key pk , a secret key sk
1: $\mathbf{S} \leftarrow D_s^{n \times m}$
2: $\mathbf{E} \leftarrow D_e^{n \times m}$
3: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E} \quad \triangleright \mathbf{B} \in R_q^{n \times m}$
4: **return** $\text{pk} := \mathbf{B}, \text{sk} := \mathbf{S}$

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

Input: Public parameter $\text{pp} = \mathbf{A}$, set of public keys $(\text{pk}_i = \mathbf{B}_i)_{i \in [N]}$, message M
Output: Ciphertext $\text{ct} = (\text{ct}_0, (\hat{\text{ct}}_i)_{i \in [N]})$
1: $\mathbf{r}_0 := (\mathbf{R}, \mathbf{E}') \leftarrow D_s^{\bar{m} \times n} \times D_e^{\bar{m} \times n}$
2: $\text{ct}_0 := \text{mEnc}^l(\text{pp}; \mathbf{r}_0)$
3: **for** $i \in [N]$ **do**
4: $\mathbf{r}_i := \mathbf{E}_i'' \leftarrow D_e^{\bar{m} \times m}$
5: $\hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, \text{pk}_i, \text{M}; \mathbf{r}_0, \mathbf{r}_i)$
6: **return** $\text{ct} := (\text{ct}_0, \hat{\text{ct}}_1, \dots, \hat{\text{ct}}_N)$

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

Input: Public parameter $\text{pp} = \mathbf{A}$, public key $\text{pk}_i = \mathbf{B}_i$, message M , randomness $\mathbf{r}_0 = (\mathbf{R}, \mathbf{E}')$ and $\mathbf{r}_i = \mathbf{E}_i''$
Output: (Public key dependent) ciphertext $\hat{\text{ct}}_i$
1: $\mathbf{V}_i \leftarrow \mathbf{RB}_i + \mathbf{E}_i'' + \text{Encode}(\text{M}) \quad \triangleright \mathbf{V}_i \in R_q^{\bar{m} \times m}$
2: **return** $\hat{\text{ct}}_i := \mathbf{V}_i$

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

Input: Public parameter $\text{pp} = \mathbf{A}$, randomness $\mathbf{r}_0 = (\mathbf{R}, \mathbf{E}')$
Output: (Public key independent) ciphertext ct_0
1: $\mathbf{U} \leftarrow \mathbf{RA} + \mathbf{E}' \quad \triangleright \mathbf{U} \in R_q^{\bar{m} \times n}$
2: **return** $\text{ct}_0 := \mathbf{U}$

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

Input: Secret key $\text{sk} = \mathbf{S}$, ciphertext $\text{ct} = (\mathbf{U}, \mathbf{V})$
Output: Message M
1: $\mathbf{M} \leftarrow \mathbf{V} - \mathbf{US} \quad \triangleright \mathbf{M} \in R_q^{\bar{m} \times m}$
2: **return** $\text{M} := \text{Decode}(\mathbf{M})$

Fig. 4: Lattice-based mPKE via the Lindner-Peikert framework [39]. mExt with input index i is defined by picking the relevant components $(\text{ct}_0, \hat{\text{ct}}_i)$ from ct .

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 $\mathcal{H} = \{\text{H}_k : \mathcal{X} \subset \mathbb{F}_{p^2} \rightarrow \{0, 1\}^w\}_{k \in K}$. 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 $\epsilon \in \{0, 1, 2\}$ [22]. 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 $\mathbf{pp} \leftarrow \mathbf{mGen}(1^\kappa)$ is fixed [32]. The first component of each element in $\mathcal{C}_{\text{single}}$ contains a curve 2^{e_2} -isogenous to E . We denote by W the set $\{j(E/\langle P_2 + [r]Q_2 \rangle) \mid r \in \mathbb{Z}_{2^{e_2}}\}$ 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 $\gamma(\mathbf{pp}, \mathbf{pk}) \geq e_2$. This value is independent of the public key \mathbf{pk} and E, B_2, B_3 in \mathbf{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 [32]).

IND-(Anon-)CPA Security. The IND-CPA security of the SIDH-based mPKE follows from the SSDDH assumption and the Leftover Hash Lemma (see App. A.2). The proof of the following lemma is given in App. C.2 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*

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

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 [32]. 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 $\overline{\mathbf{mEnc}}$ 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 App. C.3. The scheme in Fig. 10 considers 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 (see Rem. 7).

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 CSIDH mPKE, for reasons explained later. We compare these mKEMs to

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

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: $H \leftarrow \mathcal{H}$
 - 3: **return** $\text{pp} := (E, \{(e_j, B_j)\}_{j=2,3}, H)$
-

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

Input: Public parameter $\text{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, H)$ **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** $\text{pk} := (E_3, U_2, V_2), \text{sk} := s$
-

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

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

- 1: $r_0 := r \leftarrow \mathbb{Z}_{2^{e_2}}$
 - 2: $\text{ct}_0 := \text{mEnc}^i(\text{pp}; r_0)$
 - 3: **for** $i \in [N]$ **do**
 - 4: $\hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, \text{pk}_i, M; r_0)$
 - 5: **return** $\text{ct} := (\text{ct}_0, \hat{\text{ct}}_1, \dots, \hat{\text{ct}}_N)$
-

Algorithm 10 $\text{mEnc}^d(\text{pp}, \text{pk}_i, M; r_0)$

Input: Public parameter $\text{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, H)$, public key $\text{pk}_i = (E_3^{(i)}, U_2^{(i)}, V_2^{(i)})$, message M , randomness $r_0 = r$ **Output:** (Public key dependent) ciphertext $\hat{\text{ct}}_i$

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

Algorithm 11 $\text{mEnc}^i(\text{pp}; r_0)$

Input: Public parameter $\text{pp} = (E, \{(e_j, B_j)\}_{j=2,3}, H)$, randomness $r_0 = r$ **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** $\text{ct}_0 := (E_2, U_3, V_3)$
-

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

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

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

Fig. 5: SIDH-based mPKE via hashed ElGamal [22]. mExt with input index i is defined by picking the relevant components $(\text{ct}_0, \hat{\text{ct}}_i)$ from ct . Note that mEnc^d does not require any randomness r_i for $i \in [N]$.

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 $|x|$ 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:

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

This ratio converges to a value independent of N when N tends to infinity. Indeed, with mKEM the encryptor broadcasts a single multi-ciphertext of size $|\text{ct}_0| + \sum_{i \in [N]} |\widehat{\text{ct}}_i| = |\text{ct}_0| + N|\widehat{\text{ct}}_i|$, whereas with N instances of KEM he broadcasts N ciphertexts $\text{ct} = (\text{ct}_0, \widehat{\text{ct}}_i)$ – except for NewHope, see footnote 4 – for a total size $N|\text{ct}_0| + N|\widehat{\text{ct}}_i|$. The value (3) is the ratio of these two sizes, that is:

$$\frac{N|\text{ct}_0| + N|\widehat{\text{ct}}_i|}{|\text{ct}_0| + N|\widehat{\text{ct}}_i|} \xrightarrow{N \rightarrow \infty} 1 + \frac{|\text{ct}_0|}{|\widehat{\text{ct}}_i|}. \quad (4)$$

Let $k_{\text{comm}} = 1 + \frac{|\text{ct}_0|}{|\widehat{\text{ct}}_i|}$. 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):

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

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

Full specifications of these 7 schemes are available at [44]. 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 $\text{Module Integer-LWE}$. In addition, each scheme has different parameters and uses different tweaks. A widespread trick is for $\hat{\text{ct}}_i$ to drop the least significant bits of \mathbf{V}_i , since the message \mathbf{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 $|\hat{\text{ct}}_i|$, 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 $N \cdot |\text{ct}|$ bytes for a NIST KEM, whereas using its mKEM counterpart costs $|\text{ct}_0| + N \cdot |\hat{\text{ct}}_i|$. 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).

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

Scheme	$ \text{ct}_0 $	$ \widehat{\text{ct}}_i $	$ \text{ct} $	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 ⁴	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

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^{5,6}, 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. Additional experiments in App. E 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, we compare the mPKE of Fig. 10 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

⁵ <https://github.com/Microsoft/PQCrypto-LWEKE>

⁶ <https://github.com/pq-crystals/kyber/>

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

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

compare it with our mPKE from Sec. 5.2 (see also Fig. 10 for the details). To obtain proof-of-concept implementation of mPKE for CSIDH and mKEM for SIKE, we have modified implementation available in the NOBS library⁷.

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	$ \text{ct}_0 $	$ \widehat{\text{ct}}_i $	$ \text{ct} $	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

⁷ <https://github.com/henrydcase/nobs>

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 $|\text{ct}_0|$ is a slower operation than computing $|\widehat{\text{ct}}_i|$.

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 CGKA = (Init, Create, Add, Remove, Update, 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 $(\text{ID}_i)_{i \in [N]}$ 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.** 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 **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, $\text{siblings}(u)$, is the set of nodes $v \neq u$ in T with the same parent as u . Its *path*, $\text{path}(u)$, is the set of nodes between u and root , including u but excluding root . Its *co-path*, $\text{copath}(u)$, is the set of siblings of nodes in its path: $\text{copath}(u) = \bigcup_{v \in \text{path}(u)} \text{siblings}(v)$. 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 $(pk_i, sk_i) = \text{mGen}(\text{pp}; \text{PRG}(\text{seed}_i)_L)$, where $\text{PRG}(\cdot)_L$ (resp. $\text{PRG}(\cdot)_R$) denotes the left (resp. right) half output of the PRG. In particular, mGen is run on randomness $\text{PRG}(\text{seed}_i)_L$. 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 $i \in \text{path}(u)$. 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 $i \in \text{path}(u)$, update its keypair: $(pk_i, sk_i) = \text{mGen}(\text{pp}; \text{PRG}(\text{seed}_i)_L)$, and compute a new secret seed for its parent: $\text{seed}_{\text{parent}(i)} = \text{PRG}(\text{seed}_i)_R$.
- (U3) For each $i \in \text{path}(u)$, compute the ciphertext

$$\text{ct}_i \leftarrow \text{mEncaps}(\text{pp}, (pk_j)_{j \in \text{siblings}(i)}; \text{seed}_{\text{parent}(i)}). \quad (6)$$

Note that mEncaps is derandomized here. For our construction in Fig. 2, this is equivalent to setting the random message $M_i = \text{PRG}(\text{seed}_{\text{parent}(i)})$.

- (U4) Send the update package $(pk_i, \text{ct}_i)_{i \in \text{path}(u)}$ 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 **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 ct_i .
- (P3) Recover the secret seeds of all remaining common ancestors of u and v by computing $\text{seed}_{\text{parent}(i)} = \text{PRG}(\text{seed}_i)_R$. The update secret is $I = \text{seed}_{\text{root}}$

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:

$$\lceil \log_2(N) \rceil \cdot (|pk| + |ct_0| + |\widehat{ct}_i|) \quad (7)$$

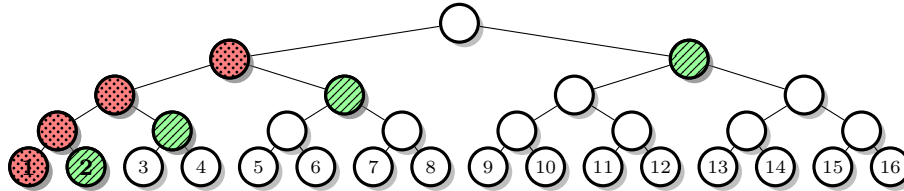


Fig. 6: TreeKEM

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).

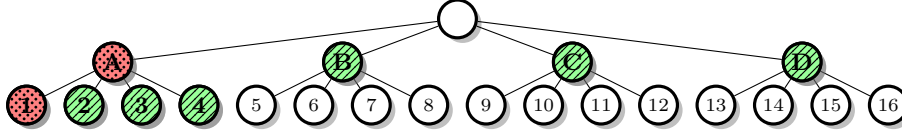


Fig. 7: 4-ary TreeKEM

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 $i \in \text{path}(u)$, the update package encapsulates $\text{seed}_{\text{parent}(i)}$ under the key pk_j for each $j \in \text{siblings}(i)$. In the example of Fig. 7, this means that an update package issued by 1 encapsulates seed_A under $\text{pk}_2, \text{pk}_3, \text{pk}_4$, and $\text{seed}_{\text{root}}$ under $\text{pk}_B, \text{pk}_C, \text{pk}_D$. The bandwidth gain happens exactly here: since the same value seed_A is encapsulated under $\text{pk}_2, \text{pk}_3, \text{pk}_4$, 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:

$$\lceil \log_m(N) \rceil \cdot (|\text{pk}| + |\text{ct}_0| + (m-1) \cdot |\widehat{\text{ct}}_i|). \quad (8)$$

One can see that (8) generalizes (7) to any integer $m > 2$. It is clear from (8) that whenever $|\text{pk}| + |\text{ct}_0| \gg |\widehat{\text{ct}}_i|$, 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 $m-1 \approx \frac{|\text{pk}| + |\text{ct}_0|}{|\widehat{\text{ct}}_i|}$. According to (7), the bytesize of an update package for binary TreeKEM will then be approximately $\lceil \log_2(N) \rceil \cdot m \cdot |\widehat{\text{ct}}_i|$. On the other hand, the bytesize – given by (8) – for our proposal is about $\lceil \log_m(N) \rceil \cdot 2(m-1) \cdot |\widehat{\text{ct}}_i|$. Compared to the standard TreeKEM, our proposal improves communication cost by a factor equal to the ratio of the two values, which is approximately:

$$\frac{\lceil \log_2(N) \rceil \cdot m \cdot |\widehat{\text{ct}}_i|}{\lceil \log_m(N) \rceil \cdot 2(m-1) \cdot |\widehat{\text{ct}}_i|} \xrightarrow{N \rightarrow \infty} \frac{m}{2(m-1)} \cdot \log_2(m) = O(\log m).$$

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.

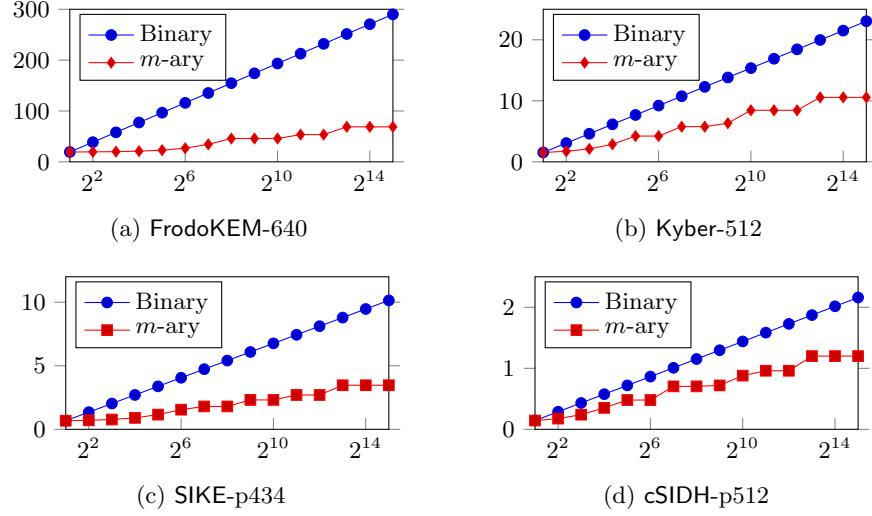


Fig. 8: Comparing the classic “binary” TreeKEM with m -ary TreeKEM, when instantiated with four schemes: FrodoKEM, Kyber, SIKE and cSIDH. In each case, the x -axis represent the number N of group members (from 2 to 2^{15}) and the y -axis represent the maximal size of an update package in kilobytes.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

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.
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 multi-cast 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 schemes. *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. D. Cozzo and N. P. Smart. Sashimi: Cutting up csi-fish secret keys to produce an actively secure distributed signing protocol. In *PQCRYPTO 2020*, pp. 169–186.
20. 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.
21. J. Czajkowski, C. Majenz, C. Schaffner, and S. Zur. Quantum lazy sampling and game-playing proofs for quantum indistinguishability. *Cryptology ePrint Archive*, Report 2019/428.
22. 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.
23. A. W. Dent. A designers guide to kems. In *IMA International Conference on Cryptography and Coding*, pp. 133–151. Springer.
24. A. El Kaafarani, S. Katsumata, and F. Pintore. Lossy csi-fish: Efficient signature scheme with tight reduction to decisional csidh-512. In *PKC 2020*, pp. 157–186.

25. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *CRYPTO'99*, pp. 537–554.
26. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396.
27. H. Hiwatari, K. Tanaka, T. Asano, and K. Sakumoto. Multi-recipient public-key encryption from simulators in security proofs. *ACISP 09*, pp. 293–308.
28. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. *TCC 2017*, pp. 341–371.
29. D. Hofheinz and E. Kiltz. Secure hybrid encryption from weakened key encapsulation. *CRYPTO 2007*, pp. 553–571.
30. A. Hosoyamada and T. Iwata. 4-round Luby-Rackoff construction is a qPRP. In *ASIACRYPT 2019*, pp. 145–174.
31. K. Hövelmanns, E. Kiltz, S. Schäge, and D. Unruh. Generic authenticated key exchange in the quantum random oracle model. In *PKC (to appear)*. Springer.
32. 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>.
33. 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.
34. H. Jiang, Z. Zhang, and Z. Ma. Key encapsulation mechanism with explicit rejection in the quantum random oracle model. *PKC 2019*, pp. 618–645.
35. 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.
36. 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*, pp. 703–728. Springer.
37. K. Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. *PKC 2002*, pp. 48–63.
38. A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599.
39. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. *CT-RSA 2011*, pp. 319–339.
40. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *EUROCRYPT 2010*, pp. 1–23.
41. V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. *EUROCRYPT 2013*, pp. 35–54.
42. T. Matsuda and G. Hanaoka. Key encapsulation mechanisms from extractable hash proof systems, revisited. *PKC 2013*, pp. 332–351.
43. M. A. Nielsen and I. Chuang. Quantum computation and quantum information.
44. NIST. Post-quantum cryptography - round 2 submissions. <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
45. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. *CT-RSA 2001*, pp. 159–175.
46. 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.

47. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *37th ACM STOC*, pp. 84–93.
48. T. Saito, K. Xagawa, and T. Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. *EUROCRYPT 2018*, pp. 520–551.
49. N. P. Smart. Efficient key encapsulation to multiple parties. *SCN 04*, pp. 208–219.
50. E. E. Targhi and D. Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. *TCC 2016-B*, pp. 192–216.
51. D. Unruh. Revocable quantum timed-release encryption. *EUROCRYPT 2014*, pp. 129–146.
52. Z. Yang. On constructing practical multi-recipient key-encapsulation with short ciphertext and public key. *SCN*, 8(18):4191–4202.
53. M. Zhandry. How to record quantum queries, and applications to quantum indistinguishability. *CRYPTO 2019*, pp. 239–268.

Supplementary Material

A Omitted Preliminaries

A.1 CSIDH-based assumption

Let p be an odd prime of the form $4\ell_1\ell_2\cdots\ell_t - 1$, where ℓ_1, \dots, ℓ_t are small odd primes. $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ is the set containing all supersingular elliptic curves E over \mathbb{F}_p - modulo isomorphisms over \mathbb{F}_p - such that there exists an isomorphism between the order $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{-p})$ and $\text{End}_p(E)$ mapping $\sqrt{-p} \in \mathcal{O}$ into the Frobenius endomorphism $(x, y) \mapsto (x^p, y^p)$. We note that $|\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)| \approx \sqrt{p}$. We consider \mathcal{O} equal to $\mathbb{Z}[\sqrt{-p}] = \{m + n\sqrt{-p} \mid m, n \in \mathbb{Z}\}$, in which case, provided that $p \equiv 3 \pmod{8}$, each isomorphism class in $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ can be uniquely represented by a single element of \mathbb{F}_p [16]. The ideal class group $\mathcal{C}\ell(\mathcal{O})$ acts freely and transitively on $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$; we denote \star such action. A special integral ideal $\mathfrak{J}_{\ell_i} \subset \mathcal{O}$ corresponds to each prime ℓ_i . These ideals allow an easy computation of the group action, since the action of \mathfrak{J}_{ℓ_i} on a curve $E \in \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ is determined by an isogeny having as kernel the unique rational ℓ_i -torsion subgroup of E .

In the following we restrict our attention to the case where $\mathcal{C}\ell(\mathcal{O})$ is cyclic and generated by $\mathfrak{g} \in \{[\mathfrak{J}_{\ell_i}] \mid i = 1, \dots, t\}$. The security of the CSIDH-based scheme detailed in Fig. 10 relies on the decisional variant, recently exploited in [19, 24], of the best-known GAIP assumption [16].

Definition 12 (Decisional CSIDH (dCSIDH)). *Let p be a prime of the form $4\ell_1\ell_2\cdots\ell_t - 1$ and \mathfrak{g} a generator of the ideal class group $\mathcal{C}\ell(\mathcal{O})$ having order N , where $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$. We say that the advantage of algorithm \mathcal{A} in solving the $\text{dCSIDH}_{p, \mathfrak{g}}$ ⁸ problem is*

$$\begin{aligned} \text{Adv}_{p, \mathfrak{g}}^{\text{dCSIDH}}(\mathcal{A}) := & \left| \Pr[E \leftarrow \mathcal{E}\ell\ell_p(\mathcal{O}, \pi), (a, b) \leftarrow (\mathbb{Z}_N)^2 : \right. \\ & 1 \leftarrow \mathcal{A}(E, \mathfrak{g}^a \star E, \mathfrak{g}^b \star E, \mathfrak{g}^a \star (\mathfrak{g}^b \star E))] \\ & - \Pr[E \leftarrow \mathcal{E}\ell\ell_p(\mathcal{O}, \pi), (a, b, c) \leftarrow (\mathbb{Z}_N)^3 : \\ & \left. 1 \leftarrow \mathcal{A}(E, \mathfrak{g}^a \star E, \mathfrak{g}^b \star E, \mathfrak{g}^c \star E,) \right] \right|. \end{aligned}$$

We say the $\text{dCSIDH}_{p, \mathfrak{g}}$ problem is hard if for any (possibly quantum) efficient adversary \mathcal{A} , its advantage is negligible.

Three sets of CSIDH parameters have been proposed so far, namely CSIDH-512, CSIDH-1024 and CSIDH-1792. Only for CSIDH-512 the structure of the corresponding group $\mathcal{C}\ell(\mathcal{O})$ is known, and it adheres to the assumption made above, i.e. $\mathcal{C}\ell(\mathcal{O})$ cyclic and generated by $\mathfrak{g} \in \{[\mathfrak{J}_{\ell_i}] \mid i = 1, \dots, t\}$. For the other two sets of parameters, efficiency reasons dictate to restrict to elements of $\mathcal{C}\ell(\mathcal{O})$

⁸ We intentionally put d in dCSIDH so as not to confuse the CSIDH key exchange protocol with the assumption.

of the form $[\mathcal{J}_{\ell_i}]^{e_i} \dots [\mathcal{J}_{\ell_t}]^{e_t}$, with the integers e_i satisfying the bound $|e_i| \leq B$, where B is a suitable (small) natural number. Sampling uniformly from the set $\{[\mathcal{J}_{\ell_i}]^{e_i} \dots [\mathcal{J}_{\ell_t}]^{e_t} \mid |e_i| \leq B\}$ determines a distribution D_B over $\mathcal{Cl}(\mathcal{O})$ and, fixing $E_0 \in \mathcal{Ell}_p(\mathcal{O}, \pi)$, a distribution D_{E_0} over $\mathcal{Ell}_p(\mathcal{O}, \pi)$ itself. The distribution D_B is heuristically assumed to be close to the uniform distribution over $\mathcal{Cl}(\mathcal{O})$. The dCSIDH (and the GAIP) problem is (are) believed to be hard also in this more general case. However, its formulation requires to be slightly modified. Indeed, fixed a curve E_0 in $\mathcal{Ell}_p(\mathcal{O}, \pi)$, the curve E is sampled from D_{E_0} , while elements sampled from D_B replace the powers of g .

A.2 Randomness Extraction

The *min-entropy* of a random variable X is defined as $\mathbf{H}_\infty(X) = -\log_2(\max_x \Pr[X = x])$. We recall the definition of family of universal hash functions.

Definition 13 (Universal Hash Functions). *A family of functions $\mathcal{H} = \{H_k : \mathcal{X} \rightarrow \mathcal{D}\}_{k \in K}$ is called a family of universal hash functions if, for all $x, x' \in \mathcal{X}$ with $x \neq x'$, we have $\Pr_{H \leftarrow \mathcal{H}}[H(x) = H(x')] \leq \frac{1}{|\mathcal{D}|}$.*

It is well known that one can extract uniform random bits from a high min-entropy source using universal hash functions [26].

Lemma 3 (Leftover Hash Lemma). *Let $\mathcal{H} = \{H_k : \mathcal{X} \rightarrow \mathcal{D}\}_{k \in K}$ be a family of universal hash functions. Let H be sampled uniformly from \mathcal{H} , X be a random variable independent of H and with values in \mathcal{X} , and $U(\mathcal{D})$ be the uniform distribution over \mathcal{D} . Then, the following holds*

$$\Delta((H, H(X)), (H, U(\mathcal{D}))) \leq \frac{1}{2} \cdot \sqrt{2^{-\mathbf{H}_\infty(X)} \cdot |\mathcal{D}|}.$$

where Δ denotes the statistical distance.

A.3 Recipient Anonymity for mPKE and mKEM

We provide the formal definitions of recipient anonymity.

Definition 14 (IND-Anon-CPA). *Let mPKE be a decomposable multi-recipient PKE. We define IND-Anon-CPA associated with a PPT fake encryption algorithm $\overline{\text{mEnc}}$ by a game illustrated in Fig. 9 and say the (possibly quantum) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ wins if the game outputs 1. For an adversary \mathcal{A} , we define its advantage against IND-Anon-CPA security of mPKE parameterized by $N \in \mathbb{N}$ as*

$$\text{Adv}_{\text{mPKE}, N}^{\text{IND-Anon-CPA}}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|.$$

Definition 15 (IND-Anon-CCA). *Let mKEM be a multi-recipient KEM. We define IND-Anon-CCA associated with a PPT fake encryption algorithm $\overline{\text{mEncaps}}$ by a game illustrated in Fig. 9 and say the (possibly quantum) adversary \mathcal{A} wins if the game outputs 1. For an adversary \mathcal{A} , we define its advantage against IND-Anon-CCA security of mKEM parameterized by $N \in \mathbb{N}$ as*

$$\text{Adv}_{\text{mKEM}, N}^{\text{IND-Anon-CCA}}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|.$$

GAME IND-Anon-CPA

```

1:  $\text{pp} \leftarrow \text{mSetup}(1^\kappa)$ 
2: for  $i \in [N]$  do
3:    $(\text{pk}_i, \text{sk}_i) \leftarrow \text{mGen}(\text{pp})$ 
4:  $(\text{M}^*, \text{state}) \leftarrow \mathcal{A}_1(\text{pp}, (\text{pk}_i)_{i \in [N]})$ 
5:  $\text{ct}_0^* \leftarrow \text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{M}^*)$ 
6:  $\text{ct}_1^* \leftarrow \text{mEnc}(\text{pp}, N)$ 
7:  $b \leftarrow \{0, 1\}$ 
8:  $b' \leftarrow \mathcal{A}_2(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{ct}_b^*, \text{state})$ 
9: return  $[b = b']$ 

```

GAME IND-Anon-CCA

```

1:  $\text{pp} \leftarrow \text{mSetup}(1^\kappa)$ 
2: for  $i \in [N]$  do
3:    $(\text{pk}_i, \text{sk}_i) \leftarrow \text{mGen}(\text{pp})$ 
4:  $(\text{K}_0^*, \text{ct}_0^*) \leftarrow \text{mEncaps}(\text{pp}, (\text{pk}_i)_{i \in [N]})$ 
5:  $(\text{K}_1^*, \text{ct}_1^*) \leftarrow \text{mEncaps}(\text{pp}, N)$ 
6:  $b \leftarrow \{0, 1\}$ 
7:  $b' \leftarrow \mathcal{A}^{\mathcal{D}}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{ct}_b^*, \text{K}_b^*)$ 
8: return  $[b = b']$ 

```

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

```

1:  $\text{ct}_{b,i}^* := \text{mExt}(i, \text{ct}_b^*)$ 
2: if  $\text{ct} = \text{ct}_{b,i}^*$  then
3:   return  $\perp$ 
4:  $\text{K} := \text{mDecaps}(\text{sk}_i, \text{ct})$ 
5: return  $\text{K}$ 

```

Fig. 9: IND-Anon-CPA of mPKE and IND-Anon-CCA of mKEM.

B Omitted Proofs from Generic Construction: Sec. 4

B.1 Omitted Proof of Lem. 4

The following is the lemma required to complete the proof of our classical generic construction (Thm. 1).

Lemma 4. *We have $|\Pr[\text{E}_6] - \Pr[\text{E}_7]| \leq 2 \cdot \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + \frac{(q_G + q_H)}{|\mathcal{M}|}$ for some classical PPT adversary \mathcal{B}_{IND} with a runtime about the same as that of \mathcal{A} .*

Proof. Since the two games are identical unless Query occurs, we have $|\Pr[\text{E}_6] - \Pr[\text{E}_7]| \leq \Pr[\text{Query}]$. Therefore, in the following, we upper bound $\Pr[\text{Query}]$. Let us construct an IND-CPA adversary $\mathcal{B}_{\text{IND}} = (\mathcal{B}_{\text{IND}1}, \mathcal{B}_{\text{IND}2})$ which runs \mathcal{A} as a subroutine: On input $(\text{pp}, (\text{pk}_i)_{i \in [N]})$, $\mathcal{B}_{\text{IND}1}$ samples M_0^* and M_1^* uniformly random over \mathcal{M} and outputs $(\text{M}_0^*, \text{M}_1^*, \text{state} := (\text{M}_0^*, \text{M}_1^*))$. $\mathcal{B}_{\text{IND}2}$ receives $\text{ct}^* \leftarrow \text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{M}_b^*)$ and runs \mathcal{A} on input $(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{ct}^*)$ and answers its RO queries by using its RO provided by the IND-CPA game. $\mathcal{B}_{\text{IND}2}$ outputs $b' := 0$, if M_0^* is queried to $\text{H}(\cdot)$, $\text{G}_1(\cdot)$, or $\text{G}_2(\star, \cdot)$ and M_1^* is not; outputs $b' := 1$, if M_1^* is queried to $\text{H}(\cdot)$, $\text{G}_1(\cdot)$, or $\text{G}_2(\star, \cdot)$ and M_0^* is not; and a random b' otherwise. Here, the runtime of \mathcal{B}_{IND} is about the same as \mathcal{A} .

Let us denote by BAD the event that \mathcal{A} queries M_{1-b}^* to $\text{H}(\cdot)$, $\text{G}_1(\cdot)$, or $\text{G}_2(\star, \cdot)$. Since M_{1-b}^* is completely hidden from \mathcal{A} , we have $\Pr[\text{BAD}] \leq (q_G + q_H)/|\mathcal{M}|$. Then, we have:

$$\begin{aligned}
& \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) = |\Pr[b' = b] - 1/2| \\
& = \left| \Pr[\text{Query}] \cdot (\Pr[b' = b | \text{Query} \wedge \text{BAD}] \Pr[\text{BAD}] + \Pr[b' = b | \text{Query} \wedge \neg \text{BAD}] \Pr[\neg \text{BAD}]) \right|
\end{aligned}$$

$$\begin{aligned}
& + \Pr[\neg \text{Query}] \left(\Pr[b' = b | \neg \text{Query} \wedge \text{BAD}] \Pr[\text{BAD}] + \Pr[b' = b | \neg \text{Query} \wedge \neg \text{BAD}] \Pr[\neg \text{BAD}] \right) - 1/2 \Big| \\
& = \left| \Pr[\text{Query}] \left(\frac{1}{2} \cdot \Pr[\text{BAD}] + \Pr[\neg \text{BAD}] \right) + \Pr[\neg \text{Query}] \left(\frac{1}{2} \cdot \Pr[\neg \text{BAD}] \right) - 1/2 \right| \\
& = \frac{1}{2} \cdot |\Pr[\text{Query}] + \Pr[\text{BAD}]| \geq \frac{1}{2} \cdot (\Pr[\text{Query}] - \Pr[\text{BAD}]),
\end{aligned}$$

where we used the fact that events **Query** and **BAD** occur independently. Therefore, we have

$$\Pr[\text{Query}] \leq 2 \cdot \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + \Pr[\text{BAD}] \leq 2 \cdot \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + \frac{(q_G + q_H)}{|\mathcal{M}|}.$$

□

□

B.2 Omitted Proof of Thm. 2

The following is the proof of security for our generic construction in the QROM.

Proof (Proof of Thm. 2). Let \mathcal{A} be a quantum PT adversary against the IND-CCA security of **mKEM**. We upper bound its advantage by considering the following game sequence. We denote E_i as the event \mathcal{A} wins in **Game_i**.

- **Game₁**: This is the real IND-CCA security game. In particular, $\Pr[E_1] := \text{Adv}_{\text{mKEM}, N}^{\text{IND-CCA}}(\mathcal{A})$.

- **Game₂**: In this game, we replace the computation of $H'(\text{seed}_i, \cdot)$ by a random function $\hat{H}'_i(\cdot)$ in case $M = \perp$ or $\text{ct} \neq (\text{ct}_0, \hat{\text{ct}})$ occurs when answering the decapsulation oracle with input $i \in [N]$. Here, $\hat{H}'_i(\cdot)$ is a random function that cannot be accessed by the adversary. Due to Lem. 6, and a standard hybrid argument, we have

$$|\Pr[E_1] - \Pr[E_2]| \leq q'_H \cdot N \cdot 2^{-\frac{\ell+1}{2}}.$$

Using Compressed Oracles. From here on, we consider the oracles G_1 and G_2 are simulated using a single oracle \hat{G} (by using appropriate domain separation) and further assume that \hat{G} is implemented by a compressed standard oracle **CStO** initialized by the empty database D [53] (see App. D.1). Without loss of generality, we assume \mathcal{A} is provided two compressed standard oracles **CStO₁** and **CStO₂** initialized by the empty database D_1 and D_2 for better readability. Note that D_1 (resp. D_2) will contain tuples of the form (M, r_0) (resp. $((pk, M), r)$).

- **Game₃**: In this game, we add an additional check at the end of the game to see if a “bad” randomness was ever used. For fixed $\text{aux} := (\text{pp}, (pk_i, sk_i)_{i \in [N]})$, define the set of “bad” randomness as

$$\mathcal{R}_N^{\text{bad}}(\text{aux}, M) := \left\{ (r_0, r_1, \dots, r_N \in \mathcal{R}_N) \mid \begin{array}{l} \exists j \in [N] \text{ s.t. } M \neq \text{mDec}(sk_j^p, \text{mExt}(j, \text{ct})), \\ \text{where } \text{ct}_0 := \text{mEnc}^i(\text{pp}; r_0), \\ \forall i \in [N], \hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, pk_i, M; r_0, r_i), \\ \text{ct} := (\text{ct}_0, \hat{\text{ct}}_1, \dots, \hat{\text{ct}}_N). \end{array} \right\}$$

We further define $\mathcal{R}_{N,1}^{\text{bad}}(\text{aux}, \mathbf{M}) := \{r_0 \mid \exists(r_0, r_1, \dots, r_N) \in \mathcal{R}_N^{\text{bad}}(\text{aux}, \mathbf{M})\}$ and $\mathcal{R}_{N,2}^{\text{bad}}(\text{aux}, \mathbf{M}) := \{r \mid \exists i \in [N], \exists(r_0, r_1, \dots, r_{i-1}, r, r_{i+1}, \dots, r_N) \in \mathcal{R}_N^{\text{bad}}(\text{aux}, \mathbf{M})\}$. Also, denote S to be the set of messages (excluding \perp) that was obtained by decrypting the ciphertext when answering the decapsulation query. Then, after \mathcal{A} outputs its guess at the end of the game, we check the databases D_1 and D_2 if an input of the form (\mathbf{M}, r_0) where $\mathbf{M} \in S$ and $r_0 \in \mathcal{R}_{N,1}^{\text{bad}}(\text{aux}, \mathbf{M})$, or an input of the form $((\mathbf{M}, \text{pk}_i), r)$ for any $i \in [N]$ where $\mathbf{M} \in S$ and $r \in \mathcal{R}_{N,2}^{\text{bad}}(\text{aux}, \mathbf{M})$ are found, respectively. We call the event BAD_{rand} if such tuple is found and change \mathcal{A} 's output to be a random bit. Otherwise, it is defined exactly the same as in the previous game. Since the two games are identical unless BAD_{rand} occurs, we have $|\Pr[\mathbf{E}_2] - \Pr[\mathbf{E}_3]| \leq \Pr[\text{BAD}_{\text{rand}}]$.

We upper bound $\Pr[\text{BAD}_{\text{rand}}]$. When answering the decapsulation query, we query CStO_1 and CStO_2 on \mathbf{M} and measure the output to proceed with the check $\text{ct} \neq (\text{ct}_0, \hat{\text{ct}})$. Note that \mathcal{A} only makes classical queries to the decapsulation oracle. Consider the case \mathbf{M} and $(\mathbf{M}, \text{pk}_i)_{i \in [N]}$ was absent in the databases D_1 and D_2 when the decapsulation oracle was queried. Then the joint output will be distributed uniformly over \mathcal{R}_N . Therefore, the probability of sampling a bad randomness is $|\mathcal{R}_N^{\text{bad}}(\text{aux}, \mathbf{M})| / |\mathcal{R}_N|$. By definition, we have $\delta_N \geq \mathbb{E}[\max_{\mathbf{M} \in \mathcal{M}} |\mathcal{R}_N^{\text{bad}}(\text{aux}, \mathbf{M})| / |\mathcal{R}_N|]$, where the expectation is taken over the randomness used to sample $\text{aux} = (\text{pp}, (\text{pk}_i, \text{sk}_i)_{i \in [N]})$. (Here, recall δ_N is the correctness parameter of mPKE .) On the other hand, consider the case either \mathbf{M} or $(\mathbf{M}, \text{pk}_i)_{i \in [N]}$ was included in the databases D_1 or D_2 . Then, we can consider \mathcal{A} and the challenger as an algorithm that queries the compressed oracle CStO and finds an input which maps to an element in the bad randomness space. By Lem. 7, we can bound this probability by $(q_G + q_D + 1)^2 \cdot |\mathcal{R}_N^{\text{bad}}(\text{aux}, \mathbf{M})| / |\mathcal{R}_N|$. As before, this can be further upper bound by $(q_G + q_D + 1)^2 \cdot \delta_N$ by taking the expectation.

Therefore, we conclude that $\Pr[\text{BAD}_{\text{rand}}]$ occurs with probability at most $2(q_G + q_D + 1)^2 \cdot \delta_N$. Hence,

$$|\Pr[\mathbf{E}_2] - \Pr[\mathbf{E}_3]| \leq 2(q_G + q_D + 1)^2 \cdot \delta_N.$$

(The next Game₄ and Game₈ aim to get rid of the secret keys sk_i to answer \mathcal{A} 's decapsulation oracle queries.)

- **Game₄**: In this game, we include a conceptual change in the way we answer the decapsulation oracle query. During responding to the decapsulation oracle query, if $\mathbf{M} \neq \perp$, we perform a test in superposition to check if whether \mathbf{M} and $(\text{pk}_i, \mathbf{M})_{i \in [N]}$ are in the databases D_1 or D_2 . Note that this test can be performed in the standard basis. To be precise, we first prepare an extra registry (which we call b). We then evaluate a classical binary output function on D_1 and D_2 (in superposition) which evaluates to 1 if and only if D_1 includes \mathbf{M} and D_2 includes $(\text{pk}_i, \mathbf{M})_{i \in [N]}$, and finally writes the output into the newly created b register. Let us call this test implemented by a unitary operation by TEST_1 . We overload the notation and also use TEST_1 to denote the unitary operation itself.

We then, subsequently un-compute TEST_1 , which is equivalent to performing TEST_1 again. Since we compute the test and immediately un-compute the test, this modification incurs no change compared to Game_3 . Hence,

$$\Pr[\mathbf{E}_3] = \Pr[\mathbf{E}_4].$$

- Game_5 : In this game, we alter the time on which we un-compute TEST_1 . Namely, we perform TEST_1 after we finish the check $\text{ct} \neq (\text{ct}_0, \widehat{\text{ct}})$.

To make the statement more clear, we first recall how the latter check is implemented by unitary operations. First note that checking $\text{ct} \neq (\text{ct}_0, \widehat{\text{ct}})$ is equivalent to checking $(G_1(M), G_2(\text{pk}_1, M), \dots, G_2(\text{pk}_N, M)) \in \mathcal{R}_N(\text{aux}, \text{ct}, M)$, where $\mathcal{R}_N(\text{aux}, \text{ct}, M)$ is the subset of \mathcal{R}_N which maps an encryption of M to ct . Due to γ -spreadness, we have $2^{-\gamma} \geq \mathbb{E}[\max_{\text{ct} \in \mathcal{C}, M \in \mathcal{M}} |\mathcal{R}_N(\text{aux}, \text{ct}, M)| / |\mathcal{R}_N|]$, where the expectation is taken over the randomness of pp and (pk, sk) . Namely, we check (M, r_0) and $((\text{pk}_i, M), r_i)_{i \in [N]}$ such that $(r_0, r_1, \dots, r_N) \in \mathcal{R}_N(\text{aux}, \text{ct}, M)$ are included in the database D_1 and D_2 , respectively. Let the classical function which performs the above check as TEST_2 . Then, by the definition of compressed standard oracles, we first perform $\text{StdDecomp} |M\rangle |D_1\rangle |D_2\rangle$ (where we ignore the irrelevant registers right now), then apply TEST_2 that writes the output to a newly prepared registry which we call w . Conditioned on the w registry being $|0\rangle$, we compute $H'(\text{ct})$ and otherwise compute $H(M)$. Finally, we perform $\text{StdDecomp} |M\rangle |D_1\rangle |D_2\rangle$ again.

To show that Game_4 and Game_5 are indistinguishable, we must check show that performing $\text{StdDecomp} \circ \text{TEST}_2 \circ \text{StdDecomp} \circ \text{TEST}_1$ is indistinguishable from performing $\text{TEST}_1 \circ \text{StdDecomp} \circ \text{TEST}_2 \circ \text{StdDecomp}$. Here, note that we ignore the computation of H' and H since they commute with TEST_1 by noticing that they work on different registries. Notice that in case M or $(\text{pk}_i, M)_{i \in [N]}$ are already in the databases D_1 and D_2 , the corresponding output registry cannot be \perp (e.g., (M, \perp) is not in the database D_1), since each query to CStO ends by removing such entries by definition. Therefore, again by definition, StdDecomp is the same as an identity operation for such inputs. Now, in case M or $(\text{pk}_i, M)_{i \in [N]}$ are not in the databases D_1 and D_2 , StdDecomp adds $\frac{1}{\sqrt{2^{|r_0|}}} \sum_{r_0} |M, r_0\rangle$ or $\frac{1}{\sqrt{2^{|r_i|}}} \sum_{r_i} |M, r_i\rangle$ to the databases D_1 and D_2 , respectively. Denote TEST_1^\dagger as the unitary operation that first applies the quantum Fourier transform to the registry corresponding to (r_0, r_1, \dots, r_N) and then outputs 1 if and only if all the corresponding registry is not 0. Then, from the above arguments, we have $\text{TEST}_1^\dagger \circ \text{StdDecomp} = \text{StdDecomp} \circ \text{TEST}_1$. Since TEST_1^\dagger tests whether (r_0, r_1, \dots, r_N) is equal to $(0, \dots, 0)$ in the Fourier basis and TEST_2 tests whether $(r_0, r_1, \dots, r_N) \in \mathcal{R}_N(\text{aux}, \text{ct}, M)$ in the standard basis, we can invoke Lem. 8 to argue that TEST_1^\dagger and TEST_2 “almost” commute. Namely, performing $\text{TEST}_2 \circ \text{TEST}_1^\dagger$ or $\text{TEST}_1^\dagger \circ \text{TEST}_2$ alters the distribution of the view of \mathcal{A} by at most $8\sqrt{|\mathcal{R}_N(\text{aux}, \text{ct}, M)| / |\mathcal{R}_N|}$. Finally, using the same argument for swapping TEST_1^\dagger and StdDecomp as before, we conclude that performing $\text{StdDecomp} \circ \text{TEST}_2 \circ \text{StdDecomp} \circ \text{TEST}_1$ is indistinguishable from performing $\text{TEST}_1 \circ \text{StdDecomp} \circ \text{TEST}_2 \circ \text{StdDecomp}$. In particular, by taking the expec-

tation taken over the randomness used to sample $\mathbf{aux} = (\mathbf{pp}, (\mathbf{pk}_i, \mathbf{sk}_i)_{i \in [N]})$, we conclude

$$|\Pr[\mathbf{E}_4] - \Pr[\mathbf{E}_5]| \leq q_{\mathcal{D}} \cdot 8\sqrt{2^{-\gamma}}.$$

- **Game₆**: In this game, after we perform (the first) TEST_1 , we perform an additional computation which takes as input the bit b , which is the output of TEST_1 . Recall $b = 1$, if \mathbf{M} and $(\mathbf{pk}_i, \mathbf{M})_{i \in [N]}$ are in the databases D_1 and D_2 , respectively, and $b = 0$, otherwise. Now, in case $b = 0$, we simply compute $\mathbf{H}'(\mathbf{ct})$ and return it to the adversary, ignoring the rest of the checks. Otherwise, it is the same as in **Game₅**. The only way for \mathcal{A} to distinguish the two games is it queries the decapsulation oracle on $(i, \mathbf{ct} \neq \mathbf{ct}_i^*)$ such that $\mathbf{ct} = (\mathbf{ct}_0, \hat{\mathbf{ct}}_i)$ where $\mathbf{M} := \text{mDec}(\mathbf{sk}_i^p, \mathbf{ct})$ or $(\mathbf{pk}_i, \mathbf{M})_{i \in [N]}$ are not in the databases. Since, when \mathbf{M} are not included in the databases, the corresponding randomness are in uniform distribution, due to γ -spreadness, we have

$$|\Pr[\mathbf{E}_5] - \Pr[\mathbf{E}_6]| \leq q_{\mathcal{D}} \cdot 2^{-\gamma}.$$

- **Game₇**: In this game, we further modify the way we answer the decapsulation oracle query. Recall in the previous game, we computed $\mathbf{M} \leftarrow \text{mDec}(\mathbf{sk}_i^p, \mathbf{ct})$ and tested several times whether \mathbf{M} is in the databases or not. In this game, we instead scan through all messages \mathbf{M} inside the databases such that \mathbf{M} is inside D_1 and $(\mathbf{pk}_i, \mathbf{M})$ is inside D_2 for all $i \in [N]$. It then checks if $\mathbf{ct} = (\mathbf{ct}_0, \hat{\mathbf{ct}}_i)$, where $\mathbf{ct}_0, \hat{\mathbf{ct}}_i$ are encrypted using the measured randomness, and returns $\mathbf{H}(\mathbf{M})$ if it holds and $\mathbf{H}'(\mathbf{ct})$ otherwise.

We check that the two games are identical unless event BAD_{rand} occurs. First, notice that if the decapsulation oracle in **Game₆** outputs $\mathbf{K} := \mathbf{H}(\mathbf{M})$, then \mathbf{M} was in the databases and $\mathbf{ct} = (\mathbf{ct}_0, \hat{\mathbf{ct}}_i)$ holds. Therefore, the decapsulation oracle in **Game₇** outputs the same \mathbf{K} as well. On the other hand, assume the decapsulation oracle in **Game₇** outputs $\mathbf{K} := \mathbf{H}(\mathbf{M})$ for some \mathbf{M} in the dabases such that $\mathbf{ct} = (\mathbf{ct}_0, \hat{\mathbf{ct}}_i)$ where $\mathbf{ct}_0 := \text{mEnc}^i(\mathbf{pp}; \mathbf{G}_1(\mathbf{M}))$ and $\hat{\mathbf{ct}}_i := \text{mEnc}^d(\mathbf{pp}, \mathbf{pk}_i, \mathbf{M}; \mathbf{G}_1(\mathbf{M}), \mathbf{G}_2(\mathbf{pk}_i, \mathbf{M}))$. Then, since we have no correctness error conditioning on BAD_{rand} not occurring, \mathbf{ct} must decrypt to \mathbf{M} . Hence, this implies that the decapsulation oracle **Game₆** outputs the same \mathbf{K} as well. Finally, since when BAD_{rand} occurs, we force the adversary to output a random bit in both games, we have

$$\Pr[\mathbf{E}_6] = \Pr[\mathbf{E}_7].$$

- **Game₈**: In this game, we undo the change we made in **Game₃** and will no longer abort in case event BAD_{rand} occurs. Due to the same argument as before, we have

$$|\Pr[\mathbf{E}_7] - \Pr[\mathbf{E}_8]| \leq 2(q_{\mathcal{G}} + 1)^2 \cdot \delta_N.$$

At this point, the challenger no longer requires the secret keys \mathbf{sk}_i .

(The next final **Game₉** aims to get rid of \mathbf{M}^* in the challenge ciphertext.)

- **Game₉**: In this game, we sample the random message $\mathbf{M}^* \leftarrow \mathcal{M}$ to be used to generate the challenge ciphertext at the beginning. We then modify the compressed oracle (where we assume that \mathbf{H} is also modeled by a compressed

oracle now) so that when \mathcal{A} makes a query, we measure it's input register to see if M^* is included. Namely, we replace CStO with $\text{CStO} \setminus S$, where $S = \{M^*, (\text{pk}_i, M^*)_{i \in [N]}\}$. Note that for simplicity, we handle CStO_1 and CStO_2 by one compressed oracle and assume that the domains are separated. In case, we ever measure an input in the set S , we abort the game and replace \mathcal{A} 's output by a random bit. We call this event **FIND**. Then, by Thm. 3, we have

$$|\Pr[E_8] - \Pr[E_9]| \leq \sqrt{q_G \Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z_0)]} + \sqrt{q_G \Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z_1)]}, \quad (9)$$

where $z_0 := (\text{pp}, (\text{pk}_i, \text{sk}_i)_{i \in [N]}, \text{ct}^*, K_0^* = H(M^*))$ and z_1 is the same as z_0 except that $K_1^* \leftarrow \mathcal{K}$ is provided rather than K_0^* . Note that $\text{ct}^* := \text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}; G_1(M^*), G_2(\text{pk}_1, M^*), \dots, G_2(\text{pk}_N, M^*))$. Here, we omit the oracle access \mathcal{A} has to H for simplicity.

Before, bounding the above probability in Eq. (9), we mention that in Game_9 , the adversary has no winning advantage. This is because, in case event **FIND** does not occur, then \mathcal{A} has no information on M^* , hence, $K_0^* = H(M^*)$ is distributed uniformly to \mathcal{A} . Moreover, in case event **FIND** occurs, we force \mathcal{A} to output a random bit. Therefore, we have $\Pr[\text{Game}_9] = 1/2$.

It remains to bound the probability in Eq. (9). Denote $p_b := \Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z_b)]$ for $b \in \{0, 1\}$. We first focus on p_0 . We consider two cases, let **Case₁** (resp. **Case₂**) be the case that **FIND** occurs after (resp. before) \mathcal{A} receives the challenge ciphertext. In **Case₁**, since M^* is never queried by \mathcal{A} before it receives the challenge ciphertext, the output of $G(M^*)$, $G(\text{pk}_i, M^*)$ or $H(M^*)$ for $i \in [N]$ are completely hidden on time of preparing the challenge ciphertext. Therefore, we can replace z_0 by z'_0 where we replace ct^* in z_0 to be $\text{ct}^{'*} \leftarrow \text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}; r_0, r_1, \dots, r_N)$ for uniformly random (r_0, r_1, \dots, r_N) and $K_0^* \leftarrow \mathcal{K}$. Then, we can invoke the IND-CPA security of the underlying mPKE. Namely, there exists an adversary \mathcal{B}_{IND} with runtime about that of \mathcal{A} such that

$$\left| \Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z'_0)] - \Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z''_0)] \right| \leq \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}),$$

where $z''_0 := (\text{pp}, (\text{pk}_i, \text{sk}_i)_{i \in [N]}, \text{ct}^{''*}, K_0^* \leftarrow \mathcal{K})$ and $\text{ct}^{''*}$ is a random encryption to the message 0. By Thm. 3, we have $\Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z''_0)] \leq 4q_G \cdot |S| / |\mathcal{M}|$ since z''_0 is now independent of S . On the other hand, when **Case₂** occurs, it means \mathcal{A} queried M^* without receiving (ct^*, K_0^*) . Therefore, we can use Thm. 3 since it's input is independent of M^* . Therefore, combining the two cases, we have

$$p_0 \leq \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + 8q_G \cdot |S| / |\mathcal{M}|.$$

We can use the same argument for p_1 . Therefore, combining everything together, we can upper bound Eq. (9) by

$$|\Pr[E_8] - \Pr[E_9]| \leq 2 \cdot \sqrt{q_G \text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{B}_{\text{IND}}) + 8q_G^2 \cdot |S| / |\mathcal{M}|},$$

where $|S| = N + 1$. Collecting all the bounds, we obtain the bound in the statement. This completes the proof. \square

C Omitted Details from PQ Instantiation of mKEM: Sec. 5

C.1 Omitted Proofs from Lattice Part: Sec. 5.1

The following is the proof for Lem. 1.

Proof (Proof of Lem. 1). Let \mathcal{A} be an efficient (classical or quantum) adversary against the IND-CPA security of mPKE. We upper bound its advantage by considering the following game sequence. We denote E_i as the event \mathcal{A} wins in Game_i .

- **Game₁**: This is the real IND-CPA security game. In particular, $\text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{A}) := |\Pr[E_1] - 1/2|$.

- **Game₂**: In this game, we change how the public keys $(\text{pk}_i)_{i \in [N]}$ are created. Rather than generating each \mathbf{B}_i as $\mathbf{A}\mathbf{S}_i + \mathbf{E}_i$ for $i \in [N]$, we simply sample a random $\mathbf{B} \leftarrow R_q^{n \times m}$. It is easy to see that this modification is indistinguishable assuming the $\text{LWE}_{n, n, Nm}$ assumption. Hence, we can construct an adversary \mathcal{B}_1 with around the same running time as \mathcal{A} such that

$$|\Pr[E_1] - \Pr[E_2]| \leq \text{Adv}_{n, n, Nm}^{\text{LWE}}(\mathcal{B}_1).$$

- **Game₃**: In this game, we change how the challenge ciphertext ct^* is created. Namely, rather than sampling $\text{ct}^* := (\mathbf{U}, (\mathbf{V}_i)_{i \in [N]})$ as valid LWE samples, we simply sample a random $(\mathbf{U}, (\mathbf{V}_i)_{i \in [N]}) \leftarrow R_q^{\bar{m} \times n} \times (R_q^{\bar{m} \times m})^N$. Similarly to above, this reduces directly to the $\text{LWE}_{(n+Nm), n, \bar{m}}$, where note that we take the transpose of the LWE sample since the secret matrix (i.e., $\mathbf{R} \leftarrow D_s^{\bar{m} \times n}$) is on the left-hand now. Hence, we can construct an adversary \mathcal{B}_2 with around the same running time as \mathcal{A} such that

$$|\Pr[E_2] - \Pr[E_3]| \leq \text{Adv}_{(n+Nm), n, \bar{m}}^{\text{LWE}}(\mathcal{B}_2).$$

Finally, in **Game₃**, the challenge ciphertext is distributed uniformly random and independently from the challenge bit b . Therefore, no adversary can have a distinguishing advantage. Hence, $\Pr[E_3] = 1/2$. Combining everything together, we get the desired bound. \square

C.2 Omitted Proofs from Isogeny Part: Sec. 5.2

The following is the proof for Lem. 2.

Proof (Proof of Lem. 2). Let \mathcal{A} be an efficient (classical or quantum) adversary against the IND-CPA security of mPKE. We upper bound its advantage by considering the following game sequence. We denote by E_i the event \mathcal{A} wins in Game_i .

- **Game₀**: This is the real IND-CPA security game. In particular, $\text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{A}) := |\Pr[E_0] - 1/2|$.

- **Game_{1,j}** for $j \in [0, N]$: In this game, we modify the challenger so that it sets $\widehat{\mathbf{ct}}_i^* := \text{Enc}^d(\mathbf{pp}, \mathbf{pk}_i, M^*; r_0)$ for $i \in [N - j]$ and $\widehat{\mathbf{ct}}_i^* \leftarrow H(J'_i) \oplus \text{Encode}(M)$ for $i \in [j]$. Here, J'_i denotes the j -invariant of the elliptic curve $E / \langle P_2 + [s']Q_2, P_3 + [r']Q_3 \rangle$, where s' and r' are sampled uniformly from $\mathbb{Z}_{2^{e_2}}$ and $\mathbb{Z}_{3^{e_3}}$, respectively. The challenger outputs $\mathbf{ct}^* := (\mathbf{ct}_0^*, \widehat{\mathbf{ct}}_1^*, \dots, \widehat{\mathbf{ct}}_N^*)$ as the challenge ciphertext. We note that **Game_{1,0}** corresponds to **Game₀**. Each game is indistinguishable from the previous one under the $\text{SSDDH}_{p,E,B_2,B_3}$ assumption. In particular, we can construct an adversary \mathcal{B} with around the same running time as \mathcal{A} such that

$$|\Pr[\mathbf{E}_{1,j}] - \Pr[\mathbf{E}_{1,j+1}]| \leq \text{Adv}_{p,E,B_2,B_3}^{\text{SSDDH}}(\mathcal{B})$$

for every j in $[0, N - 1]$.

- **Game₂**: In this game, the challenger is modified so that it sets $\widehat{\mathbf{ct}}_i^* \leftarrow h'_i \oplus \text{Encode}(M)$ for all $i \in [N]$, where h'_i is an element sampled uniformly from $\{0, 1\}^w$. This game is indistinguishable from the previous game due to the Leftover Hash Lemma (Lem. 3). In particular, let X be the distribution on $W \subset \mathcal{X}$ induced by uniformly sampling the pair $(s', r') \in \mathbb{Z}_{2^{e_2}} \times \mathbb{Z}_{3^{e_3}}$ and setting X to be the j -invariant of the elliptic curve $E / \langle P_2 + [s']Q_2, P_3 + [r']Q_3 \rangle$. It is known that such distribution approximate the uniform one [22], and hence $\mathbf{H}_\infty(X) \approx \log_2 p$. Consequently, invoking the Leftover Hash Lemma N times, we have

$$|\Pr[\mathbf{E}_{1,N}] - \Pr[\mathbf{E}_2]| \leq \frac{N}{2} \cdot \sqrt{2^w/p}.$$

Finally, since the challenge ciphertexts are distributed uniformly random for both challenge bits, 0 and 1, we have $\Pr[\mathbf{E}_2] = 1/2$. This concludes the proof. \square

C.3 Omitted Details on CSIDH-based mPKE Sec. 5.2

The mPKE deduced from CSIDH is provided in Fig. 10. In **mSetup** a prime p of the form $4\ell_1\ell_2 \cdots \ell_t - 1$, where ℓ_1, \dots, ℓ_t are small odd primes, is chosen. Then the public parameter \mathbf{pp} output by the algorithm consists of: a generator \mathbf{g} of the cyclic ideal class group $\mathcal{Cl}(\mathcal{O})$, where $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$, and its order $n \approx \sqrt{p}$; a supersingular elliptic curve E over \mathbb{F}_p uniformly sampled from $\mathcal{Ell}_p(\mathcal{O}, \pi)$; a hash-function \mathbf{H} uniformly sampled from a family of universal hash functions $\mathcal{H} = \{\mathbf{H}_k : \mathcal{Ell}_p(\mathcal{O}, \pi) \rightarrow \{0, 1\}^w\}_{k \in K}$ indexed by the finite set K . Furthermore, **Encode** (resp. **Decode**) is an efficiently computable bijective function that maps elements from the message space (resp. $\{0, 1\}^w$) to $\{0, 1\}^w$ (resp. message space). The details of **Encode** and **Decode** are not relevant for this section.

The CSIDH-based mPKE satisfies all the properties required by the “multi-recipient” Fujisaki-Okamoto transform. It is easy to see that the scheme is perfectly correct, hence $\delta = 0$. Furthermore, the output distribution of **mEncⁱ** coincides with the uniform distribution over $\mathcal{Ell}_p(\mathcal{O}, \pi)$ (it is induced by the free and

Algorithm 13 $\text{mSetup}(1^\kappa)$

Input: Security parameter 1^κ
Output: Public parameter pp
1: Select p and \mathbf{g}
2: $n \leftarrow \text{Order}(\mathbf{g}), \mathbf{H} \leftarrow \mathcal{H}$
3: $E \leftarrow \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$
4: **return** $\text{pp} := (\mathbf{g}, n, E, \mathbf{H})$

Algorithm 14 $\text{mGen}(\text{pp})$

Input: Public parameter $\text{pp} = (\mathbf{g}, n, E, \mathbf{H})$
Output: Public key pk , secret key sk
1: $a \leftarrow \mathbb{Z}_n$
2: $H \leftarrow \mathbf{g}^a \star E$
3: **return** $\text{pk} := H, \text{sk} := a$

Algorithm 15 $\text{mEnc}(\text{pp}, (\text{pk}_i)_{i \in [N]}, \text{M})$

Input: Public parameter $\text{pp} = (\mathbf{g}, n, E, \mathbf{H})$,
set of public keys $(\text{pk}_i = H_i)_{i \in [N]}$, mes-
sage M
Output: Ciphertext $\text{ct} = (\text{ct}_0, (\hat{\text{ct}}_i)_{i \in [N]})$
1: $r_0 := r \leftarrow \mathbb{Z}_n$
2: $\text{ct}_0 := \text{mEnc}^l(\text{pp}; r_0)$
3: **for** $i \in [N]$ **do**
4: $\hat{\text{ct}}_i := \text{mEnc}^d(\text{pp}, \text{pk}_i, \text{M}; r_0)$
5: **return** $\text{ct} := (\text{ct}_0, \hat{\text{ct}}_1, \dots, \hat{\text{ct}}_N)$

Algorithm 16 $\text{mEnc}^d(\text{pp}, \text{pk}_i, \text{M}; r_0)$

Input: Public parameter $\text{pp} = (\mathbf{g}, n, E, \mathbf{H})$, public key $\text{pk}_i = H_i$,
message M , randomness $r_0 = r$
Output: (Public key dependent) cipher-
text $\hat{\text{ct}}_i$
1: $H'_i \leftarrow \mathbf{g}^r \star H_i$
2: $F_i \leftarrow \mathbf{H}(H'_i) \oplus \text{Encode}(\text{M})$
3: **return** $\hat{\text{ct}}_i := F_i$

Algorithm 17 $\text{mEnc}^i(\text{pp}; r_0)$

Input: Public parameter $\text{pp} = (\mathbf{g}, n, E, \mathbf{H})$,
randomness $r_0 = r$
Output: (Public key independent) cipher-
text ct_0
1: $E' \leftarrow \mathbf{g}^r \star E$
2: **return** $\text{ct}_0 := E'$

Algorithm 18 $\text{mDec}(\text{sk}, \text{ct})$

Input: Public parameter $\text{pp} = (\mathbf{g}, n, E, \mathbf{H})$, Secret key $\text{sk} = a$,
ciphertext $\text{ct} = (E', F)$
Output: Message M
1: $H' \leftarrow \mathbf{g}^a \star E'$
2: $\text{M} \leftarrow F \oplus \mathbf{H}(H')$
3: **return** $\text{M} := \text{Decode}(\text{M})$

Fig. 10: CSIDH-based mPKE via hashed ElGamal.mExt with input index i is defined in the obvious way by picking the relevant components $(\text{ct}_0, \hat{\text{ct}}_i)$ from ct . Note that mEnc^d does not require any randomness r_i for $i \in [N]$ (except that used to compute the action \star).

transitive group action \star). Therefore $\gamma(\text{pp}, \text{pk}) \geq (\log_2 p)/2$ for all $\text{pp} \in \text{Setup}(1^\kappa)$ and $(\text{pk}, \text{sk}) \in \text{Gen}(\text{pp})$, and hence $\gamma = (\log_2 p)/2$.

IND-CPA Security. Analogously to the SIDH-based mPKE, the IND-CPA security of the scheme in Fig. 10 follows from the dCSIDH assumption (Def. 12) and the Leftover Hash Lemma (App. A.2). In particular we have:

Lemma 5. *Assume mPKE as shown in Fig. 10. Then, for any (classical or quantum) IND-CPA adversary \mathcal{A} , there exist (classical or quantum) adversary \mathcal{B}*

such that

$$\text{Adv}_{\text{mPKE}, N}^{\text{IND-CPA}}(\mathcal{A}) \leq N \cdot \left(\text{Adv}_{p, g}^{\text{dCSIDH}}(\mathcal{B}) + (1/2)\sqrt{2^w/\sqrt{p}} \right). \quad (10)$$

The proof of Lem. 5 is just an adaptation of that of Lem. 2. To be precise, a sequence of hybrid games $E_{1,j}$, with $j \in [0, N]$, replaces H'_i with a uniformly random curve $\tilde{H}'_i \in \mathcal{E}\ell_p(\mathcal{O}, \pi)$, and so $|\Pr[E_{1,j}] - \Pr[E_{1,j+1}]| \leq \text{Adv}_{p, g}^{\text{dCSIDH}}(\mathcal{B}_1)$ for every j in $[0, N-1]$. At this point, we can invoke the Leftover Hash Lemma to argue that the challenge ciphertext is distributed uniformly random. The value of w can be parameterized so that in Eq. (10) the second term on the right can be safely discarded. Finally, we observe that the above-sketched proof also implies IND-Anon-CPA, with mEnc that outputs a tuple composed by a ciphertext ct_0 and N uniformly random elements in $\{0, 1\}^w$.

Remark 7 (CSIDH with ideal class group of unknown structure). The scheme described in Fig. 10 can be easily adapted to the case where the structure of the ideal class group $\mathcal{C}\ell(\mathcal{O})$ is unknown. In particular, in that case the public parameter pp comes with two distributions D_B and D_{E_0} . Then E is sampled from D_{E_0} , while the values in line 1 in the mSetup and mEnc algorithms are sampled from D_B .

D Background on Quantum Random Oracles

The main purpose of this section is to recall the *compressed oracle* techniques introduced in [53]. At a high level, it allows for powerful proof techniques used in classical random oracle model (ROM) to also be usable in the quantum ROM (QROM). For example, the compressed oracle technique allows for *lazy sampling* (i.e., the output of an entry is defined when it becomes necessary) and allows checking what the adversary has queried to the random oracle thus far. Both of which were considered to be difficult, if not impossible, before the astute observation by Zhandry. Below, we assume the readers to have basic knowledge of quantum computations and refer to [43] for a thorough introduction. Our presentation of the digest of Zhandry's result follows closely to that made in [30]. We refer [53] for the full detail and [30] for a more in-depth summary of [53] (along with an alternative formalization of compressed oracles).

Stateful Oracles. Let oracle \mathcal{O} have a k -qubit quantum state. Then, \mathcal{O} is modeled as a sequence of unitary operators $(\mathcal{O}_1, \dots, \mathcal{O}_q)$ that acts on the first $(m+n)$ -qubits of \mathcal{A} 's quantum register in addition to \mathcal{O} 's quantum register. Here, we assume the first m -registers of \mathcal{A} represents the input to \mathcal{O} and the last n -registers of \mathcal{A} are used to write the response of \mathcal{O} . When we run \mathcal{A} relative to the oracle \mathcal{O} , the unitary operation $U_0 \otimes I_k, \mathcal{O}_1, U_1 \otimes I_k, \dots, U_q \otimes I_k, \mathcal{O}_q$ act sequentially on the initial state $|0^\ell\rangle \otimes |\text{init}_{\mathcal{O}}\rangle$, where $|\text{init}_{\mathcal{O}}\rangle$ is the initial state of \mathcal{O} . Finally, \mathcal{A} measures the resulting quantum state $(U_q \otimes I_k)\mathcal{O}_q \dots \mathcal{O}_1(U_0 \otimes I_k)|0^\ell\rangle \otimes |\text{init}_{\mathcal{O}}\rangle$, and returns the measurement result as the output. We write

$b \leftarrow \mathcal{A}^\mathcal{O}()$ to denote the event that \mathcal{A} runs relative to the oracle \mathcal{O} and outputs b . We assume without loss of generality that all quantum registers including those of the oracles are measured only once at the end of the game.

D.1 Compressed Oracles and QROM

Standard Oracles. An oracle \mathcal{O}_H can be implemented with an encoding of a function $H : \{0, 1\}^m \rightarrow \{0, 1\}^n$ and an operator StO that is independent of the description of H . Here, we assume the function H is encoded into the $(n2^m)$ -qubit state $|H(0)\rangle|H(1)\rangle \cdots |H(2^m - 1)\rangle$. The unitary operator StO is defined as

$$\text{StO} : |x\rangle |y\rangle \otimes |\alpha_0\rangle \cdots |\alpha_{2^m-1}\rangle \mapsto |x\rangle |y \oplus \alpha_x\rangle \otimes |\alpha_0\rangle \cdots |\alpha_{2^m-1}\rangle,$$

where $\alpha_x \in \{0, 1\}^n$ for each $x \in [0, 2^m-1]$. In particular, we have $\text{StO} |x\rangle |y\rangle |H\rangle = |x\rangle |y \oplus H(x)\rangle |H\rangle$. Here, the x registers represent inputs to the function H , the y registers are for writing the response, and we assume that the x, y registers come from the adversary.

Zhandry [53] observed that one can formalize the QROM by allowing the adversary \mathcal{A} to run relative to StO , where H is initialized to the uniform superposition over all H , that is, $\sum_H \frac{1}{\sqrt{2^{n2^m}}} |H\rangle$. More concretely, the whole quantum state before \mathcal{A} makes the $(i+1)$ -th quantum query can be expressed by

$$|\phi_{i+1}\rangle = (U_q \otimes I_k) \text{StO} \cdots \text{StO} (U_0 \otimes I_k) \left(|0^\ell\rangle \otimes \sum_H \frac{1}{\sqrt{2^{n2^m}}} |H\rangle \right). \quad (11)$$

This is in contrast to previous formalization of QROM where one (quantumly accessible) function H was randomly sampled and fixed throughout the security game. Zhandry shows that the new notion of QROM is perfectly indistinguishable from the previous QROM from the view of the adversary \mathcal{A} .

Compressed Standard Oracles. An issue with the above is that we cannot simulate the oracle efficiently as we must store the whole function H . Therefore, in case we need to efficiently simulate the view of the adversary within a security proof, say because we want to base security on a computational assumption, then we cannot use the above as it is. Zhandry's [53] main observation is that we can consider a *compressed* version of the standard oracle which allows for efficient simulation.

To get a better intuition, consider changing the basis of the y register and α_i registers in Eq. (11) from the standard computational basis $\{|u\rangle\}_{u \in \{0,1\}^n}$ to the so called *Fourier basis* $\{H^{\otimes n} |u\rangle\}_{u \in \{0,1\}^n}$, where H is the Hadamard operator. Denoting elements represented in the Fourier basis by “ $\hat{\cdot}$ ”, Eq. (11) is equivalent to

$$|\phi_{i+1}\rangle = (U_q \otimes I_k) \text{StO} \cdots \text{StO} (U_0 \otimes I_k) \left(|0^\ell\rangle \otimes |\widehat{0^{n2^m}}\rangle \right) \quad (12)$$

$$= \sum_{xyz\hat{D}} a'_{xyz\hat{D}} |xyz\rangle \otimes |\hat{D}\rangle, \quad (13)$$

where $a'_{xyz\widehat{D}}$ denotes a complex number satisfying $\sum_{xyz\widehat{D}} |a'_{xyz\widehat{D}}|^2 = 1$, z is \mathcal{A} 's working register, and $|\widehat{D}\rangle = |\widehat{\alpha_0}\rangle \cdots |\widehat{\alpha_{2^m-1}}\rangle$ is a concatenation of 2^m n -bit strings. Moreover, we have the following

$$\text{StO} |x\rangle |\widehat{y}\rangle \otimes |\widehat{\alpha_0}\rangle \cdots |\widehat{\alpha_{2^m-1}}\rangle = |x\rangle |\widehat{y}\rangle \otimes |\widehat{\alpha_0}\rangle \cdots |\widehat{\alpha_x \oplus \widehat{y}}\rangle \cdots |\widehat{\alpha_{2^m-1}}\rangle,$$

In particular, only one data is written in superposition when an adversary makes a query. Therefore, since $|\widehat{\alpha_x}\rangle \neq 0^n$ for at most i many x , \widehat{D} can be regarded as a *database* which stores at most i many non-zero entries. We can bring back this argument to the standard basis by performing the Hadamard operator $H^{\otimes n}$ to each $\widehat{\alpha_x}$ in \widehat{D} and obtaining another database D . Intuitively, $(x, \alpha_x) \in D$ corresponds to the condition that A queried x to the oracle and received back α_x . We call D a *standard database* or simply a database.

Now that we established that D only has at most q entries that are non-zero after \mathcal{A} queries the oracle, Zhandry's [53] main idea was to compress the description of the oracle so that we simply keep track of the database for which the entry are non-zero. In summary, Zhandry observed that the QRO can be described as a stateful quantum oracle CStO called the *compressed standard oracle*. Roughly, CStO comprises of the following procedures:

1. On input (x, α_x) , look for a tuple $(x, \alpha_x) \in D$. If such a tuple exists, return $|x\rangle |y \oplus \alpha_x\rangle$.
2. Otherwise, create a new register initialized to the state $\frac{1}{\sqrt{2^n}} \sum_{\alpha_x} |\alpha_x\rangle$. Add the registers (x, α_x) to D and respond with $|x\rangle |y \oplus \alpha_x\rangle$.
3. Finally, test whether the registers containing α_x contain 0^n in the Fourier basis. If so, remove the tuple from D . Otherwise, leave the tuple in D .

It may be helpful to understand the above by considering the classical ROM. The first and second item corresponds to *lazy sampling*. That is, if the database is defined on x , then it outputs whatever written there, and otherwise, it samples a random $\alpha_x \leftarrow \{0, 1\}^n$ element on the fly. Here, sampling a random element in QROM amounts to creating the superposition $\frac{1}{\sqrt{2^n}} \sum_{\alpha_x} |\alpha_x\rangle$. The third item is unique to the quantum setting, and is a vital procedure to keep the state of the oracle and the adversary un-entangled in case the data x is no longer used by the adversary. In particular, by appropriately *forgetting* information, we can simulate the view of \mathcal{A} without every being detected. The formal description of CStO follows.

More formally, a compressed standard oracle CStO is defined as $\text{CStO} = \text{StdDecomp} \circ \text{CStO}' \circ \text{StdDecomp}$,⁹ where StdDecomp and CStO' are unitary operations described below and the oracles's state is initialized as the empty database D . Here, D will be represented as an element of the set S^q , where $S = (\{0, 1\}^m \cup \{\perp\}) \times \{0, 1\}^n$ and we call D *empty* when $D = ((\perp, 0^n), \dots, (\perp, 0^n))$, where the number of $(\perp, 0^n)$ pairs is equal to q (i.e., the number of oracle query

⁹ Note that without loss of generality we simplify the discussion by omitting the operator *Increase* in [53].

\mathcal{A} makes). In fact, we have additional requirements for D , such as requiring D to be sorted in some canonical order, however, since we do not explicitly use them, we refer [53] for the details.

The unitary operation **StdDecomp** takes as input $|x\rangle|y\rangle|D\rangle$ and does the following:

- If $D(x) = \perp$, return $\frac{1}{\sqrt{2^n}} \sum_{\alpha_x} |D \cup (x, \alpha_x)\rangle$. This corresponds to item 2 above, where we “randomly sample” an output for x if x is yet to be specified.
- If $D(x) \neq \perp$, denote $|D\rangle = \sum_{\alpha_x} a_{\alpha_x} |D' \cup (x, \alpha_x)\rangle$ where $D'(x) = \perp$, and return $|D'\rangle$ if $a_{\alpha_x} = \frac{1}{\sqrt{2^n}}$ for all α_x and return $|D\rangle$ otherwise. In other words, if D is already specified on x and if the corresponding α_x registers contain 0 in the Fourier basis, it will remove x and the α_x registers from D which corresponds to item 3 above. Otherwise, **StdDecomp** is the identity.

The unitary operation **CStO'** takes as input $|x\rangle|y\rangle|D\rangle$ and simply returns $|x\rangle|y \oplus D(x)\rangle \otimes |D\rangle$. Here, note that **CStO'** is well-defined as it is always applied after **StdDecomp**, which guarantees that $D(x) \neq \perp$.

This completes the explanation of compressed standard oracles.

D.2 Useful Lemmas

We prepare some useful lemmas required for our proof.

General Lemmas.

Lemma 6. ([48, Lem. 2.2]) *Let ℓ be an integer. Let $H: \{0, 1\}^\ell \times \mathcal{X} \rightarrow \mathcal{Y}$ and $H': \mathcal{X} \rightarrow \mathcal{Y}$ be two independent random functions. If an unbounded time quantum adversary \mathcal{A} makes a query to H at most Q_H times, then we have*

$$\left| \Pr[\text{seed} \leftarrow \{0, 1\}^\ell : 1 \leftarrow \mathcal{A}^{[H], [H(\text{seed}, \cdot)]}(1^\lambda)] - \Pr[1 \leftarrow \mathcal{A}^{[H], [H']}](1^\lambda) \right| \leq Q_H \cdot 2^{-\frac{\ell+1}{2}}.$$

Lemma 7. *Let $S \subset X \times Y$ be an arbitrary set. For any adversary making q queries to **CStO** initialized by the empty database D and an arbitrary number of database read queries, if the database D is measured after the q queries, the probability it contains a pair of the form $(x, y) \in S$ is at most $q^2|S|/|Y|$.*

Commutativity of Quantum Tests. In classical computation, the order of how one checks $x \in S$ and $x \in T$ for sets S, T is irrelevant. However, in quantum computation, the order matters. The following provides a condition for when two unitary operations, each representing some check, “almost” commutes. That is, the condition for when we can swap the order of the checks while remaining unnoticed by an adversary.

For a set S and value x , let $(x \stackrel{?}{=} S)$ as the boolean function that outputs 1 if and only if $x \in S$. Let $C: Z \rightarrow \{0, 1\}$. A *computational basis test*, denoted CBT_C , performs the unitary defined as $|x, S, b, z\rangle \mapsto |x, S, b \oplus (C(z) \cdot (x \stackrel{?}{=} S)), z\rangle$. Namely, conditioned on $x \in S$, it writes the result of $C(z)$ in the b register. We call x the

test register, S the set register, b the output register, and z the auxiliary register. Analogously, we define a *Fourier basis test*, denoted FBT_C , which is identical to CBT_C , except that it performs the Hadamard operator $H^{\otimes n}$ on the x registers before and after the test.

Definition 16 (Almost commutativity). *Let U_0 and U_1 be unitaries over the same quantum system. We say U_0 and U_1 ϵ -commute if, for any initial state ρ , the images of ρ under U_0U_1 and U_1U_0 are at most ϵ -far in trace distance.*

Lemma 8 ([53, Lemma 39]). *Consider a quantum system over n -bit strings x , subsets $S, T \subseteq \{0, 1\}^n$ of size at most s and t , respectively, output registers $b, c \in \{0, 1\}$, and auxiliary information z . Then the following unitaries $8\sqrt{st/2^n}$ -almost commute:*

- CBT_C , where x is the test register, S is the set register, b is the output register, (c, z) is the auxiliary register.
- FBT_D , where x is the test register, T is the set register, c is the output register, (b, z) is the auxiliary register.

One-way to Hiding for Compressed Oracles. We recall the (slightly simplified) one-way to hiding (O2H) lemma for compressed oracles [21], which was originally defined for standard oracles [6, 51].

Let CStO be a compressed standard oracle on database D . For any $x \in \{0, 1\}^n$, denote $\text{CStO} \setminus \{x\}$ the *punctured* compressed oracle that is defined equal to CStO , except that we measure whether x is included in D after every query made by the adversary \mathcal{A} . Formally, define a classical procedure f_{FIND_x} that takes as a database D and outputs 1 if and only if x is in D . That is, there exists some y such that $(x, y) \in D$. Then the measurement is implemented by initializing a new register to 0, and then evaluating f_{FIND_x} in superposition and XORing the output into the newly generated register. Finally, the register is measured. By FIND , we denote the event that the measurement results in 1 at least once after the q queries made by \mathcal{A} .

Theorem 3 (Compressed Oracle O2H). *Let CStO be a compressed standard oracle initialized on an empty database. Let $S \subset X$ and z be a random string, where (S, z) may have an arbitrary joint distribution. Then, for all quantum algorithms \mathcal{A} issuing at most q queries that, on input z , output either 0 or 1, we have*

$$\left| \Pr[1 \leftarrow \mathcal{A}^{\text{CStO}}(z)] - \Pr[1 \leftarrow \mathcal{A}^{\text{CStO} \setminus S}(z)] \right| \leq 2 \cdot \sqrt{q \Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus S}(z)]}.$$

If furthermore, $S \leftarrow 2^X$ and z are independent,

$$\Pr[\text{FIND} : \mathcal{A}^{\text{CStO} \setminus \{x\}}(z)] \leq \frac{4q \cdot |S|}{|X|}.$$

E Additional Experiments for FrodoKEM

This section provides additional experimental results on the running time of FrodoKEM and its **mKEM** variant. We recall that we started from the Github implementation of FrodoKEM¹⁰ and tweaked it to instantiate an **mKEM**.

As documented in FrodoKEM’s specification, the running time of FrodoKEM is highly dependent on two factors:

- Whether pseudorandomness is generated using AES128 or SHAKE128.
- Whether the target platform supports instructions that speed up AES128 or SHAKE128.

Conveniently, the Github implementation of FrodoKEM provides compilation flag that allow to switch between AES128 or SHAKE128 (`GENERATION_A`), and turn on/off platform-specific instructions (`OPT_LEVEL`). We set various compilation flags, and compared in each case the running times of running FrodoKEM and our **mKEM** (normalized by the number of users). Measurements were done on an Intel(R) Core(TM) i5-8250U CPU at 1.60GHz, with Turbo Boost disabled. Results are given in Tab. 5: **KEM** stands for the initial, single-recipient version of FrodoKEM, and **mKEM** stands for our **mKEM** instantiation. Note that the runtime of our **mKEM** instantiation is normalized by the number of recipient.

Our main finding is that the running time of our **mKEM** instantiation only mildly depends on the compilation flags we act on: running times differ by factors less than 2. This stands in sharp contrast with the standard **KEM**, which running time vary by factors up to 39. Logically, this impacts the computational gain k_{cycles} provided by our **mKEM** techniques. In one case, we report an (amortized) speed-up of a factor 323 by using our **mKEM** over the trivial **mKEM** (which runs the standard **KEM** in parallel).

Our interpretation of these results is that the performance of our **mKEM** instantiation is much less reliant on the presence of platform-specific instructions for AES and/or SHAKE. As a consequence, it is likely to perform across a wide array of platforms in a more consistent manner than the standard version of FrodoKEM. From a deployment perspective, we view this as a desirable characteristic.

¹⁰ <https://github.com/Microsoft/PQCrypto-LWEKE>

Table 5: Encapsulation times of FrodoKEM for different compilation flags. Running times for **mKEM** are normalized by the number of users (1000).

Compilation flags	Parameter set	Trivial KEM	Our mKEM	k_{cycles}
GENERATION_A= AES128	FrodoKEM-640	2 165 086	280 173	7.73
OPT_LEVEL= FAST	FrodoKEM-976	4 277 417	428 522	9.98
	FrodoKEM-1344	6 979 744	565 357	12.35
GENERATION_A= SHAKE128	FrodoKEM-640	5 052 476	275 510	18.34
OPT_LEVEL= FAST	FrodoKEM-976	10 671 922	448 188	23.81
	FrodoKEM-1344	18 781 051	604 893	31.05
GENERATION_A= SHAKE128	FrodoKEM-640	18 896 506	291 660	64.79
USE_OPENSSL=FALSE	FrodoKEM-976	38 617 382	443 122	87.15
OPT_LEVEL= REFERENCE	FrodoKEM-1344	73 604 047	652 053	112.88
GENERATION_A= AES128	FrodoKEM-640	65 454 403	328 682	199.14
USE_OPENSSL=FALSE	FrodoKEM-976	145 201 863	550 714	263.66
OPT_LEVEL= REFERENCE	FrodoKEM-1344	275 885 548	853 822	323.12

Table of Contents

Scalable Ciphertext Compression Techniques for Post-Quantum KEMs and their Applications	1
<i>Shuichi Katsumata, Kris Kwiatkowski, Federico Pintore, Thomas Prest</i>	
1 Introduction	1
1.1 Our Contributions and Techniques	3
2 Preliminaries	6
2.1 Hard Problems for Lattices	6
2.2 Hard Problems for Isogenies	7
3 Multi-Recipient PKE and KEM	8
3.1 Decomposable Multi-Recipient Public Key Encryption	8
3.2 Multi-Recipient Key Encapsulation Mechanism	10
3.3 Recipient Anonymity for mPKE and mKEM	11
4 FO Transform: (IND-CPA mPKE) \Rightarrow (IND-CCA mKEM)	11
4.1 Generic Construction via FO Transform	11
4.2 Proof for Classical Case	13
4.3 Proof for Quantum Case	15
4.4 Adding Recipient Anonymity	15
5 Multi-Recipient KEM from Post-Quantum Assumptions	15
5.1 Multi-Recipient KEM from Lattices	15
5.2 Multi-Recipient KEMs from Isogenies	16
6 Instantiating mKEM with NIST Candidates and CSIDH	18
6.1 Comparison Methodology	20
6.2 Instantiation with Lattice-based NIST Candidates	20
6.3 Instantiation with Isogeny-Based schemes	22
7 Application to Secure Group Messaging	24
7.1 Syntax and Notations for Group Messaging	24
7.2 Concrete Instantiations of m -ary TreeKEM	27
A Omitted Preliminaries	32
A.1 CSIDH-based assumption	32
A.2 Randomness Extraction	33
A.3 Recipient Anonymity for mPKE and mKEM	33
B Omitted Proofs from Generic Construction: Sec. 4	34
B.1 Omitted Proof of Lem. 4	34
B.2 Omitted Proof of Thm. 2	35
C Omitted Details from PQ Instantiation of mKEM: Sec. 5	40
C.1 Omitted Proofs from Lattice Part: Sec. 5.1	40
C.2 Omitted Proofs from Isogeny Part: Sec. 5.2	40
C.3 Omitted Details on CSIDH-based mPKE Sec. 5.2	41
D Background on Quantum Random Oracles	43
D.1 Compressed Oracles and QROM	44
D.2 Useful Lemmas	46

E Additional Experiments for FrodoKEM	48
---	----