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

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

[1]National Institute of Advanced Industrial Science and Technology (AIST), JP
shuichi.katsumata@aist.go.jp
[2]PQShield, UK
kris.kwiatkowski, thomas.prest@pqshield.com
[3]Mathematical Institute, University of Oxford, UK
federico.pintore@maths.ox.ac.uk

November 20, 2021

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 [OBR+20, 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.

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

[1]To be more precise, it is common to rely on the KEM/DEM framework [CS03, Den03] to lower the reliance on the more inefficient public key cryptography.

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 [BCR+] 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 [Sma05][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., [Kur02, BF07, HK07, HTAS09, MH13, Yan15]. Also, similar variants of mKEM, such as *multi-message* multi-recipient *public-key encryption* [BBM00, BPS00, Kur02, BBS03], 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 [CLQY18] 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. See Appendix A for more detail. Second, earlier works such as [BF07, HTAS09, MH13] 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, [BF07] requires a notion of *weak reproducibility*, which informally 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.

[2]We note that very similar variants of mKEM have been considered prior to this work [BBM00, BPS00, Kur02, BBS03]. More details follow.

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

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

1.1 Our Contributions and Techniques

Theoretical Contribution. We provide a new simple and efficient generic construction of an IND-CCA secure multi-recipient KEM (mKEM) from any IND-CPA secure multi-recipient PKE (mPKE).[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 [BF07, HTAS09, MH13] which provide some types of generic constructions of mKEM, ours require an mPKE whereas they only require a single-recipient PKE. However, we only require very natural properties from the underlying mPKE, such as IND-CPA. Considering that our mPKE can be instantiated with diverse assumptions (including but not limited to post-quantum assumptions) in a very natural way from standard PKEs, we believe our generic construction to be more versatile and handy than previous ones.

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

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

Additionally, we show that we can use the mKEM primitive for the TreeKEM protocol obtaining significant bandwidth savings. To give some context, the importance of TreeKEM could be best understood by looking at its parent protocol, MLS [OBR+20, BBM+20], an IETF draft for secure (group) messaging. MLS has gained considerable industrial traction and has attracted a fair amount of academic scrutiny. TreeKEM constitutes the cryptographic backbone of MLS, as well as its main bottleneck in bandwidth and computational efficiency. Indeed, given N users, it requires each of them to compute and send $O(\log N)$ ciphertexts at regular intervals. We highlight a simple but powerful interplay between TreeKEM and mKEM, and show that by applying our technique we can reduce communication cost by a factor between 1.8 and 4.2 compared to using standard KEMs.

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

1.1.1 Our Techniques: Generic Construction of IND-CCA secure mKEM.

On a high level, our generic construction can be seen as a generalization of the Fujisaki-Okamoto (FO) transform [FO99]. The FO transform (roughly) converts any IND-CPA secure PKE into an IND-CCA secure KEM. There are several variants of the FO transform and most of the variants are secure in the ROM [OP01, CHJ+02, Den03, HHK17] and/or QROM [TU16, HHK17, SXY18, JZC+18, JZM19b, JZM19a, BHH+19, Zha19, KSS+]. The high-level construction is as follows: to encrypt, we sample a random message $\mathsf{M} \leftarrow \mathcal{M}$ and derive randomness for the underlying encryption algorithm of the PKE by hashing M with a hash function G modeled as a (Q)RO. That is, $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M}; \mathsf{G}(\mathsf{M}))$. The session key is then set as $\mathsf{K} := \mathsf{H}(\mathsf{M})$, where H is another hash function modeled as a (Q)RO. To decrypt, we first decrypt $\mathsf{M}' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$ and then only accept $\mathsf{K} = \mathsf{H}(\mathsf{M}')$ if M' re-encrypts back to ct, that is, we check $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{M}'; \mathsf{G}(\mathsf{M}'))$. Although the actual proof is rather complicated, intuitively, it achieves IND-CCA security since the adversary must have queried message M to G to have constructed a valid ciphertext ct for message M. 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: $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M}; \mathsf{r}) \rightarrow \vec{\mathsf{ct}}$, where $\vec{\mathsf{ct}}$ is targeted to the set of N recipients with public keys $(\mathsf{pk}_i)_{i\in[N]}$. There is also an extraction algorithm mExt which takes as input an index $i \in [N]$ and $\vec{\mathsf{ct}}$, and outputs the ciphertext component ct_i targeted to the i-th recipient, say R_i, holding pk_i. Recipient R_i can then run the decryption algorithm on ct_i using its secret key sk_i. The reason why the FO transform cannot be directly applied to mPKE becomes clear. Assume $\mathbf{r} = \mathsf{G}(\mathsf{M})$ and that recipient R_i decrypted to M. Then, to check validity of ct_i, R_i must re-encrypt the *entire* ciphertext $\vec{\mathsf{ct}}$ by running $\mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M}; \mathsf{r})$. Therefore, the decapsulation time will depend on N, which is highly undesirable.

To get around this issue, in this work we consider a slight variant of mPKE with a *decomposable* flavor. Informally, a decomposable multi-encryption algorithm mEnc takes randomness of the form $\mathsf{r} = (\mathsf{r}_0, \mathsf{r}_1, \cdots, \mathsf{r}_N)$ as input, and creates a public-key-*independent* ciphertext $\mathsf{ct}_0 \leftarrow \mathsf{mEnc}^{\mathsf{i}}(\mathsf{r}_0)$ and public-key-*dependent* ciphertexts $\widehat{\mathsf{ct}}_i \leftarrow \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i)$. The resulting ciphertext for recipient R_i is then $\mathsf{ct}_i = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$. We view this as a natural formalization of mPKE as it is satisfied by all the mPKE constructions that we are aware of. Moreover, this feature is desirable in practice as it allows to parallelize part of the encryption algorithm. Now, to perform the FO transform, we derive $\mathsf{r}_0 = \mathsf{G}(\mathsf{M})$ and $\mathsf{r}_i = \mathsf{G}(\mathsf{pk}_i, \mathsf{M})$. It is evident that R_i can re-encrypt and check the validity of its ciphertext. Notably, the decapsulation time is now independent of N. With this new formalization, the proof in the (classical) ROM follows in a straightforward manner (with minor modification) from the standard FO transform [HHK17].

The security proof of our mKEM in the *quantum* ROM (QROM) requires slight 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 [TU16, HHK17, SXY18, JZC+18, JZM19b, JZM19a, BHH+19] – seem to be an ill fit for mPKE. This is because in the multi-recipient setting, the decapsulation oracle is required to output an *identical* session key K when the ciphertext is valid, while it is required to output *different* rejection values for each users when the ciphertext is invalid. Due to this discrepancy between valid and invalid ciphertexts (i.e., the former requires to output an identical random value, whereas the latter requires to output different random values), 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 always 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 [Zha19]. 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 [Zha19] 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 2.1 (Learning with Errors (LWE))**.** *Let d, q, n_1, n_2, n_3 be natural numbers, and D_{s} and D_{e} be distributions over R_q. We say that the advantage of algorithm $\mathcal{A}$ in solving the (decisional) $\mathsf{LWE}_{n_1,n_2,n_3}$ problem over the ring R_q is*

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

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

Roughly, when $d = 1$ and $n_1 = n_2 \geq 1$ (with an appropriate choice of distributions $D_{\mathsf{s}}, D_{\mathsf{e}}$), the above corresponds to the standard LWE [Reg05], and when $d \geq 1$ and $n_1 = n_2 = 1$, the above corresponds to the ring LWE (Ring-LWE) problem [LPR10, LPR13]. 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 [LS15]. 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 [BPR12], where the least significant bits are removed. The benefit of this variant is that we no longer require to sample the noise, as it is removed. Below the function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$, where $q > p \geq 2$, is defined as $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \mod p$. The definition of the LWR problem follows.

Definition 2.2 (Learning with Rounding (LWR))**.** *Let d, p, q, n_1, n_2, n_3 be natural numbers such that $q > p$, and D_{s} a distributions over R_q. We say that the advantage of algorithm $\mathcal{A}$ in solving the (decisional) $\mathsf{LWR}_{n_1,n_2,n_3}$ problem over the rings R_p and R_q is*

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

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

2.2 Hard Problems for Isogenies

In the following sections we propose two different isogeny-based schemes: one stemming from the SIDH key exchange [DFJP14] and the other from the CSIDH key exchange [CLM+18]. Both key exchanges share common mathematical tools, but several technical differences make them, and their descendants, substantially different. As a consequence, schemes in the SIDH family rely on hardness assumptions different from those used for schemes in the CSIDH family. Our schemes make no exception, as they use distinct security assumptions.

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

The security of our scheme relies on a decisional variant, named SSDDH [DFJP14], of the SSCDH assumption. The latter is used by one of NIST second-round candidate KEMs, called SIKE [JAC+19], which is deduced from the key exchange SIDH.

Definition 2.3 (Supersingular Decisional Diffie-Hellman (SSDDH))**.** *Let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$ such that $|E(\mathbb{F}_{p^2})| = (2^{e_2}3^{e_3})^2$. We say that the advantage of algorithm $\mathcal{A}$ in solving the $\mathsf{SSDDH}_{p,E,B_2,B_3}$ problem is*

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

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

CSIDH-based assumption. Let p be an odd prime of the form $4\ell_1\ell_2\cdots\ell_t - 1$, where $\ell_1, \ldots, \ell_t$ 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 $\mathrm{End}_p(E)$ 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 $\mathbb{F}_p$ [CLM+18]. The ideal class group $\mathcal{C}\ell(\mathcal{O})$ acts freely and transitively on $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ via the group action $\star$. A special integral ideal $\mathfrak{I}_{\ell_i} \subset \mathcal{O}$ corresponds to each prime ℓ_i. These ideals allow an easy computation of the group action. In particular, the action of $\mathfrak{I}_{\ell_i}$ 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 $\mathfrak{g} \in \{[\mathfrak{I}_{\ell_i}] | i = 1, \ldots, t\}$. The security of the CSIDH-based scheme we introduce in Section 5.2 relies on the decisional variant, recently exploited in [EKP20, CS20], of the best-known GAIP assumption [CLM+18].

Definition 2.4 (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 $\mathsf{dCSIDH}_{p,\mathfrak{g}}$[4] problem is*

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

We say the $\mathsf{dCSIDH}_{p,\mathfrak{g}}$ 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 group G is know, and it adheres to the assumption made above, i.e. G cyclic and generated by $\mathfrak{g} \in \{[\mathfrak{I}_{\ell_i}] | i = 1, \ldots, t\}$. For the other two sets of parameters, efficiency reasons dictate to restrict to elements of G of the form $\mathfrak{I}_{\ell_i}^{e_i} \cdots \mathfrak{I}_{\ell_t}^{e_t}$, 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 $\{\mathfrak{I}_{\ell_i}^{e_i} \cdots \mathfrak{I}_{\ell_t}^{e_t} \,|\, |e_i| \leq B\}$ determines a distribution D_B over G and, fixing $E_0 \in \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$, a distribution D_{E_0} over $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ itself. The distribution D_B is heuristically assumed to be close to the uniform distribution over G. 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 $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$, the curve E is sampled from D_{E_0}, while elements sampled from D_B replace the powers of $\mathfrak{g}$.

[4] We intentionally put d in dCSIDH so as not to confuse the CSIDH key exchange protocol [CLM+18] with the assumption.

2.3 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 2.5 (Universal Hash Functions)**.** *A family of functions $\mathcal{H} = \{\mathsf{H}_k : \mathcal{X} \to \mathcal{D}\}_{k\in K}$ is called a family of universal hash functions, if for all $x, x' \in \mathcal{X}$ with $x \neq x'$, we have $\Pr_{\mathsf{H}\leftarrow\mathcal{H}}[\mathsf{H}(x) = \mathsf{H}(x')] \leq \frac{1}{|\mathcal{D}|}$.*

It is well known that one can extract uniform random bits from a high min-entropy source using universal hash functions [HILL99].

Lemma 2.6 (Leftover Hash Lemma)**.** *Let $\mathcal{H} = \{\mathsf{H}_k : \mathcal{X} \to \mathcal{D}\}_{k\in K}$ 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 $U(D)$ be the uniform distribution over D. Then, the following holds*

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

3 Multi-Recipient PKE and KEM

3.1 Decomposable Multi-Recipient Public Key Encryption

Definition 3.1 (Decomposable Multi-Recipient Public Key Encryption)**.** *A* (single-message) decomposable multi-recipient public key encryption (mPKE) *over a message space $\mathcal{M}$ and ciphertext spaces $\mathcal{C}$ and $\mathcal{C}_{\mathsf{single}}$ consists of the following five algorithms $\mathsf{mPKE} = (\mathsf{mSetup}, \mathsf{mGen}, \mathsf{mEnc}, \mathsf{mExt}, \mathsf{mDec})$:*

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

Although we can consider *non-decomposable* multi-recipient PKEs, we only focus on decomposable schemes as they are compatible with the Fujisaki-Okamoto (FO) transform [FO99]. 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 $\vec{\mathsf{ct}}$ 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 3.2 (Correctness). *A* mPKE *is* δ*-correct if*

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

where the expectation is also taken over $\mathsf{pp} \leftarrow \mathsf{mSetup}(1^\kappa)$ *and* $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{mGen}(\mathsf{pp})$.[5]

We also define the notion of *well-spreadness* [FO99] which states informally that the ciphertext has high min-entropy.

Definition 3.3 (γ-Spreadness). *Let* mPKE *be a decomposable multi-recipient* PKE *with message space* $\mathcal{M}$ *and ciphertext spaces* $\mathcal{C}$ *and* $\mathcal{C}_{\mathsf{single}}$. *For all* $\mathsf{pp} \in \mathsf{Setup}(1^\kappa)$, *and* $(\mathsf{pk},\mathsf{sk}) \in \mathsf{Gen}(\mathsf{pp})$, *define*

$$\gamma(\mathsf{pp},\mathsf{pk}) := -\log_2\left(\max_{\mathsf{ct}\in\mathcal{C}_{\mathsf{single}},\mathsf{M}\in\mathcal{M}} \Pr_{\mathsf{r}_0,\mathsf{r}}\left[\mathsf{ct} = \left(\mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp};\mathsf{r}_0),\mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp},\mathsf{pk},\mathsf{M};\mathsf{r}_0,\mathsf{r})\right)\right]\right).$$

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

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

Definition 3.4 (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 Figure 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*

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

Remark 3.5 (Insider corruption). We point out that insider corruptions for mPKE are not considered [Sma05, BF07]. 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 3.6 (Inefficient mPKE from any standard (single-recipient) PKE). Our definition of mPKE captures the trivial solution of sending different ciphertexts obtained with a standard single-recipient PKE to multiple recipients. That is, independently encrypting the same message to all recipients using their respective public keys. In the above syntax of mPKE, this amounts to setting $\mathsf{mEnc}^{\mathsf{i}}$ as a null function and setting r_0 as an empty string. Also, mExt will simply pick the relevant ciphertext component for the particular recipient. Therefore, in the context of ciphertext compression, the goal is to obtain a mPKE with better efficiency/ciphertext-size compared to this trivial method.

Remark 3.7 (Number of recipients). In general, the number of recipients $N = \mathsf{poly}(\kappa)$ can be chosen arbitrary by the sender (or adversary). Some schemes may require an upper bound on N since the concrete provably-secure parameters may have a dependance on N, e.g., the reduction loss degrades by a factor of $1/N$. Our proposal does not require such an upper bound since N only shows up in a statistical manner, and so we can handle large N, say $N = 2^{15}$, without having any large impact on the concrete parameter choice.

3.2 Multi-Recipient Key Encapsulation Mechanism

Definition 3.8 (Multi-Recipient Key Encapsulation Mechanism). *A* (single-message) multi-recipient key encapsulation mechanism (mKEM) *over a key space* $\mathcal{K}$ *and ciphertext space* $\mathcal{C}$ *consists of the following five algorithms* $\mathsf{mKEM} = (\mathsf{mSetup},\mathsf{mGen},\mathsf{mEncaps},\mathsf{mExt},\mathsf{mDecaps})$:

- $\mathsf{mSetup}(1^\kappa) \rightarrow \mathsf{pp}$: *The setup algorithm on input the security parameter* 1^κ *outputs a public parameter* pp.

[5] We could have defined correctness with respect to N-recipients as we do for mKEM (see Definition 3.9), however, we use this single recipient definition since it is more easier to handle during in the security proof.

GAME IND-CPA

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

GAME IND-CCA

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

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

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

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

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

Definition 3.9 (Correctness)**.** *A* mKEM *is* δ_N*-correct if*

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

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

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

Definition 3.10 (IND-CCA)**.** *Let* mKEM *be a multi-recipient* KEM*. We define* IND-CCA *by a game illustrated in Figure 1 and say the (possibly quantum) adversary* $\mathcal{A}$ *(making only classical decapsulation queries to* $\mathcal{D}$*)* wins *if the game outputs* 1*. We define the advantage of* $\mathcal{A}$ *against* IND-CCA *security of* mKEM *parameterized by* $N \in \mathbb{N}$ *as*

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

We note that similarly to the remark made in Remark 3.6, 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. This is formally provided in the following Definitions 3.11 and 3.12.

Definition 3.11 (IND-Anon-CPA)**.** *Let* mPKE *be a multi-recipient PKE. We define* IND-Anon-CPA *associated with a PPT* fake encryption *algorithm* $\overline{\mathsf{mEnc}}$ *by a game illustrated in Figure 2 and say the (possibly quantum) adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ wins *if the game outputs* 1*. For quantum adversaries* $\mathcal{A}$*, we define the advantage against* IND-Anon-CPA *security of* mPKE *parameterized by* $N \in \mathbb{N}$ *as*

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

Definition 3.12 (IND-Anon-CCA)**.** *Let* mKEM *be a multi-recipient KEM. We define* IND-Anon-CCA *associated with a PPT* fake encryption *algorithm* $\overline{\mathsf{mEncaps}}$ *by a game illustrated in Figure 2 and say the (possibly quantum) adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *(making only classical decryption queries to* $\mathcal{D}$*)* wins *if the game outputs* 1*. For quantum adversaries* $\mathcal{A}$*, we define the advantage against* IND-Anon-CCA *security of* mKEM *parameterized by* $N \in \mathbb{N}$ *as*

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

GAME IND-Anon-CPA

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

GAME IND-Anon-CCA

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

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

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

Figure 2: IND-Anon-CPA of mPKE and IND-Anon-CCA of mKEM.

4 FO Transform: (IND-CPA mPKE) ⇒ (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 Figure 3. The scheme provides some form of *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. (See Remark 4.3 for more discussion). In Figure 3, $\mathsf{G}_1, \mathsf{G}_2, \mathsf{H}, \mathsf{H}'$ are hash functions modeled as random oracles in the security proof. They can be simulated by a single random oracle by using appropriate domain separation. Finally, we include an ℓ-bit seed to perform implicit rejection by viewing $\mathsf{H}'(\mathsf{seed}, \cdot)$ as a pseudorandom function in the (Q)ROM.

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. We note that correctness of our mKEM trivially holds from the correctness of the underlying mPKE.

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

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

$\underline{\mathsf{mGen}(\mathsf{pp})}$

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

$\underline{\mathsf{mExt}(i, \vec{\mathsf{ct}})}$

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

$\underline{\mathsf{mEncaps}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]})}$

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

$\underline{\mathsf{mDecaps}(\mathsf{sk}, \mathsf{ct})}$

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

Figure 3: An IND-CCA secure mKEM from a decomposable IND-CPA secure mPKE $= (\mathsf{mSetup}^{\mathsf{p}}, \mathsf{mGen}^{\mathsf{p}}, \mathsf{mEnc} = (\mathsf{mEnc}^{\mathsf{i}}, \mathsf{mEnc}^{\mathsf{d}}), \mathsf{mExt}^{\mathsf{p}}, \mathsf{mDec})$. We include the superscript $^{\mathsf{p}}$ to make the code more readable.

Theorem 4.1 (Classical: IND-CPA mPKE $\Rightarrow$ IND-CCA mKEM). *Assume* mPKE *with message space $\mathcal{M}$ is δ-correct and γ-spread. Then, for any classical PPT* IND-CCA *adversary $\mathcal{A}$ issuing at most $q_{\mathcal{D}}$ queries to the decapsulation oracle $\mathcal{D}$, a total of at most q_{G} queries to G_1 and G_2, and at most q_{H} and q'_{H} queries to H and H', there exists a classical PPT adversary $\mathcal{B}_{\mathsf{IND}}$ such that*

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

where the running time of $\mathcal{B}_{\mathsf{IND}}$ is about that of $\mathcal{A}$, and ℓ is bit-length of the seed included in the private key.

Theorem 4.2 (Quantum: IND-CPA mPKE $\Rightarrow$ IND-CCA mKEM). *Assume* mPKE *with message space $\mathcal{M}$ is δ-correct and γ-spread. Then, for any quantum PT* IND-CCA *adversary $\mathcal{A}$ issuing at most $q_{\mathcal{D}}$ classical queries to the decapsulation oracle $\mathcal{D}$, a total of at most q_{G} quantum queries to G_1 and G_2, and at most q_{H} and q'_{H} quantum queries to H and H', there exists a quantum PT adversary $\mathcal{B}_{\mathsf{IND}}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{mKEM},N}(\mathcal{A}) \leq \sqrt{8 \cdot (q_{\mathsf{G}} + 1) \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}})} + \frac{8 \cdot (q_{\mathsf{G}} + 1)}{\sqrt{|\mathcal{M}|}} \\ + 12 \cdot (q_{\mathsf{G}} + q_{\mathcal{D}} + 1)^2 \cdot \delta_N + q_{\mathcal{D}} \cdot 9\sqrt{2^{-\gamma}} + 9 \cdot 2^{\frac{-\mu}{2}} + q'_{\mathsf{H}} \cdot N \cdot 2^{\frac{-\ell+1}{2}},\end{aligned}$$

where the running time of $\mathcal{B}_{\mathsf{IND}}$ is about that of $\mathcal{A}$, ℓ is bit-length of the seed included in the private key, and $\mu = |\mathsf{r}_0| + |\mathsf{r}|$ for $(\mathsf{r}_0, \mathsf{r}) \in \mathcal{R}$ where $\mathcal{R}$ is the randomness space of mPKE *for a single ciphertext.*

Remark 4.3 (Implicit vs explicit rejection). In our construction in Figure 3, we use *implicit* rejection, where mDecaps does not explicitly output $\perp$ to indicate that the input ciphertext was invalid. This may be suitable in practice when we do not want to let the adversary know that decapsulation failed. However, we note that our proof is agnostic to this choice, and in particular, the same proof can be shown in case we want *explicit* rejection, where mDecaps outputs $\perp$ in case either $\mathsf{M} = \perp$ or ct is not the same as the reencrypted ciphertext $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$. Concretely, we obtain an IND-CCA secure mKEM with explicit rejection by simply outputting $\perp$ rather than outputting $\mathsf{H}'(\mathsf{seed}, \mathsf{ct})$ in Figure 3. We point out 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 [SXY18, BHH+19] for an example).

Finally, we would like to highlight some subtle differences in the meaning of implicit/explicit rejections between a single-user KEM and an mKEM. In the single-user setting, the meaning of implicit and explicit rejection is clear; the adversary cannot check the validity of a ciphertext by querying the decapsulation oracle since it only receives a random session key regardless of the validity of the ciphertext. In contrast, in the multi-user setting, we can consider implicit/explicit rejection concerning a single ciphertext ct_i or for the entire ciphertext $\vec{\mathsf{ct}}$. Our work provides implicit/explicit rejection in the former sense since an adversary with only ct_i cannot check the validity of the ciphertext. In fact, we observe that obtaining an mKEM scheme with implicit rejection in the latter sense is impossible. If this was possible, then regardless of a ciphertext $\vec{\mathsf{ct}}$ being valid or not, when the adversary queries the respective components of $\vec{\mathsf{ct}} = (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_i)_{i\in[N]})$ to each decapsulation oracle corresponding to users 1 to N, it must receive an identical random session key from each oracle. Then consider an adversary that modifies a valid ciphertext $\vec{\mathsf{ct}}$ to an invalid ciphertext $\vec{\mathsf{ct}}' = (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_1, \cdots, \widehat{\mathsf{ct}}_{N-1}, \widehat{\mathsf{ct}}'_N))$ that queries the decapsulation oracles. Since the decapsulation oracles corresponding to users 1 to $N-1$ are agnostic to the change made by the N-th user's ciphertext, they respond identically for both $\vec{\mathsf{ct}}$ and $\vec{\mathsf{ct}}'$. Then by assumption, the decapsulation oracle corresponding to the N-th user must respond with the same session key when it is queried on $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_N)$ and $(\mathsf{ct}_0, \widehat{\mathsf{ct}}'_N)$ as well. However, this means $\vec{\mathsf{ct}}'$ decrypts to $\vec{\mathsf{ct}}$ forming a contradiction. Therefore, in the case of mKEM, it seems we can only hope for implicit/explicit rejection concerning a single ciphertext.

4.2 Proof for Classical Case

We provide the proof of Theorem 4.1.

Proof of Theorem 4.1. Let $\mathcal{A}$ be a classical PPT adversary against the IND-CCA security of mKEM. Without loss of generality, we make a simplifying argument that $\mathcal{A}$ queries the same message to both oracles G_1 and G_2. That is, when $\mathcal{A}$ queries for an input M to the oracles, it receives back $(\mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_1, \mathsf{M}), \cdots, \mathsf{G}_2(\mathsf{pk}_N, \mathsf{M}))$. It is clear that this modification does not weaken $\mathcal{A}$, and moreover, we can always transform an adversary $\mathcal{A}$ that does not query all the oracles on the same input to an adversary that does. Below, we upper bound $\mathcal{A}$'s advantage by considering a sequence of games. We denote by E_i the event $\mathcal{A}$ wins in Game_i.

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

- Game_2: In this game, we replace the computation of $\mathsf{H}'(\mathsf{seed}_i, \cdot)$ by a random function $\widehat{\mathsf{H}}'_i(\cdot)$ in case $\mathsf{M} = \perp$ or $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ occurs when answering the decapsulation oracle with input $i \in [N]$. Since this modification remains unnoticed by the adversary unless $\mathsf{H}'(\mathsf{seed}, \cdot)$ is queried for any $\mathsf{seed} \in \{\mathsf{seed}_i\}_{i\in[N]}$, we have

$$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| \leq \frac{q'_{\mathsf{H}} \cdot N}{2^\ell}.$$

- Game_3: In this game, we add an additional check at the end of the game to see if a "bad" randomness was ever used. Define $\mathsf{aux}_i := (\mathsf{pp}, (\mathsf{pk}_i, \mathsf{sk}_i))$ for $i \in [N]$ and define the sets of bad randomness as

$$\mathcal{R}^{\mathsf{bad}}_i(\mathsf{aux}_i, \mathsf{M}) := \left\{ (\mathsf{r}_0, \mathsf{r}_i) \;\middle|\; \begin{array}{c} \mathsf{M} \neq \mathsf{mDec}(\mathsf{sk}^{\mathsf{p}}_i, (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)), \text{where} \\ \mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0), \widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i) \end{array} \right\}.$$

In addition, define S to be the set of index-message pairs (excluding the pairs with message equal to $\perp$) that was obtained by decrypting the ciphertext when answering the decapsulation query (i, ct). Then, after $\mathcal{A}$ outputs its guess at the end of the game, the challenger checks if a pair of tuples $(\mathsf{M}, \mathsf{r}_0)$ and $((\mathsf{pk}_i, \mathsf{M}), \mathsf{r}_i)$ for any $(i, \mathsf{M}) \in S$ and $(\mathsf{r}_0, \mathsf{r}_i) \in \mathcal{R}^{\mathsf{bad}}_i(\mathsf{aux}_i, \mathsf{M})$ are listed in the random oracles G_1 and G_2, respectively. We call the event $\mathsf{BAD}_{\mathsf{rand}}$ if such a set of tuples are 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 $\mathsf{BAD}_{\mathsf{rand}}$ occurs, we have $|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| \leq \Pr[\mathsf{BAD}_{\mathsf{rand}}]$. Here, we can upper bound $\Pr[\mathsf{BAD}_{\mathsf{rand}}]$

by $(q_{\mathsf{G}} + q_{\mathcal{D}} + 1) \cdot \max_i\{\left|\mathcal{R}_i^{\mathsf{bad}}(\mathsf{aux}_i, \mathsf{M})\right| / |\mathcal{R}|\}$, where $\mathcal{R}$ is the randomness space of a single ciphertext. By definition, we have $\delta \geq \mathbb{E}\left[\max_{\mathsf{M}\in\mathcal{M}} |\mathcal{R}_i^{\mathsf{bad}}(\mathsf{aux}_i, \mathsf{M})|/|\mathcal{R}|\right]$, where the expectation is taken over the randomness used to sample $\mathsf{aux}_i = (\mathsf{pp}, (\mathsf{pk}_i, \mathsf{sk}_i))$ and δ is the correctness parameter of mPKE. Hence, we conclude

$$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| \leq (q_{\mathsf{G}} + q_{\mathcal{D}} + 1) \cdot \delta.$$

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

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

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

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

Figure 4: Decapsulation oracles of Game_4 and Game_5. We enforce ct is not $\mathsf{ct}_i^* := \mathsf{mExt}(i, \vec{\mathsf{ct}}^*)$ at the input level for simplicity.

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

- Game_4: In this game, we add an additional check when answering the decapsulation oracle query. This is illustrated in Figure 4 where the red underline indicates the modification. Here, $\mathcal{L}_{\mathsf{G}}$ is a list that stores the random oracle queries made to G_1 and G_2.We have $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ if G_1 was queried on M and G_2 was queried on $(\mathsf{pk}, \mathsf{M})$ for any pk. Note that due to our assumption on $\mathcal{A}$, there does not exist an event where either G_1 or G_2 was queried on M or $(\mathsf{pk}, \mathsf{M})$ while the other oracle was not.

The only difference occurs when $\mathcal{A}$ queries a ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ such that $\mathsf{M} := \mathsf{mDec}(\mathsf{sk}_i^{\mathsf{p}}, \mathsf{ct})$ has not been queried to the random oracles G_1 and G_2 but $\mathsf{ct}_0 = \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$ and $\widehat{\mathsf{ct}}_i = \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$. Since $\mathsf{G}_1(\mathsf{M})$ and $\mathsf{G}_2(\mathsf{pk}_i, \mathsf{M})$ are information theoretically hidden from $\mathcal{A}$, we can use γ-spreadness of mPKE to conclude

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_4]| \leq q_{\mathcal{D}} \cdot 2^{-\gamma}.$$

- Game_5: In this game, we further modify the way a decapsulation-oracle query is answered. This is illustrated in Figure 4, where notice that we no longer require the secret keys sk_i to answer the queries.

If the decapsulation oracle in Game_4 outputs $\mathsf{K} := \mathsf{H}(\mathsf{M})$, then $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ and $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ holds. Therefore, the decapsulation oracle in Game_5 outputs K as well. On the other hand, assume the decapsulation oracle in Game_5 outputs $\mathsf{K} := \mathsf{H}(\mathsf{M})$ for some $\mathsf{M} \in \mathcal{L}_{\mathsf{G}}$ such that $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ where $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$ and $\widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$. Then, conditioning on no correctness error occurs, ct must decrypt to M. Hence, this implies that the decapsulation oracle Game_4 outputs the same K as well. Combining the arguments together, we get

$$\Pr[\mathsf{E}_4] = \Pr[\mathsf{E}_5].$$

- Game_6: In this game, we undo the change we made in Game_3 and no longer check whether a randomness that leads to a decryption error was sampled at the end of the game. Due to the same argument as before, we have

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_6]| \leq (q_{\mathsf{G}} + 1) \cdot \delta,$$

where the $q_{\mathcal{D}}$ factor is removed from the bound since the decapsulation oracle is no longer re-encrypting due to the change we made in Game_5. At this point, the challenger no longer requires the secret keys sk_i.

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

- Game_7: In this game, we sample the random message $\mathsf{M}^* \leftarrow \mathcal{M}$ to be used to generate the challenge ciphertext at the beginning. We then define Query as the event that $\mathcal{A}$ queries the random oracles $\mathsf{H}(\cdot)$, $\mathsf{G}_1(\cdot)$, or $\mathsf{G}_2(\star,\cdot)$ on input M^*, where $\star$ denotes an arbitrary element. When Query occurs, we abort the game and force $\mathcal{A}$ to output a random bit. We show in Lemma 4.4 that we have

$$|\Pr[\mathsf{E}_6] - \Pr[\mathsf{E}_7]| \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}}) + \frac{(q_{\mathsf{G}} + q_{\mathsf{H}})}{|\mathcal{M}|}$$

for some classical PPT adversary $\mathcal{B}_{\mathsf{IND}}$ with similar runtime as $\mathcal{A}$.

Before providing the proof of Lemma 4.4, we finish our proof for our main statement. In Game_7, when the adversary $\mathcal{A}$ does not query the random oracle H on M^*, the key K_0^* is distributed exactly the same as K_1^*. Moreover, when Query occurs (i.e., $\mathcal{A}$ queries H on M^*), $\mathcal{A}$ is forced to output a random bit. Therefore, we have

$$\Pr[\mathsf{E}_7] = \frac{1}{2}.$$

Combining everything together, we obtain the statement in Theorem 4.1.

To complete the proof, it remains to prove Lemma 4.4 below.

Lemma 4.4. *We have $|\Pr[\mathsf{E}_6] - \Pr[\mathsf{E}_7]| \leq 2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}}) + \frac{(q_{\mathsf{G}}+q_{\mathsf{H}})}{|\mathcal{M}|}$ for some classical PPT adversary $\mathcal{B}_{\mathsf{IND}}$ with a runtime about the same as that of $\mathcal{A}$.*

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

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

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

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

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

□

□

4.3 Proof for Quantum Case

The main difference between our proof and prior proofs for IND-CCA secure KEM in the QROM, e.g., [TU16,HHK17,SXY18,JZC+18,JZM19b,JZM19a,BHH+19], is that we use the *lazy sampling* with *compressed* quantum oracles introduced in [Zha19] (for reasons explained in the introduction). This allows the simulator to check the validity of the ciphertext submitted to the decapsulation oracle without interfering with the adversary's state. Since other than how we specify and interact with the random oracle, the proof structure is essentially the same as the classical case, we provide the full proof in Appendix B.

4.4 Adding Recipient Anonymity

The construction provided in Section 4.1 immediately give rise to a *recipient anonymous* mKEM if we additionally assume the underlying IND-CPA secure mPKE is IND-Anon-CPA secure. In particular, we define the fake encapsulation algorithm $\overline{\mathsf{mEncaps}}$ (see Section 3.3) as: sample $\mathsf{K} \leftarrow \mathcal{K}$, run $\vec{\mathsf{ct}} \leftarrow \overline{\mathsf{mEnc}}(\mathsf{pp}, N)$, and output $(\mathsf{K}, \vec{\mathsf{ct}})$, where $\overline{\mathsf{mEnc}}$ is the fake encryption algorithm of the underlying mPKE (see Section 3.3). The only modification to the proofs of Theorems 4.1 and 4.2 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 show that $\vec{\mathsf{ct}}^*$ is distributed independently of the public keys $(\mathsf{pk}_i)_{i\in[N]}$. We omit the full proof as it directly reduces from the IND-Anon-CPA security game.

5 Multi-Recipient KEM from Post-Quantum Assumptions

We provide two types of IND-CCA secure mKEM instantiations: one scheme based on lattices, and two schemes based on isogenies (in the SIDH and CSIDH setting). Specifically, we provide two types of IND-CPA secure mPKEs and use Theorems 4.1 and 4.2 to generically convert them into IND-CCA secure mKEMs in the ROM and QROM, respectively. As we see in Section 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 [LP11] 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 Remark 3.6).

The mPKE scheme based on the Lindner-Peikert framework [LP11] is provided in Figure 5. 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 Figure 5 has all the properties required for applying the "multi-recipient" Fujisaki-Okamoto transform (Theorems 4.1 and 4.2). First, it is straightforward to see that we can easily set the parameters as to have δ-correctness and γ-spreadness for exponentially small δ and $2^{-\gamma}$. Moreover, practical schemes such as NIST candidates also allow for exponentially small δ and $2^{-\gamma}$. It remains to show that the Linder-Peikert framework provides not only a secure PKE but also a secure mPKE.

IND-CPA *Security.* It is straightforward to see that IND-CPA security follows naturally from the LWE assumption. We provide the proof below for completeness.

Lemma 5.1. *Assume* mPKE *as shown in Figure 5. Then, for any (classical or quantum)* IND-CPA *adversary* $\mathcal{A}$*, there exist (classical or quantum) adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

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

Proof. 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 5: Lattice-based mPKE via the Lindner-Peikert framework [LP11]. mExt with input index i is defined by picking the relevant components $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ from $\vec{\mathsf{ct}}$.

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

- Game_2: In this game, we change how the public keys $(\mathsf{pk}_i)_{i\in[N]}$ are created. Rather than generating each $\mathbf{B}_i$ as $\mathbf{AS}_i + \mathbf{E}_i$ 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 $\mathsf{LWE}_{n,n,Nm}$ assumption. Hence, we can construct an adversary $\mathcal{B}_1$ with around the same running time as $\mathcal{A}$ such that

$$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| \le \mathsf{Adv}^{\mathsf{LWE}}_{n,n,Nm}(\mathcal{B}_1).$$

- Game_3: In this game, we change how the challenge ciphertext $\vec{\mathsf{ct}}^*$ is created. Namely, rather than sampling $\vec{\mathsf{ct}}^* := (\mathbf{U}, (\mathbf{V}_i)_{i\in[N]})$ 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 $\mathsf{LWE}_{(n+Nm),n,\bar{m}}$, where note that we take the transpose of the LWE sample since the secret matrix (i.e., $\mathbf{R} \leftarrow D_{\mathsf{s}}^{\bar{m}\times n}$) 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

$$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| \le \mathsf{Adv}^{\mathsf{LWE}}_{(n+Nm),n,\bar{m}}(\mathcal{B}_2).$$

Finally, in Game_3, the challenge ciphertext is distributed uniformly random and independently from the challenge bit b. Therefore, no adversary can have a distinguishing advantage. Hence, $\Pr[\mathsf{E}_3] = 1/2$. Combining everything together, we get the desired bound. □

IND-Anon-CPA *Security.* It is clear that the above proof for IND-CPA security is also a proof for IND-Anon-CPA. This can be checked by observing that in the final game, we no longer require the users public keys $(\mathsf{pk}_i = \mathbf{B}_i)_{i\in[N]}$ to simulate the challenge ciphertext. In particular, the fake encryption algorithm $\overline{\mathsf{mEnc}}$ simply outputs a random element in $R_q^{\bar{m}\times n} \times (R_q^{\bar{m}\times m})^N$.

Remark 5.2 (Using LWR instead of LWE). The mPKE presented in Figure 5 readily generalizes to the LWR setting. The only difference is that instead of adding the noise terms (i.e., $\mathbf{E}, \mathbf{E}', \mathbf{E}_i''$), we round. For instance, the public key pk will be $\lfloor \mathbf{AS} \rceil_p \in R_p^{n\times m}$ rather than $\mathbf{AS} + \mathbf{E} \in R_q^{n\times m}$. It is easy to show that mPKE has γ-spreadness, is δ-correct and 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 [DFJP14] 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 Theorems 4.1 and 4.2 for obtaining an IND-CCA secure mKEM. Since a unified presentation of the two schemes would be rather convoluted, for the sake of readability we differentiate their explanations. We note that both schemes require a family of universal hash functions $\mathcal{H} = \{\mathsf{H}_k : \mathcal{X} \subset \mathbb{F} \to \{0,1\}^w\}_{k\in K}$ indexed by a finite set K, where $\mathbb{F}$ denotes a finite field. The two schemes are detailed below, starting from the SIDH-based one.

Isogeny-based mPKE via SIDH. The mPKE deduced from SIDH is provided in Figure 6. We highlight that the public parameter pp output by mSetup on input a security parameter 1^κ consists of: a prime p of the form $2^{e_2}3^{e_3} - 1$; a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ and such that $|E(\mathbb{F}_{p^2})| = (2^{e_2}3^{e_3})^2$; bases $B_2 = \{P_2, Q_2\}$ and $B_3 = \{P_3, Q_3\}$ for $E[2^{e_2}]$ and $E[3^{e_3}]$, respectively; a hash function H uniformly sampled from a family of universal hash functions $\mathcal{H} = \{\mathsf{H}_k : \mathcal{X} \subset \mathbb{F}_{p^2} \to \{0,1\}^w\}_{k\in K}$. Here $\mathcal{X}$ is the set of all supersingular j-invariants in $\mathbb{F}_{p^2}$, for which holds $|\mathcal{X}| = p/12 + \epsilon$, with $\epsilon \in \{0,1,2\}$ [DFJP14]. Furthermore, Encode (resp. Decode) is an efficiently computable bijective function from the message space (resp. $\{0,1\}^w$) to $\{0,1\}^w$ (resp. message space). The details of Encode and Decode are not significant for this section, since they are scheme specific.

The perfect correctness of the SIDH-based public-key encryption scheme from which our mPKE is deduced implies that the latter has δ-correctness, with $\delta = 0$. In addition, for a given security parameter 1^κ, the prime $p = 2^{e_2}3^{e_3} - 1$ in the public parameter $\mathsf{pp} \leftarrow \mathsf{mGen}(1^\kappa)$ is fixed [JAC+19]. The first component of each element in $\mathcal{C}_{\mathsf{single}}$ contains a curve 2^{e_2}-isogenous to E. We denote by W the set $\{j(E/\langle P_2 + [r]Q_2\rangle) | r \in \mathbb{Z}_{2^{e_2}}\}$ of all such curves. Since $p/12 + \epsilon \gg |W|$, one expects that the number of pairs of distinct coefficients $r, \tilde{r} \in \mathbb{Z}_{2^{e_2}}$ such that $j(E/\langle P_2 + [r]Q_2\rangle) = j(E/\langle P_2 + [\tilde{r}]Q_2\rangle)$ is very small [ACC+19a]. Hence, we can assume that $|W| = 2^{e_2}$ and deduce $\gamma(\mathsf{pp}, \mathsf{pk}) \geq e_2$. This value is independent of the public key pk and E, B_2, B_3 in pp,therefore the mPKE scheme has γ-spreadness with $\gamma = e_2$. We observe that $1/2^{e_2} \approx 1/\sqrt{p}$, which is negligible in the security parameter κ ($e_2 \geq \kappa$ for any set of SIDH parameters [JAC+19]).

IND-CPA *Security.* The IND-CPA security of the SIDH-based mPKE follows from the SSDDH assumption and the Leftover Hash Lemma (see Section 2.3).

Lemma 5.3. *Assume* mPKE *as shown in Figure 6. Then, for any (classical or quantum)* IND-CPA *adversary* $\mathcal{A}$*, there exists a (classical or quantum) adversary* $\mathcal{B}$ *such that*

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

Proof. 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 6: SIDH-based mPKE via hashed ElGamal [DFJP14]. mExt with input index i is defined by picking the relevant components $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ from $\vec{\mathsf{ct}}$. Note that $\mathsf{mEnc}^{\mathsf{d}}$ does not require any randomness r_i for $i \in [N]$.

- Game_0: This is the real $\mathsf{IND\text{-}CPA}$ security game. In particular, $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{A}) := |\Pr[\mathsf{E}_0] - 1/2|$.

- $\mathsf{Game}_{1,j}$ for $j \in [0, N]$: In this game, we modify the challenger so that it sets $\widehat{\mathsf{ct}}^*_i := \mathsf{Enc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}^*; r_0)$ for $i \in [N-j]$ and $\widehat{\mathsf{ct}}^*_i \leftarrow \mathsf{H}(J'_i) \oplus \mathsf{Encode}(\mathsf{M})$ 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 $\vec{\mathsf{ct}}^* := (\mathsf{ct}^*_0, \widehat{\mathsf{ct}}^*_1, \cdots, \widehat{\mathsf{ct}}^*_N)$ as the challenge ciphertext. We note that $\mathsf{Game}_{1,0}$ corresponds to Game_0. Each game is indistinguishable from the previous one under the $\mathsf{SSDDH}_{p,E,B_2,B_3}$ assumption. In

particular, we can construct an adversary $\mathcal{B}$ with around the same running time as $\mathcal{A}$ such that

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

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

- Game_2: In this game, the challenger is modified so that it sets $\widehat{\mathsf{ct}}^*_i \leftarrow h'_i \oplus \mathsf{Encode}(\mathsf{M})$ 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 (Lemma 2.6). 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 [DFJP14], and hence $\mathbf{H}_\infty(X) \approx \log_2 p$. Consequently, invoking the Leftover Hash Lemma N times, we have

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

Finally, since the challenge ciphertexts are distributed uniformly random for both challenge bit $b \in \{0,1\}$, we have $\Pr[\mathsf{E}_2] = 1/2$. This concludes the proof. □

Remark 5.4. We note that in concrete instantiations, $\log_2 p$ assumes the values $434, 503, 610$, while the corresponding w is 128,192 or 256, respectively [DFJP14]. Therefore we have $(1/2)\sqrt{2^w/p} \leq 2^{152}$ for each pair (p, w) and it can be safely discarded in the right term of Equation (2).

IND-Anon-CPA *Security.* The above proof for IND-CPA security is also a proof for IND-Anon-CPA. Indeed, in the final game we no longer require the users public keys $(\mathsf{pk}_i)_{i\in[N]}$ to simulate the challenge ciphertext. In particular, the fake encryption algorithm $\overline{\mathsf{mEnc}}$ simply outputs a tuple composed by a ciphertext ct_0 and N uniformly random elements in $\{0,1\}^w$.

Isogeny-based mPKE via CSIDH. The mPKE deduced from CSIDH is provided in Figure 7. In mSetup a prime p of the form $4\ell_1\ell_2\cdots\ell_t - 1$, where $\ell_1, \ldots, \ell_t$ are small odd primes, is chosen. Then the public parameter pp output by the algorithm consists of: a generator $\mathfrak{g}$ of the cyclic ideal class group $\mathcal{C}\ell(\mathcal{O})$, 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 $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$; a hash-function H uniformly sampled from a family of universal hash function $\mathcal{H} = \{\mathsf{H}_k : \mathcal{E}\ell\ell_p(\mathcal{O}, \pi) \to \{0,1\}^w\}_{k\in K}$ 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 scheme 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 $\mathsf{mEnc}^{\mathsf{i}}$ coincides with the uniform distribution over $\mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$ (it is induced by the free and transitive group action $\star$). Therefore $\gamma(\mathsf{pp}, \mathsf{pk}) \geq (\log_2 p)/2$ for all $\mathsf{pp} \in \mathsf{Setup}(1^\kappa)$ and $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{pp})$, and hence $\gamma = (\log_2 p)/2$.

IND-CPA *Security.* Analogously to the SIDH-based mPKE, the IND-CPA security of the scheme in Figure 7 follows from the dCSIDH assumption (Section 2.2) and the Leftover Hash Lemma (Section 2.3). In particular we have:

Lemma 5.5. *Assume* mPKE *as shown in Figure 7. Then, for any (classical or quantum)* IND-CPA *adversary* $\mathcal{A}$*, there exist (classical or quantum) adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{A}) \leq N \cdot \left(\mathsf{Adv}^{\mathsf{dCSIDH}}_{p,\mathfrak{g}}(\mathcal{B}) + \frac{1}{2}\sqrt{2^w/\sqrt{p}}\right). \tag{3}$$

The proof of Lemma 5.5 is just an adaptation of that of Lemma 5.3. To be precise, a sequence of hybrid games $E_{1,j}$, with $j \in [0, N]$, replaces H'_i with a uniformly random curve $\tilde{H}'_i \in \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$, and so

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

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

1: Select p and $\mathfrak{g}$
2: $n \leftarrow \mathrm{Order}(\mathfrak{g}), \mathsf{H} \leftarrow \mathcal{H}$
3: $E \leftarrow \mathcal{E}\ell\ell_p(\mathcal{O}, \pi)$
4: **return** $\mathsf{pp} := (\mathfrak{g}, n, E, \mathsf{H})$

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

Input: Public parameter $\mathsf{pp} = (\mathfrak{g}, n, E, \mathsf{H})$
Output: Public key pk, secret key sk

1: $a \leftarrow \mathbb{Z}_n$
2: $H \leftarrow \mathfrak{g}^a \star E$
3: **return** $\mathsf{pk} := H, \mathsf{sk} := a$

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

Input: Public parameter $\mathsf{pp} = (\mathfrak{g}, n, E, \mathsf{H})$, set of public keys $(\mathsf{pk}_i = H_i)_{i\in[N]}$, message M
Output: Ciphertext $\vec{\mathsf{ct}} = (\mathsf{ct}_0, (\widehat{\mathsf{ct}}_i)_{i\in[N]})$

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

Algorithm 16 $\mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0)$

Input: Public parameter $\mathsf{pp} = (\mathfrak{g}, n, E, \mathsf{H})$, public key $\mathsf{pk}_i = H_i$, message M, randomness $\mathsf{r}_0 = r$
Output: (Public key dependent) ciphertext $\widehat{\mathsf{ct}}_i$

1: $H_i' \leftarrow \mathfrak{g}^r \star H_i$
2: $F_i \leftarrow \mathsf{H}(H_i') \oplus \mathsf{Encode}(\mathsf{M})$
3: **return** $\widehat{\mathsf{ct}}_i := F_i$

Algorithm 17 $\mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0)$

Input: Public parameter $\mathsf{pp} = (\mathfrak{g}, n, E, \mathsf{H})$, randomness $\mathsf{r}_0 = r$
Output: (Public key independent) ciphertext ct_0

1: $E' \leftarrow \mathfrak{g}^r \star E$
2: **return** $\mathsf{ct}_0 := E'$

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

Input: Public parameter $\mathsf{pp} = (\mathfrak{g}, n, E, \mathsf{H})$, Secret key $\mathsf{sk} = a$, ciphertext $\mathsf{ct} = (E', F)$
Output: Message M

1: $H' \leftarrow \mathfrak{g}^a \star E'$
2: $\mathbf{M} \leftarrow F \oplus \mathsf{H}(H')$
3: **return** $\mathsf{M} := \mathsf{Decode}(\mathbf{M})$

Figure 7: CSIDH-based mPKE via hashed ElGamal [DFJP14]. mExt with input index i is defined in the obvious way by picking the relevant component $(\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ from $\vec{\mathsf{ct}}$. Note that $\mathsf{mEnc}^{\mathsf{d}}$ does not require any randomness r_i for $i \in [N]$ (except that used to compute the action $\star$).

$|\Pr[\mathsf{E}_{1,j}] - \Pr[\mathsf{E}_{1,j+1}]| \leq \mathsf{Adv}^{\mathsf{dCSIDH}}_{p,\mathfrak{g}}(\mathcal{B}_1)$ 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 Equation (3) the second term on the right can be safely discarded. Finally, we observe that the above-sketched proof also implies $\mathsf{IND\text{-}Anon\text{-}CPA}$, with $\overline{\mathsf{mEnc}}$ that outputs a tuple composed by a ciphertext ct_0 and N uniformly random elements in $\{0,1\}^w$.

Remark 5.6 (CSIDH with ideal class group of unknown structure). The scheme described in Figure 7 can be easily adapted to the case where the structure of the ideal class group $G = \mathcal{C}\ell(\mathcal{O})$ 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.

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 Section 5) and then into mKEMs via our generic transformation (Theorems 4.1 and 4.2). We note that we did not consider the corresponding mKEM for the CISDH mPKE, for

reasons explained later. We compare these mKEMs to the trivial solution that uses (single-recipient) KEMs in parallel, and show that our mKEMs provide efficiency gains, both in communication and computation, of an order of magnitude.

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

6.1 Comparison Methodology

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

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

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

For (C1), we measure the ratio:

$$\frac{\text{Data broadcast when using } N \text{ instances of the original KEM}}{\text{Data broadcast when using mKEM with } N \text{ recipients}}. \tag{4}$$

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

$$\frac{N|\mathsf{ct}_0| + N|\widehat{\mathsf{ct}}_i|}{|\mathsf{ct}_0| + N|\widehat{\mathsf{ct}}_i|} \quad \underset{N\to\infty}{\longrightarrow} \quad 1 + \frac{|\mathsf{ct}_0|}{|\widehat{\mathsf{ct}}_i|}. \tag{5}$$

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

$$k_{\mathsf{cycles}} = \lim_{N\to\infty} \frac{\text{Cycles spent to run } N \text{ instances of the original KEM}}{\text{Cycles spent to run mKEM with } N \text{ recipients}} \tag{6}$$

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 Section 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 [NIS19]. 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 Figure 5, since the addition of noise error is replaced in some instances by rounding. See

Remark 5.2 for a short discussion on this change. Finally, ThreeBears is based on an extremely recent variant called Module Integer-LWE. In addition, each scheme has different parameters and uses different tweaks. A widespread trick is for $\widehat{\mathsf{ct}}_i$ to drop the least significant bits of $\mathbf{V}_i$, since the message M is encoded in the most significant bits. This reduces the size of a (multi-)ciphertext. Note that bit dropping is more beneficial to mKEMs than to KEMs as it reduces $|\widehat{\mathsf{ct}}_i|$, hence a larger bandwidth impact for mKEMs – see (5).

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 Figure 5 (IND-CPA mPKE) and Figure 3 (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 Figure 5 (see Remark 6.1 below). Second, the transform of Figure 3 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 Remark 6.1). For our mKEMs, we do not claim to achieve any security notion besides those proven in this document.

Remark 6.1 (Reusing the public matrix). A difference between Figure 5 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* [ADPS16, 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 Figure 5. 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.

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

Computational costs.

Due to time constraints, we only implemented mKEM on two lattice-based schemes: FrodoKEM and Kyber. Nevertheless, we believe these examples already showcase the efficiency gain provided by our techniques. Starting from reference implementations available on Github[7][8], we tweaked them to obtain mKEMs. As shown by Table 2, our mKEM variants perform (multi-)encapsulation between one and two orders of magnitude faster than their original KEM counterparts. Additional experiments in Appendix D 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 Figure 6, and we compare it with the trivial solution consisting in N instances of SIKE. For CSIDH, we compare the mPKE of Figure 7 with N instances of the CSIDH-based hashed ElGamal. Since CSIDH is a key-exchange, we simply construct a trivial IND-CPA secure PKE from it (rather than constructing an IND-CCA secure KEM) and compare it with our mPKE from Section 5.2 (see also

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

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

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

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

Scheme	$\lvert\mathsf{ct}_0\rvert$	$\lvert\widehat{\mathsf{ct}}_i\rvert$	$\lvert\mathsf{ct}\rvert$	k_{comm}
FrodoKEM-640	9 600	120	9 720	**81**
FrodoKEM-976	15 616	128	15 744	**123**
FrodoKEM-1344	21 504	128	21 632	**169**
Kyber-512	640	96	736	**7.67**
Kyber-768	960	128	1 088	**8.5**
Kyber-1024	1 408	160	1 568	**9.8**
LAC-128	512	200	712	**3.56**
LAC-192	1024	164	1188	**7.24**
LAC-256	1024	400	1424	**3.56**
NewHope-512-CCA-KEM[6]	896	192	1 120	**5.83**
NewHope-1048-CCA-KEM	1 792	384	2 208	**5.75**
Round5 `R5ND_1KEMb`	429	110	539	**4.9**
Round5 `R5ND_3KEMb`	756	74	830	**11.22**
Round5 `R5ND_5KEMb`	940	142	1 082	**7.62**
LightSaber	640	96	736	**7.67**
Saber	960	128	1 088	**8.5**
FireSaber	1 280	192	1 472	**7.67**
BabyBear	780	137	917	**6.69**
MamaBear	1 170	137	1 307	**9.54**
PapaBear	1 560	137	1 697	**12.38**

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

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

Figure 7 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 library[9].

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.

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

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

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

Computational costs.

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

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

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

7 Application to Secure Group Messaging

In this section, we show how our mKEM can be used to optimize the *TreeKEM* protocol [BBR18, ACDT19, ACC+19b] used within secure group messagings. The resulting protocol has a lower communication cost than the standard version of TreeKEM [BBR18, ACDT19].

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. [BBR18, ACDT19, ACC+19b, BBM+20].

Continuous group key agreement (CGKA), which generalizes the notion of continuous key agreement (CKA, see [ACD19]), 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 7.1 (Continuous Group Key Agreement. [ACDT19])**.** *A continuous group key agreement* $\mathsf{CGKA} = (\mathsf{Init}, \mathsf{Create}, \mathsf{Add}, \mathsf{Remove}, \mathsf{Update}, \mathsf{Process})$ *consists of the following algorithms:*

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

Above, Update allows a user to update the session key on behalf of the whole group (it is run on every epoch to maintain strong security), and $\mathsf{Process}$ allows each group member to process the updated session key. Four properties are required from a CGKA: correctness, privacy, forward privacy (FS), and post-compromise security (PCS). At a high level, FS states that if any group member is compromised at some point, then all previous session keys remain hidden from the attacker; and PCS states that after every compromised group member performs an update, the session key becomes secret again. As the precise definitions are not relevant to our work, we refer to [ACDT19, 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 [BBM+20]. It was first described in [BBR18] and various improvements have been proposed in [ACDT19, ACC+19b]. TreeKEM is at the heart of the MLS protocol [BBM+20], 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 is one of the costliest parts (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.

7.1.1 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 Figure 8, "1" denotes the bottom left node.

Let u be a node in a tree T. Its siblings, $\mathsf{siblings}(u)$, is the set of nodes $v \neq u$ in T with the same parent as u. Its *path*, $\mathsf{path}(u)$, is the set of nodes between u and root, including u but excluding root. Its *co-path*, $\mathsf{copath}(u)$, is the set of siblings of nodes in its path: $\mathsf{copath}(u) = \bigcup_{v\in\mathsf{path}(u)} \mathsf{siblings}(v)$. For example, in Figure 8, 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 ().

7.1.2 TreeKEM.

In TreeKEM, a (binary or m-ary) tree T is constructed with the N group members as its leaves. As an example, Figure 8 illustrates the tree T associated to a group of 16 users (numbered from 1 to 16). Let PRG be a pseudorandom generator. Then, to each node i is associated a secret seed seed_i and a keypair $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{mGen}(\mathsf{pp}; \mathsf{PRG}(\mathsf{seed}_i)_\mathsf{L})$, where $\mathsf{PRG}(\cdot)_\mathsf{L}$ (resp. $\mathsf{PRG}(\cdot)_\mathsf{R}$) denotes the left (resp. right) half output of the PRG. In particular, mGen is run on randomness $\mathsf{PRG}(\mathsf{seed}_i)_\mathsf{L}$. The root does not need a keypair, but its seed will in effect be the group secret I (i.e., session key). The *TreeKEM invariant* states that a group member u knows seed_i if and only if $i \in \mathsf{path}(u)$. When a user u performs an update (via Update), he does the following:

(U1) Generate a new secret seed seed_u for u.

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

(U3) For each $i \in \mathsf{path}(u)$, compute the ciphertext

$$\vec{\mathsf{ct}}_i \leftarrow \mathsf{mEncaps}(\mathsf{pp}, (\mathsf{pk}_j)_{j \in \mathsf{siblings}(i)}; \mathsf{seed}_{\mathsf{parent}(i)}). \tag{7}$$

Note that $\mathsf{mEncaps}$ is derandomized here. For our construction in Figure 3, this is equivalent to setting the random message $\mathsf{M}_i = \mathsf{PRG}(\mathsf{seed}_{\mathsf{parent}(i)})$.

(U4) Send the update package $(\mathsf{pk}_i, \vec{\mathsf{ct}}_i)_{i \in \mathsf{path}(u)}$ to the server, which dispatches it to the other group members (this is known as *server-side fan-out*).

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

(P1) Update each pk_i he received.

(P2) Compute the closest common ancestor w of u and v, then recover seed_w by decapsulating the adequate $\vec{\mathsf{ct}}_i$.

(P3) Recover the secret seeds of all remaining common ancestors of u and v by computing $\mathsf{seed}_{\mathsf{parent}(i)} = \mathsf{PRG}(\mathsf{seed}_i)_{\mathsf{R}}$. The update secret is $I = \mathsf{seed}_{\mathsf{root}}$

This description is more generic than previous ones [BBR18, ACDT19, ACC+19b, BBM+20] 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. Figure 8 illustrates the "classical" instantiation of TreeKEM. Each update contains at most $\lceil \log_2(N) \rceil$ public keys and as many ciphertexts, so its bytesize is at most:

$$\lceil \log_2(N) \rceil \cdot \left(|\mathsf{pk}| + |\mathsf{ct}_0| + |\widehat{\mathsf{ct}}_i| \right) \tag{8}$$

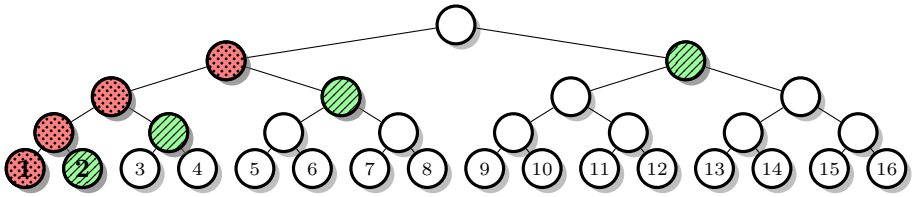


Figure 8: TreeKEM

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 [BBR18], TreeKEM can be instantiated with an m-ary tree instead of binary; see Figure 9 for an example where "1" issues a package update. At first, it is not obvious that this is more efficient than the instantiation of Figure 8, 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).

We make the following observation: when a user u issues an update, the update package may encapsulate several times the same information. Precisely, for each $i \in \mathsf{path}(u)$, the update package encapsulates $\mathsf{seed}_{\mathsf{parent}(i)}$ under the key pk_j for each $j \in \mathsf{siblings}(i)$. In the example of Figure 9, this means that an update package issued by 1 encapsulates seed_A under $\mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4$, and $\mathsf{seed}_{\mathsf{root}}$ under $\mathsf{pk}_B, \mathsf{pk}_C, \mathsf{pk}_D$. The bandwidth gain happens exactly here: since the same value seed_A is encapsulated under $\mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4$, one can use mKEM to perform this (multi-)encapsulation. And similarly at each level of the tree. Hence the total size of an update package is at most:

$$\lceil \log_m(N) \rceil \cdot \left(|\mathsf{pk}| + |\mathsf{ct}_0| + (m-1) \cdot |\widehat{\mathsf{ct}}_i| \right). \tag{9}$$

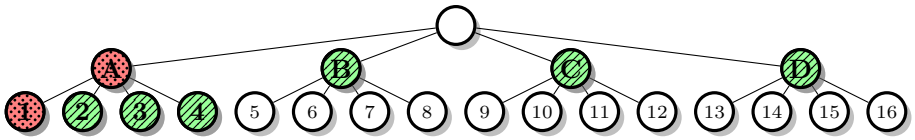


Figure 9: 4-ary TreeKEM

One can see that (9) generalizes (8) to any integer $m > 2$. It is clear from (9) that whenever $|\mathsf{pk}|+|\mathsf{ct}_0| \gg |\widehat{\mathsf{ct}}_i|$, it is advantageous efficiency-wise to take $m > 2$. This is illustrated in the next section.

7.2 Concrete Instantiations of m-ary TreeKEM

We now illustrate the substantial communication gains that can be obtained in practice with the method described above. A good rule of thumb is to take $m - 1 \approx \frac{|\mathsf{pk}|+|\mathsf{ct}_0|}{|\widehat{\mathsf{ct}}_i|}$. According to (8), the bytesize of an update package for binary TreeKEM will then be approximately $\lceil \log_2(N) \rceil \cdot m \cdot |\widehat{\mathsf{ct}}_i|$. On the other hand, the bytesize – given by (9) – for our proposal is about $\lceil \log_m(N) \rceil \cdot 2(m-1) \cdot |\widehat{\mathsf{ct}}_i|$. Compared to the standard TreeKEM, our proposal improves communication cost by a factor equal to the ratio of the two values, which is approximately:

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

Our solution provides a gain $O(\log m)$ compared to TreeKEM. A concrete comparison is provided by Figure 10, which compares the bytesize of an update package for binary TreeKEM - using FrodoKEM, Kyber, SIKE or cSIDH as a (single-recipient) KEM/PKE - and m-ary TreeKEM - using the mKEM/mPKE obtained from FrodoKEM, Kyber, SIKE or cSIDH, respectively. For the schemes considered, our proposal improves the communication cost for large groups by a factor between 1.8 and 4.2.

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

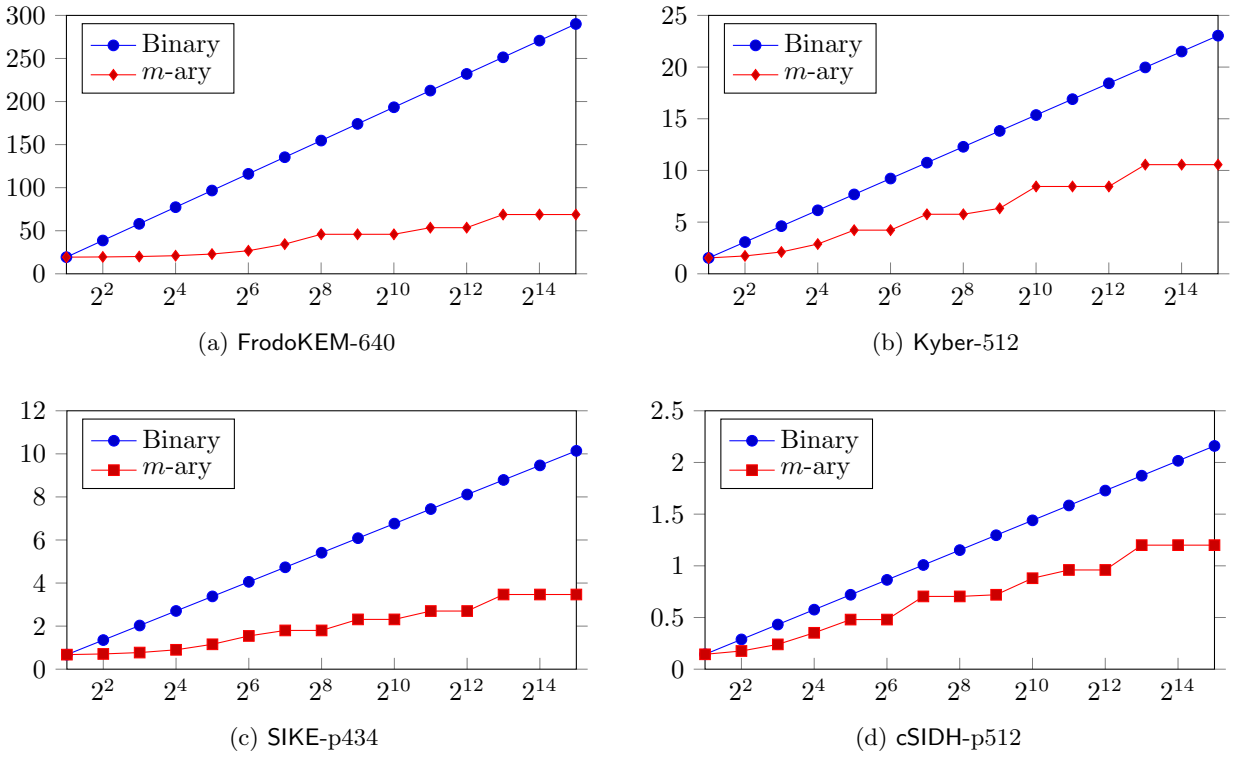
References

[ACC+19a] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018*, volume 11349 of *LNCS*, pages 322–343. Springer, Heidelberg, August 2019.

[ACC+19b] Joel Alwen, Margarita Capretto, Miguel Cueto, Chethan Kamath, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. Keep the dirt: Tainted treekem, an efficient and provably secure continuous group key agreement protocol. Cryptology ePrint Archive, Report 2019/1489, 2019. https://eprint.iacr.org/2019/1489.

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.

[ACDT19] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. Cryptology ePrint Archive, Report 2019/1189, 2019. https://eprint.iacr.org/2019/1189.



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

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

Figure 10: 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.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[AHU19] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019*, volume 11693 of *LNCS*, pages 269–295. Springer, Heidelberg, August 2019.

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

[BBM+20] Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-09, Internet Engineering Task Force, March 2020. Work in Progress.

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, May 2018.

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

[BCR+] Elaine Barker, Lily Chen, Allen Roginsky, Miles Smid, Elaine Barker, Lily Chen, Allen Roginsky, and Miles 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.

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

[BHH+19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In *TCC 2019*, LNCS, pages 61–90. Springer, Heidelberg, March 2019.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Heidelberg, April 2012.

[BPS00] Olivier Baudron, David Pointcheval, and Jacques Stern. Extended notions of security for multicast public key cryptosystems. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP 2000*, volume 1853 of *LNCS*, pages 499–511. Springer, Heidelberg, July 2000.

[CHJ+02] Jean-Sébastien Coron, Helena Handschuh, Marc Joye, Pascal Paillier, David Pointcheval, and Christophe Tymen. GEM: A generic chosen-ciphertext secure encryption method. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 263–276. Springer, Heidelberg, February 2002.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018.

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

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *TCC 2019*, LNCS, pages 1–29. Springer, Heidelberg, March 2019.

[CMSZ19] Jan Czajkowski, Christian Majenz, Christian Schaffner, and Sebastian Zur. Quantum lazy sampling and game-playing proofs for quantum indifferentiability. Cryptology ePrint Archive, Report 2019/428, 2019. https://eprint.iacr.org/2019/428.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CS20] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2018*, pages 169–186. Springer, Heidelberg, 2020.

[Den03] Alexander W Dent. A designer's guide to kems. In *IMA International Conference on Cryptography and Coding*, pages 133–151. Springer, 2003.

[DFJP14] Luca De Feo, David Jao, and Jerome Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Journal of Mathematical Cryptology*, volume 8 (3), pages 209–247, 2014.

[EKP20] Ali El Kaafarani, Shuichi Katsumata, and Federico Pintore. Lossy CSI-FiSh: Efficient signature scheme with tight reduction to decisional CSIDH-512. In *PKC 2020*, LNCS, pages 157–186. Springer, Heidelberg, 2020.

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

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

[HI19] Akinori Hosoyamada and Tetsu Iwata. 4-round Luby-Rackoff construction is a qPRP. In *ASIACRYPT 2019*, LNCS, pages 145–174. Springer, Heidelberg, December 2019.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. Construction of a pseudo-random generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

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

[HKSU20] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. In *PKC 2020*, pages 389–422, 2020.

[HTAS09] Harunaga Hiwatari, Keisuke Tanaka, Tomoyuki Asano, and Koichi Sakumoto. Multi-recipient public-key encryption from simulators in security proofs. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP 09*, volume 5594 of *LNCS*, pages 293–308. Springer, Heidelberg, July 2009.

[JAC+19] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, and Geovandro Pereira. SIKE. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[JZC+18] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018*, volume 10993 of *LNCS*, pages 96–125. Springer, Heidelberg, August 2018.

[JZM19a] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Key encapsulation mechanism with explicit rejection in the quantum random oracle model. In Dongdai Lin and Kazue Sako, editors, *PKC 2019*, volume 11443 of *LNCS*, pages 618–645. Springer, Heidelberg, April 2019.

[JZM19b] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. In *PQCRYPTO 2019*, pages 227–248, 2019.

[KSS+] Veronika Kuchta, Amin Sakzad, Damien Stehlé, Ron Steinfeld, and Shi-Feng Sun. Measure-rewind-measure: tighter quantum random oracle model proofs for one-way to hiding and cca security. In *EUROCRYPT 2020*, pages 703–728. Springer.

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

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

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

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019*, volume 11693 of *LNCS*, pages 326–355. Springer, Heidelberg, August 2019.

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

[NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[NIS19] NIST. Post-quantum cryptography - round 2 submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2019.

[OBR+20] Emad Omara, Benjamin Beurdouche, Eric Rescorla, Srinivas Inguva, Albert Kwon, and Alan Duric. The Messaging Layer Security (MLS) Architecture. Internet-Draft draft-ietf-mls-architecture-04, Internet Engineering Task Force, January 2020. Work in Progress.

[OP01] Tatsuaki Okamoto and David Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 159–175. Springer, Heidelberg, April 2001.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

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

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

[TU16] Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B*, volume 9986 of *LNCS*, pages 192–216. Springer, Heidelberg, October / November 2016.

[Unr14] Dominique Unruh. Revocable quantum timed-release encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 129–146. Springer, Heidelberg, May 2014.

[Yan15] Zheng Yang. On constructing practical multi-recipient key-encapsulation with short ciphertext and public key. *SCN*, 8(18):4191–4202, 2015.

[Zha19] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019*, volume 11693 of *LNCS*, pages 239–268. Springer, Heidelberg, August 2019.

A Simple Attack on LPN-based mKEM by [CLQY18]

We are aware of one recent work by Cheng et al., [CLQY18] that constructs a post-quantum IND-CCA secure mKEM from the LPN assumption. As with any practical IND-CCA secure encryption scheme in the ROM, they rely on the Fujisaki-Okamoto transform. However, on closer inspection, it is clear that they misuse the Fujisaki-Okamoto transform in the multi-recipient setting, and in particular, their scheme admits a trivial attack.

Similarly to our LWE-based mKEM, they require randomness that is specific to each users. In the context of mPKE, this means r_i in mEnc is not an empty string (see Definition 3.1). However, when they apply the Fujisaki-Okamoto transform, they only generate the randomness used by all the users via H(M) and samples the rest of the user specific randomness on its own. Consequently, since the ciphertext cannot be reencrypted back from the message, there is no way to validate the user specific part of the ciphertext. Concretely, in their construction, decryption will succeed even if we add a slight noise in the user specific part of the ciphertext. Therefore, as an adversary, we can simply add a small noise to the challenge ciphertext to recover the challenge message (or sessions key) to break IND-CCA security.

B Proof of Theorem 4.2

We provide the proof of Theorem 4.2 (post-quantum IND-CCA security of our mKEM) in this section. Minimal background on quantum computation is provided in Appendix C, and we refer for more details to other works such as [CMSZ19, AHU19, Zha19, HKSU20].

Proof of Theorem 4.2. 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.

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

- Game_2: In this game, we replace the computation of $\mathsf{H}'(\mathsf{seed}_i, \cdot)$ by a random function $\widehat{\mathsf{H}}'_i(\cdot)$ in case $\mathsf{M} = \bot$ or $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ occurs when answering the decapsulation oracle with input $i \in [N]$. Due to Lemma C.1, and a standard hybrid argument, we have

$$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| \leq q'_{\mathsf{H}} \cdot N \cdot 2^{\frac{-\ell+1}{2}}.$$

Using Compressed Oracles. We first make a simplifying argument that $\mathcal{A}$ queries the same message to all of the oracles G_1, G_2, and H, and that G_2 is only queried on $\mathsf{pk}_i \in \{\mathsf{pk}_1, \cdots, \mathsf{pk}_N\}$. That is, when $\mathcal{A}$ queries for an input M to the oracles, it receives back $(\mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_1, \mathsf{M}), \cdots, \mathsf{G}_2(\mathsf{pk}_N, \mathsf{M}), \mathsf{H}(\mathsf{M}))$. It is clear that this modification does not weaken $\mathcal{A}$ since querying G_2 on other pk provides $\mathcal{A}$ no additional advantage. Here, we assume $\mathcal{A}$ obtains $(N+2)$ results by making one query. Moreover, we can always transform an adversary $\mathcal{A}$ that does not query all the oracles on the same input to an adversary that does.

Next, throughout the security game, we assume the oracles $\mathsf{G}_1(\cdot)$, $\mathsf{G}_2(\mathsf{pk}_1, \cdot), \cdots, \mathsf{G}_2(\mathsf{pk}_N, \cdot)$, and $\mathsf{H}(\cdot)$ to be simulated by a single oracle $\widehat{\mathsf{G}}(\cdot)$. In particular, we define $\mathsf{G}_1(\mathsf{M}) := \widehat{\mathsf{G}}(\mathsf{npk}, \mathsf{M})$, $\mathsf{G}_2(\mathsf{M}) := \widehat{\mathsf{G}}(\mathsf{pk}_i, \mathsf{M})$ for $i \in [N]$, and $\mathsf{H}(\mathsf{M}) := \widehat{\mathsf{G}}(\mathsf{key}, \mathsf{M})$, where npk and key are special symbols distinct from all pk_i's. Namely, we implement the oracles from a single oracle by using appropriate domain separation. Finally, we simulate the random oracle $\widehat{\mathsf{G}}$ by a compressed standard oracle CStO initialized by the empty database D [Zha19] (see Appendix C.1).

- Game_3: In this game, we add an additional check at the end of the game to see if a "bad" randomness was ever used. Define $\mathsf{aux}_i := (\mathsf{pp}, (\mathsf{pk}_i, \mathsf{sk}_i))$ for $i \in [N]$ and define the sets of bad randomness as

$$\mathcal{R}^{\mathsf{bad}}_i(\mathsf{aux}_i, \mathsf{M}) := \left\{ (\mathsf{r}_0, \mathsf{r}_i) \;\middle|\; \begin{array}{c} \mathsf{M} \neq \mathsf{mDec}(\mathsf{sk}^{\mathsf{p}}_i, (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)), \text{where} \\ \mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{r}_0), \widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{r}_0, \mathsf{r}_i) \end{array} \right\}.$$

In addition, define S to be the set of index-message pairs (excluding the pairs with message equal to $\bot$) that was obtained by decrypting the ciphertext when answering the decapsulation query (i, ct). Note that S only consists of classical elements since $\mathcal{A}$ only makes classical decapsulation queries. Then, after $\mathcal{A}$ outputs its guess at the end of the game, the challenger checks the database D (i.e., state of the compressed standard oracle) for a pair of tuples $((\mathsf{npk}, \mathsf{M}), \mathsf{r}_0)$ and $((\mathsf{pk}_i, \mathsf{M}), \mathsf{r}_i)$ for any $(i, \mathsf{M}) \in S$ and $(\mathsf{r}_0, \mathsf{r}_i) \in \mathcal{R}^{\mathsf{bad}}_i(\mathsf{aux}_i, \mathsf{M})$. We call the event $\mathsf{BAD}_{\mathsf{rand}}$ if such a set of tuples are 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 $\mathsf{BAD}_{\mathsf{rand}}$ occurs, we have $|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| \leq \Pr[\mathsf{BAD}_{\mathsf{rand}}]$. By Lemma C.2, we can upper bound $\Pr[\mathsf{BAD}_{\mathsf{rand}}]$ by $6 \cdot (q_{\mathsf{G}} + q_{\mathcal{D}} + 1)^2 \cdot \max_i \{ |\mathcal{R}^{\mathsf{bad}}_i(\mathsf{aux}_i, \mathsf{M})| / |\mathcal{R}| \} + 3 \cdot 2^{-\mu/2}$, where $\mathcal{R}$ is the randomness space of a single ciphertext and $\mu = |\mathsf{r}_0| + |\mathsf{r}|$ for any $(\mathsf{r}_0, \mathsf{r}) \in \mathcal{R}$. By definition, we have $\delta \geq \mathbb{E}\left[\max_{\mathsf{M} \in \mathcal{M}} |\mathcal{R}^{\mathsf{bad}}_i(\mathsf{aux}_i, \mathsf{M})| / |\mathcal{R}|\right]$, where the expectation is taken over the randomness used to sample $\mathsf{aux}_i = (\mathsf{pp}, (\mathsf{pk}_i, \mathsf{sk}_i))$ and δ is the correctness parameter of mPKE. Hence, we conclude

$$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| \leq 6 \cdot (q_{\mathsf{G}} + q_{\mathcal{D}} + 1)^2 \cdot \delta + 3 \cdot 2^{-\mu/2}.$$

(The next Game_4 through Game_8 aim to get rid of the secret keys sk_i to answer $\mathcal{A}$'s decapsulation oracle queries.)

- Game_4: In this game, we make a conceptual change in the way we answer the decapsulation oracle query. During responding to the decapsulation oracle query, if $\mathsf{M} \neq \bot$, we perform a test in superposition to check if whether $(\mathsf{npk}, \mathsf{M})$ and $(\mathsf{pk}_i, \mathsf{M})$ are in the database D. 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 (in superposition) which evaluates to 1 if and only if D includes $(\mathsf{npk}, \mathsf{M})$ and $(\mathsf{pk}_i, \mathsf{M})$, 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,

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

- Game_5: In this game, we alter the time on which we un-compute TEST_1. Namely, we perform TEST_1 after we finish responding to the decapsulation oracle query.

To make the statement more clear, we first recall how the decapsulation oracle query is implemented by unitary operations. Since we only care about the computation made after we check $\mathsf{M} \neq \bot$, we focus on how the check $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ is implemented. First note that checking $\mathsf{ct} \neq (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ is equivalent to checking $(\mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M})) \in \mathcal{R}_i(\mathsf{aux}_i, \mathsf{ct}, \mathsf{M})$, where $\mathcal{R}_i(\mathsf{aux}_i, \mathsf{ct}, \mathsf{M})$ is the subset of the randomness space $\mathcal{R}$ which maps an encryption of M to ct with respect to $\mathsf{pp}, \mathsf{pk}_i$. Namely, we check $((\mathsf{npk}, \mathsf{M}), \mathsf{r}_0)$ and $((\mathsf{pk}_i, \mathsf{M}), \mathsf{r}_i)$ such that $(\mathsf{r}_0, \mathsf{r}_i) \in \mathcal{R}_i(\mathsf{aux}_i, \mathsf{ct}, \mathsf{M})$ are included in the database D. Let the classical function which performs this check as TEST_2. Then, by the definition of compressed standard oracles, we first perform $\mathsf{StdDecomp}\,|\mathsf{M}\rangle\,|D\rangle$ (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 $\widehat{\mathsf{H}}'_i(\mathsf{ct})$ and otherwise compute $\mathsf{H}(\mathsf{M})$, where recall $\mathsf{H}(\mathsf{M})$ is also simulated by the compressed standard oracle. Finally, we perform $\mathsf{StdDecomp}\,|\mathsf{M}\rangle\,|D\rangle$ again to decompress the oracle.

To show that Game_4 and Game_5 are indistinguishable, we must show that performing $\mathsf{StdDecomp} \circ \mathsf{TEST}_2 \circ \mathsf{StdDecomp} \circ \mathsf{TEST}_1 \circ \mathsf{TEST}_1$ is indistinguishable from performing $\mathsf{TEST}_1 \circ \mathsf{StdDecomp} \circ \mathsf{TEST}_2 \circ \mathsf{StdDecomp} \circ \mathsf{TEST}_1$, where the former (resp. latter) corresponds to Game_4 (resp. Game_5). Since both computation starts with applying TEST_1, we ignore them below. We also ignore the computation of $\widehat{\mathsf{H}}'_i$ and H since they commute with TEST_1 by noticing that they work on different registries. Notice that in case $(\mathsf{npk}, \mathsf{M})$ and $(\mathsf{pk}_i, \mathsf{M})$ were already in the database D, the corresponding output registry cannot be $\bot$ (i.e., $((\mathsf{npk}, \mathsf{M}), \bot)$ and $((\mathsf{pk}_i, \mathsf{M}), \bot)$ cannot be in the database D), since each query to CStO ends by removing such entries by definition. Therefore, $\mathsf{StdDecomp}$ is the same as an identity operation for such inputs. Now, in case $(\mathsf{npk}, \mathsf{M})$ or $(\mathsf{pk}_i, \mathsf{M})$ are not in the database D, $\mathsf{StdDecomp}$ adds either $\frac{1}{\sqrt{2^{|\mathsf{r}_0|}}} \sum_{\mathsf{r}_0} |(\mathsf{npk}, \mathsf{M}), \mathsf{r}_0\rangle$ or $\frac{1}{\sqrt{2^{|\mathsf{r}_i|}}} \sum_{\mathsf{r}_i} |(\mathsf{pk}_i, \mathsf{M}), \mathsf{r}_i\rangle$, respectively, to the database D. Denote $\mathsf{TEST}_1^\dagger$ as the unitary operation that first applies the quantum Fourier transform to the registry corresponding to $|\mathsf{r}_0\rangle$ and $|\mathsf{r}_i\rangle$ and then outputs 1 if and only if both registries are not identical to 0. Then, from the above argument, we have $\mathsf{TEST}_1^\dagger \circ \mathsf{StdDecomp} = \mathsf{StdDecomp} \circ \mathsf{TEST}_1$. Now, since $\mathsf{TEST}_1^\dagger$ tests whether $(\mathsf{r}_0, \mathsf{r}_i)$ is equal to $(0, 0)$ in the Fourier basis and TEST_2 tests whether $(\mathsf{r}_0, \mathsf{r}_i) \in \mathcal{R}_i(\mathsf{aux}_i, \mathsf{ct}, \mathsf{M})$ in the standard basis, we can invoke Lemma C.4 to argue that $\mathsf{TEST}_1^\dagger$ and TEST_2 "almost" commute. Namely, performing $\mathsf{TEST}_2 \circ \mathsf{TEST}_1^\dagger$ or $\mathsf{TEST}_1^\dagger \circ \mathsf{TEST}_2$ alters the distribution of the view of $\mathcal{A}$ by at most $8\sqrt{|\mathcal{R}_i(\mathsf{aux}_i, \mathsf{ct}, \mathsf{M})| \,/\, |\mathcal{R}|}$. Finally, using the same argument for swapping $\mathsf{TEST}_1^\dagger$ and $\mathsf{StdDecomp}$ as before, we conclude that performing $\mathsf{StdDecomp} \circ \mathsf{TEST}_2 \circ \mathsf{StdDecomp} \circ \mathsf{TEST}_1 \circ \mathsf{TEST}_1$ is indistinguishable from performing $\mathsf{TEST}_1 \circ \mathsf{StdDecomp} \circ \mathsf{TEST}_2 \circ \mathsf{StdDecomp} \circ \mathsf{TEST}_1$. In particular, by taking the expectation over the randomness used to sample $\mathsf{aux} = (\mathsf{pp}, (\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[N]})$, we conclude

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

where recall we have $2^{-\gamma} \geq \mathbb{E}\left[\max_{\mathsf{ct}\in\mathcal{C}_{\mathsf{single}}, \mathsf{M}\in\mathcal{M}} |\mathcal{R}(\mathsf{aux}, \mathsf{ct}, \mathsf{M})|/|\mathcal{R}|\right]$ due to γ-spreadness.

- Game_6: In this game, after we perform (the first) TEST_1, we perform an additional computation that takes as input the bit b, which is the output of TEST_1. Let $i \in [N]$ be the index queried to the decapsulation oracle. Recall $b = 1$, if $(\mathsf{npk}, \mathsf{M})$ and $(\mathsf{pk}_i, \mathsf{M})$ are in the database D and $b = 0$, otherwise. In this game, in case $b = 0$, we simply compute $\widehat{\mathsf{H}}'_i(\mathsf{ct})$ and return it to the adversary, ignoring the rest of the checks. Otherwise, it is the same as in Game_5. The only way for $\mathcal{A}$ to distinguish the two games is if it queries the decapsulation oracle on $(i, \mathsf{ct} \neq \mathsf{ct}_i^*)$ such that $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ and $\mathsf{M} := \mathsf{mDec}(\mathsf{sk}_i^{\mathsf{P}}, \mathsf{ct})$, where $(\mathsf{npk}, \mathsf{M})$ or $(\mathsf{pk}_i, \mathsf{M})$ are not in the database. Due to our assumption on how $\mathcal{A}$ queries the compressed standard oracle, if either one of the pair $(\mathsf{npk}, \mathsf{M})$ or $(\mathsf{pk}_i, \mathsf{M})$ is not found in the database, then the corresponding randomnesses are in uniform distribution with overwhelming probability in the view of the adversary (due to Lemma C.2). Therefore, by γ-spreadness the probability of $\mathcal{A}$ choosing ct such that $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}})$ is bounded by $2^{-\gamma}$. Hence, we have

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_6]| \leq q_{\mathcal{D}} \cdot 2^{-\gamma} + 3 \cdot 2^{-\mu/2},$$

where recall $\mu = |\mathsf{r}_0| + |\mathsf{r}|$ for any $(\mathsf{r}_0, \mathsf{r}) \in \mathcal{R}$.

- Game_7: In this game, we further modify the way we answer the decapsulation oracle query. Recall in the previous game, we computed $\mathsf{M} \leftarrow \mathsf{mDec}(\mathsf{sk}_i^{\mathsf{p}}, \mathsf{ct})$ and tested several times whether $(\mathsf{npk}, \mathsf{M})$ and $(\mathsf{pk}_i, \mathsf{M})$ are in the database to respond to the decapsulation query. In this game, to answer the decapsulation oracle query, the challenger instead just scans through all the inputs of the form $(\mathsf{npk}, \mathsf{M})$ and $(\mathsf{pk}_i, \mathsf{M})$ inside the database and checks if $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$, where $\mathsf{ct}_0, \widehat{\mathsf{ct}}_i$ are encrypted using the measured randomness. The challenger returns $\mathsf{H}(\mathsf{M})$ if it holds and $\widehat{\mathsf{H}}'_i(\mathsf{ct})$ otherwise.

We check that the two games are identical unless event $\mathsf{BAD}_{\mathsf{rand}}$ occurs. First, notice that if the decapsulation oracle in Game_6 outputs $\mathsf{K} := \mathsf{H}(\mathsf{M})$, then M was in the database and $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ holds. Therefore, the decapsulation oracle in Game_7 outputs the same K as well. On the other hand, assume the decapsulation oracle in Game_7 outputs $\mathsf{K} := \mathsf{H}(\mathsf{M})$ for some M in the dabase such that $\mathsf{ct} = (\mathsf{ct}_0, \widehat{\mathsf{ct}}_i)$ where $\mathsf{ct}_0 := \mathsf{mEnc}^{\mathsf{i}}(\mathsf{pp}; \mathsf{G}_1(\mathsf{M}))$ and $\widehat{\mathsf{ct}}_i := \mathsf{mEnc}^{\mathsf{d}}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{M}; \mathsf{G}_1(\mathsf{M}), \mathsf{G}_2(\mathsf{pk}_i, \mathsf{M}))$. Then, since we have no correctness error conditioning on $\mathsf{BAD}_{\mathsf{rand}}$ not occurring, ct must decrypt to M. Hence, this implies that the decapsulation oracle Game_6 outputs the same K as well. Finally, since when $\mathsf{BAD}_{\mathsf{rand}}$ occurs, we force the adversary to output a random bit in both games, we have

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

- Game_8: In this game, we undo the change we made in Game_3 and will no longer abort in case event $\mathsf{BAD}_{\mathsf{rand}}$ occurs. Due to the same argument as before, we have

$$|\Pr[\mathsf{E}_7] - \Pr[\mathsf{E}_8]| \leq 6 \cdot (q_{\mathsf{G}} + 1)^2 \cdot \delta + 3 \cdot 2^{-\mu/2}.$$

At this point, the challenger no longer requires the secret keys sk_i. Moreover, the challenger can be run efficiently since it does not need to check the event $\mathsf{BAD}_{\mathsf{rand}}$.

(The next Game_9 *and* Game_{10} *aim to get rid of* M^* *in the challenge ciphertext.)*

- Game_9: In this game, we sample all the randomness used to construct the challenge ciphertext at the beginning of the game and program the compressed standard oracle. That is, the challenger samples the random message $\mathsf{M}^* \leftarrow \mathcal{M}$, randomness $(\mathsf{r}_0^*, \mathsf{r}_1^*, \cdots, \mathsf{r}_N^*)$, and random key K^* that will be used to generate the challenge ciphertext $\vec{\mathsf{ct}}^*$ at the beginning of the game. Namely, the challenger sets $\vec{\mathsf{ct}}^* := \mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]}; \mathsf{r}_0^*, \mathsf{r}_1^*, \cdots, \mathsf{r}_N^*)$ and $\mathsf{K}_0^* := \mathsf{K}^*$. It then sets the compressed standard oracle CStO to an *almost* compressed standard oracle $\mathsf{CStO}[T]$, where $T = \{((\mathsf{npk}, \mathsf{M}^*), \mathsf{r}_0^*), (((\mathsf{pk}_i, \mathsf{M}^*), \mathsf{r}_i^*))_{i\in[N]}, ((\mathsf{key}, \mathsf{M}^*), \mathsf{K}^*)\}$. This has the effect of patching the randomness used to generate the challenge ciphertexts to be consistent with the oracle, e.g., $\widehat{\mathsf{G}}(\mathsf{npk}, \mathsf{M}^*) = \mathsf{G}_1(\mathsf{M}^*) = \mathsf{r}_0^*$. Since this does not alter the view of the adversary, we have

$$\Pr[\mathsf{E}_8] = \Pr[\mathsf{E}_9].$$

- Game_{10}: In this game, we further modify how the almost compressed standard oracle is set up. At the beginning of the game, the challenger further samples fresh randomness $(\widetilde{\mathsf{r}}_0^*, \widetilde{\mathsf{r}}_1^*, \cdots, \widetilde{\mathsf{r}}_N^*)$ and key $\widetilde{\mathsf{K}}^*$ independent from $(\mathsf{r}_0^*, \mathsf{r}_1^*, \cdots, \mathsf{r}_N^*)$ and K^* used to construct the challenge ciphertext. It then sets $\widetilde{T} = \{((\mathsf{npk}, \mathsf{M}^*), \widetilde{\mathsf{r}}_0^*), (((\mathsf{pk}_i, \mathsf{M}^*), \widetilde{\mathsf{r}}_i^*))_{i\in[N]}, ((\mathsf{key}, \mathsf{M}^*), \widetilde{\mathsf{K}}^*)\}$ and initializes the almost compressed standard oracle as $\mathsf{CStO}[\widetilde{T}]$. Observe that in this game, the adversary only obtains information on $(\mathsf{r}_0^*, \mathsf{r}_1^*, \cdots, \mathsf{r}_N^*)$ and K^* through the challenge ciphertext query. Therefore, since the keys $\mathsf{K}_0^* = \mathsf{K}^*$ and K_1^* are distributed uniformly random from $\mathcal{A}$'s view regardless of the challenge bit $b \in \{0, 1\}$, we have

$$\Pr[\mathsf{E}_{10}] = 1/2.$$

It remains to prove that Game_9 and Game_{10} are indistinguishable. Let $\mathcal{B}$ be an algorithm that simulates the interaction between the challenger and the adversary $\mathcal{A}$ given oracle access to an almost compressed standard oracle. More formally, $\mathcal{B}$ on input $z_b = (\mathsf{pp}, (\mathsf{pk})_{i\in[N]}, \vec{\mathsf{ct}}^*, \mathsf{K}_b^*)$ simulates the challenger using z_b conditioning on the challenge bit being $b \in \{0, 1\}$ and runs $\mathcal{A}$. $\mathcal{B}$ outputs whatever output by $\mathcal{A}$. In

case $\mathcal{B}$ has oracle access to $\mathsf{CStO}[T]$ (resp. $\mathsf{CStO}[\widetilde{T}]$) it corresponds to Game_9 (resp. Game_{10}). Then, by Theorem C.5, we have

$$\begin{aligned}|\Pr[\mathsf{E}_9] - \Pr[\mathsf{E}_{10}]| &= \sum_{b\in\{0,1\}} \frac{1}{2}\left|\Pr[1 \leftarrow \mathcal{B}^{\mathsf{CStO}[T]}(z_b)] - \Pr[1 \leftarrow B^{\mathsf{CStO}[\widetilde{T}]}(z_b)]\right| \\ &\leq \sum_{b\in\{0,1\}} \sqrt{(q_{\mathsf{G}}+1)\Pr[\mathsf{FIND} : \mathcal{B}^{\mathsf{CStO}[\widetilde{T}]\setminus S}(z_b)]}, \end{aligned} \tag{10}$$

where $S = \{(\mathsf{npk}, \mathsf{M}^*), ((\mathsf{pk}_i, \mathsf{M}^*)), (\mathsf{key}, \mathsf{M}^*)\}$. Observe that the randomness $(\mathsf{r}_0^*, \mathsf{r}_1^*, \cdots, \mathsf{r}_N^*)$ used to generate $\vec{\mathsf{ct}}^*$ in z_b for $b \in \{0,1\}$ is independent from $\widetilde{T}$. Therefore we can invoke the IND-CPA security of the underlying mPKE to change z_b to $\widetilde{z}_b$, where the only difference between z_b and $\widetilde{z}_b$ is that we swap $\vec{\mathsf{ct}}^*$ to $\vec{\widetilde{\mathsf{ct}}}^* = \mathsf{mEnc}(\mathsf{pp}, (\mathsf{pk}_i), 0; \mathsf{r}_0^*, \mathsf{r}_1^*, \cdots, \mathsf{r}_N^*)$. Hence, the probability inside Equation (10) can be further upper bounded by

$$\Pr[\mathsf{FIND} : \mathcal{B}^{\mathsf{CStO}[\widetilde{T}]\setminus S}(z_b)] \leq \Pr[\mathsf{FIND} : \mathcal{B}^{\mathsf{CStO}[\widetilde{T}]\setminus S}(\widetilde{z}_b)] + 2\cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}}),$$

where $\mathcal{B}_{\mathsf{IND}}$ is the IND-CPA adversary that is naturally induced by $\mathcal{B}$. Finally, since $\widetilde{T}$ and $\widetilde{z}_b$ are distributed independently except for the public $(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[N]})$, and M^* is distributed uniformly over $\mathcal{M}$, by Theorem C.5, we have

$$\Pr[\mathsf{FIND} : \mathcal{B}^{\mathsf{CStO}[\widetilde{T}]\setminus S}(\widetilde{z}_b)] \leq \frac{4(q_{\mathsf{G}}+1)}{|\mathcal{M}|}.$$

Collecting all the bounds, we have

$$|\Pr[\mathsf{E}_9] - \Pr[\mathsf{E}_{10}]| \leq 2\cdot\sqrt{2(q_{\mathsf{G}}+1)\cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{mPKE},N}(\mathcal{B}_{\mathsf{IND}})} + \frac{8(q_{\mathsf{G}}+1)}{\sqrt{\mathcal{M}}},$$

where we use the concavity of the square root function. This concludes the proof. □

C Background on Quantum Random Oracles

The main purpose of this section is to recall the *compressed oracle* techniques introduced in [Zha19]. 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 techniques 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 [NC02] for a thorough introduction. Our presentation of the digest of Zhandry's result follows closely to that made in [HI19]. We refer [Zha19] for the full detail and [HI19] for a more in-depth summary of [Zha19] (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 $(\mathcal{O}_1, \cdots, \mathcal{O}_q)$ 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 next 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 $U_0 \otimes I_k, \mathcal{O}_1, U_1 \otimes I_k, \cdots, U_q \otimes I_k, \mathcal{O}_q$ act sequentially on the initial state $|0^\ell\rangle \otimes |\mathsf{init}_{\mathcal{O}}\rangle$, where $|\mathsf{init}_{\mathcal{O}}\rangle$ is the initial state of $\mathcal{O}$. Finally, $\mathcal{A}$ measures its output register of the resulting quantum state $(U_q \otimes I_k)\mathcal{O}_q \cdots \mathcal{O}_1(U_0 \otimes I_k)\,|0^\ell\rangle \otimes |\mathsf{init}_{\mathcal{O}}\rangle$, 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.

C.1 Compressed Oracles and QROM

Standard Oracles. An oracle $\mathcal{O}_{\mathsf{H}}$ can be implemented with an encoding of a function $\mathsf{H} : \{0,1\}^m \to \{0,1\}^n$ 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 $|\mathsf{H}\rangle := |\mathsf{H}(0)\rangle\,|\mathsf{H}(1)\rangle\cdots|\mathsf{H}(2^m-1)\rangle$. The unitary operator StO is defined as

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

where $\alpha_x \in \{0,1\}^n$ for each $x \in [0, 2^{m-1}]$. In particular, we have $\mathsf{StO}\,|x\rangle\,|y\rangle\,|\mathsf{H}\rangle = |x\rangle\,|y\oplus\mathsf{H}(x)\rangle\,|\mathsf{H}\rangle$. 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 [Zha19] 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, $\sum_{\mathsf{H}} \frac{1}{\sqrt{2^{n2^m}}}|\mathsf{H}\rangle$. More concretely, the whole quantum state before $\mathcal{A}$ makes the $(i+1)$-th quantum query can be expressed by

$$|\phi_{i+1}\rangle = (U_i \otimes I_k)\mathsf{StO}\cdots\mathsf{StO}(U_0\otimes I_k)\left(|0^\ell\rangle \otimes \sum_{\mathsf{H}} \frac{1}{\sqrt{2^{n2^m}}}|\mathsf{H}\rangle\right). \tag{11}$$

This is in contrast to previous formalization of QROM where one (quantumly accessible) function H was randomly sampled and fixed throughout the security game. By considering the mixed state of the adversary $\mathcal{A}$, it can be seen 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 [Zha19] main observation is that we can consider a *compressed* version of the standard oracle which allows for such efficient simulation.

To get a better intuition, consider changing the basis of the α_i registers in Equation (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 " $\widehat{\ }$ ", Equation (11) is equivalent to

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

$$= \sum_{xyz\widehat{D}} a'_{xyz\widehat{D}}\,|xyz\rangle\otimes|\widehat{D}\rangle\,, \tag{13}$$

where $a'_{xyz\widehat{D}}$ denotes a complex number satisfying $\sum_{xyz\widehat{D}}\left|a'_{xyz\widehat{D}}\right|^2 = 1$, 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, if $\mathcal{A}$'s y-register is also expressed in the Fourier basis, then we have the following:

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

In particular, the StO operator adds at most one data in the oracle's register. Therefore, since $|\widehat{\alpha_x}\rangle \neq 0^n$ for at most i many x after the i-th query, $\widehat{D}$ 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 $\widehat{D}$ only has at most q entries that are non-zero after $\mathcal{A}$ queries the oracle, Zhandry's [Zha19] main idea was to compress the description of the oracle so that we simply keep track of the database for which the entries that 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 $\mathsf{CStO} = \mathsf{StdDecomp} \circ \mathsf{CStO}' \circ \mathsf{StdDecomp}$,[10] where $\mathsf{StdDecomp}$ 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 $S = (\{0,1\}^m \cup \{\bot\}) \times \{0,1\}^n$ and we call D *empty* when $D = ((\bot, 0^n), \cdots, (\bot, 0^n))$, where the number of $(\bot, 0^n)$ pairs is equal to q (i.e., the number of oracle query $\mathcal{A}$ 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 [Zha19] for the details.

The unitary operation $\mathsf{StdDecomp}$ takes as input $|x\rangle |y\rangle \otimes |D\rangle$ and does the following:

- If $D(x) = \bot$, 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 $D(x) \neq \bot$, denote $|D\rangle = \sum_{\alpha_x} a_{\alpha_x} |D' \cup (x, \alpha_x)\rangle$ where $D'(x) = \bot$, and return $|D'\rangle$ if $a_{\alpha_x} = \frac{1}{\sqrt{2^n}}$ for all $\alpha_x \in \{0,1\}^n$ 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, $\mathsf{StdDecomp}$ is the identity.

The unitary operation CStO' takes as input $|x\rangle |y\rangle \otimes |D\rangle$ 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 $\mathsf{StdDecomp}$, which guarantees that $D(x) \neq \bot$.

This completes the explanation of compressed standard oracles.

Almost Compressed Oracles. We consider a slight variant of the above compressed oracle known as the *almost* compressed oracle [LZ19]. The only difference is that an almost compressed oracle has polynomially many inputs which are uncompressed. That is, the database will be initialized as $D = ((\bot, 0^n), \cdots, (\bot, 0^n), (x_1^*, y_1^*), \cdots, (x_t^*, y_t^*))$ for some polynomial t, where $D(x_i^*) = y_i^*$ for all $i \in [t]$ is guaranteed. Since t is polynomial, this oracle still provides an efficient implementation. We denote this oracle as $\mathsf{CStO}[T]$, where $T = \{(x_i^*, y_i^*)_{i \in [t]}\}$. By considering the mixed state of $\mathcal{A}$, it can be checked that $\mathcal{A}$ cannot distinguish having oracle access to CStO or $\mathsf{CStO}[T]$ for randomly chosen y_i^* values.

C.2 Useful Lemmas

We prepare some useful lemmas required for our proof.

General Lemmas.

Lemma C.1 ([SXY18, Lem. 2.2]). *Let ℓ be an integer. Let $\mathsf{H}\colon \{0,1\}^\ell \times \mathcal{X} \to \mathcal{Y}$ and $\mathsf{H}'\colon \mathcal{X} \to \mathcal{Y}$ 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*

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

[10] Note that without loss of generality we simplify the discussion by omitting the operator $\mathsf{Increase}$ in [Zha19].

Lemma C.2 ([CMS19, Lem. 4.9, 5.7], [Zha19, Thm. 3]). *Let* CStO *be a compressed oracle over* $X \times Y$. *For every* $x \in X$, *let* S_x *be a subset of* Y *and denote* $s = \max_x |S_x|$. *Then, for any adversary making* q *queries to* CStO *initialized by the empty database* D, *if the database* D *is measured after the* q *queries, the probability it contains a pair of the form* $(x, y) \in S_x$ *for any* $x \in X$ *is at most* $6q^2 \cdot s/|Y| + 3 \cdot 2^{-\log(|Y|)/2}$.

Commutativity of Quantum Tests. In classical computation, the order of how one checks $x \in S$ and $x \in T$ for sets S, T makes no difference. However, in quantum computation, the order may matter since the two checks (expressed by unitaries) may not necessarily commute. 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 $C : Z \to \{0,1\}$. 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 C.3 (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 C.4 ([Zha19, 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 Almost Compressed Oracles. We recall the one-way to hiding (O2H) lemma for almost compressed oracles, which was originally defined for standard oracles [Unr14, AHU19].

Let T be a set over $X \times Y$ of size t. Let $\mathsf{CStO}[T]$ be an almost compressed standard oracle on database D such that $D(x) = y$ for all $(x, y) \in T$. For any subset S in X, denote $\mathsf{CStO}[T]\backslash S$ the *punctured* almost compressed oracle that is defined equal to $\mathsf{CStO}[T]$, except that we measure if the query of adversary $\mathcal{A}$ is in S or not. Formally, define a classical procedure f_{FIND_S} that takes $\mathcal{A}$'s input and computes 1 if and only if $\mathcal{A}$'s input is in S. Then the measurement is implemented by initializing a new register to $|0\rangle$, and then evaluating f_{FIND_S} 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 $|1\rangle$ at least once after the q queries made by $\mathcal{A}$. The following theorem allows us to bound $\mathcal{A}$'s distinguishing probability of two almost compressed random oracles that are decompressed with different output values. Below, let $a \| b$ denote a concatenation of two strings a and b.

Theorem C.5 (Almost Compressed Oracle O2H). *For* $t \in \mathbb{N}$ *and all* $i \in [t]$, *let* $x_i^* \leftarrow X$, $(y_i^*, \tilde{y_i}^*) \leftarrow Y^2$ *and set* $T = \{(x_i^*, y_i^*)\}_{i \in [t]}$, $\widetilde{T} = \{(x_i^*, \tilde{y}_i^*)\}_{i \in [t]}$, *and* $S = \{x_i^*\}_{i \in [t]}$. *Further, let* z *be a random string. Here,* $T, \widetilde{T}$, *and* z *may have an arbitrary joint distribution. Let* $\mathsf{CStO}[T]$ *be an almost compressed standard oracle that is uncompressed on points* $(x, y) \in T \subset X \times Y$. *Then, for all quantum algorithms* $\mathcal{A}$ *issuing at most* q *queries that, on input* z, *output either 0 or 1, we have*

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

If furthermore, all x_i^* have a common suffix x^*, i.e., for all $i \in [t]$, $x_i^* = \bar{x}_i^* \| x^*$, x^* is sampled uniformly random, and $(x^*, (\tilde{y}_i^*)_{i\in[t]})$ and z are independent, we have

$$\Pr[\mathsf{FIND} : \mathcal{A}^{\mathsf{CStO}[\widetilde{T}]\setminus S}(z)] \leq \frac{4q}{2^{|x^*|}}.$$

In particular, z may include $(\bar{x}_i^*)_{i\in[t]}$.

The first half of the above theorem is an immediate consequence of [AHU19, Theorem 1] by considering the mixed state of $\mathcal{A}$. Namely, by tracing out the oracle's register, the mixed state of $\mathcal{A}$ in our theorem and that of [AHU19] are identical. The second half of the above theorem follows from [AHU19, Theorem 2], where we again notice that the bound is agnostic to whether $\mathcal{A}$ is given oracle access to a standard oracle or an almost compressed oracle.

D 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 FrodoKEM[11] 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 Table 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.

[11] 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**

Contents