Faster generic IND-CCA secure KEM using encrypt-then-MAC

Anonymous Submission

Abstract. Indistinguishability under chosen-ciphertext attack (IND-CCA) is the accepted standard of security for key encapsulation mechanisms (KEM). However, provably chosen-ciphertext security is difficult to achieve from scratch. In this work, we generalize the "Hashed ElGamal" encryption scheme proposed by M. Abdalla et al into an IND-CCA secure KEM by combining an OW-PCA secure public-key encryption (PKE) scheme with an existentially unforgeable message authentication code (MAC) in a pattern called encrypt-then-MAC. Compared to the Fujisaki-Okamoto transformation, the encrypt-then-MAC transformation replaces de-randomization and re-encryption with computing a MAC tag. When instantiated with the PKE sub-routines of ML-KEM, the encrypt-then-MAC transformation achieves massive computational efficiency improvements over ML-KEM in a variety of practical scenarios.

Keywords: Key encapsulation mechanism, message authentication code, post-quantum cryptography, lattice cryptography, Fujisaki-Okamoto transformation

1 Introduction

A key encapsulation mechanism (KEM) [Sho01] is a cryptographic primitive that allows two parties to establish a shared secret over an insecure channel. The desired security standard for a KEM is called indistinguishability under chosen-ciphertext attack (IND-CCA). Intuitively speaking, IND-CCA security requires that no efficient adversary can distinguish a pseudorandom shared secret from a uniformly random bit string of equal length, even with access to a decapsulation oracle throughout the attack. However, building a provably IND-CCA secure KEM is tremendously difficult. Early attempts without formal proofs, such as RSA encryption defined in PKCS#1 v1.5 [Kal98], were later shown to be vulnerable to practical chosen-ciphertext attacks [Ble98]. In recent decades, the most viable approach has been to start with cryptographic primitives possessing weaker security properties, such as a public-key encryption (PKE) scheme with one-way security under chosen-plaintext attack (OW-CPA), then add steps to achieve *ciphertext non-malleability* [BN00]. Some of the earliest proposals for generic IND-CCA secure constructions include OAEP [BR94], Fujisaki-Okamoto transformation [FO99][FO13], REACT [OP01b], and GEM [CHJ$^+$02].

On the other hand, chosen-ciphertext security is a solved problem in symmetric cryptography. It is well understood that, by combining an IND-CPA secure symmetric encryption scheme with an existentially unforgeable message authentication code (MAC) in a pattern called encrypt-then-MAC [Kra01], one can build an authenticated encryption scheme [BN00] that achieves IND-CCA security. While this technique cannot be directly applied in the context of public-key cryptography due to the lack of a shared symmetric key between the two communicating parties, the concept of authenticating ciphertext using a MAC still has strong merits. Abdalla, Rogaway, and Bellare proposed DHIES (also known as "Hashed ElGamal")[ABR99][ABR01], a hybrid public-key encryption (HPKE) scheme whose IND-CCA security reduces to the Gap Diffie-Hellman assumption [OP01a] under the

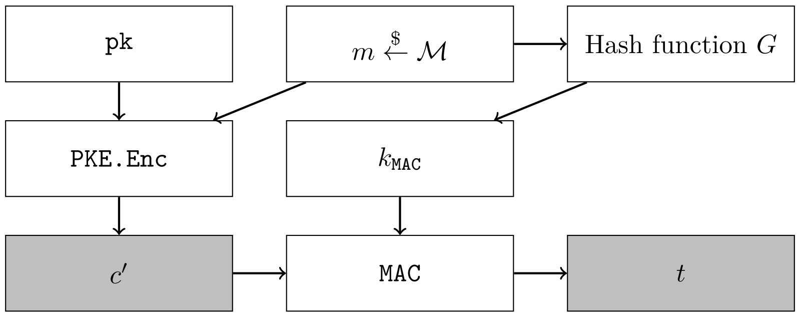

Figure 1: Combining PKE with MAC using encrypt-then-MAC to encapsulate a shared secret. The returned values are colored grey

random oracle model. The technique behind DHIES is to derive both the shared secret and a symmetric MAC key by hashing a random PKE plaintext, encrypting the PKE plaintext, then authenticating the PKE ciphertext using the previously derived MAC key. When the Gap Diffie-Hellman assumption holds and the MAC is existentially unforgeable, no efficient adversary can recover the decryption of an unknown ciphertext even with access to a decryption oracle because it cannot produce a valid tag for such unknown ciphertext.

1.1 Our contribution

In this paper, we propose a generic IND-CCA secure KEM using techniques adapted from DHIES. We call our proposed scheme the encrypt-then-MAC transformation due to the conceptual similarity to the namesake symmetric encryption technique for achieving authenticated encryption. Our scheme mainly differs from DHIES in its versatility and input requirement. Whereas the IND-CCA security of DHIES reduces specifically to the Gap Diffie-Hellman assumption, the chosen-ciphertext security of the encrypt-then-MAC KEM reduces more generally to the OW-PCA security [OP01b] of the input scheme (we will show in Section 3 that the Gap Diffie-Hellman assumption implies OW-PCA security). In addition, we propose that because each call to encapsulation samples a fresh PKE plaintext, the encrypt-then-MAC KEM can be instantiated with one-time secure MAC such as Poly1305 (M. Abdalla et al. originally proposed to use HMAC and CBC-MAC, which are many-time secure MAC but less efficient than one-time MAC, see Section 4.1), which greatly improves the computational efficiency of the scheme.

Construction. From a high level, the encrypt-then-MAC transformation takes as input a PKE and a MAC, then outputs a KEM. The output KEM encapsulates by encrypting a randomly sampled PKE plaintext and hashing the PKE plaintext-ciphertext pair into the shared secret. A symmetric key is derived from hashing the PKE plaintext, then used to produce a MAC tag on the PKE ciphertext. At decapsulation, the PKE secret key is used to recover the PKE plaintext, which is then hashed into a symmetric key that is used to verify the MAC tag against the input ciphertext. A summary of the data flow in the encrypt-then-MAC transformation can be found in Figure 1. A detailed description of individual routines can be found in Figure 6.

Security. In Section 3, we prove that if the input PKE is one-way secure against plaintext-checking attack (OW-PCA) and the MAC is one-time existentially unforgeable, then the output KEM is IND-CCA secure. The IND-CCA security of the output KEM reduces tightly to the OW-PCA security of the input PKE, and non-tightly to the unforgeability of the MAC, though the non-tightness will likely not be a weakness because the construction makes it easy to plug in MAC of any desired security level.

Implementation and performance comparison. In Section 4 and 5, we applied the encrypt-then-MAC transformation to the K-PKE subroutines defined in ML-KEM [oST24], which we call ML-KEM$^+$. Compared with ML-KEM, which is based on the $U^{\not\perp}$ variant (implicit rejection with rigidity) of the modular Fujisaki-Okamoto KEM transformation, ML-KEM$^+$ demonstrates massive performance improvements in decapsulation while only incurring minimal performance penalty in encapsulation runtime and ciphertext size. On an AMD EPYC 9R14 CPU, ML-KEM$^+$ with Poly1305 achieves between 72%-80% reduction of CPU cycles needed for decapsulation while only incurring 2%-7% increase of CPU cycles needed for encapsulation. See Table 1 for a summary of performance comparison of individual routines against ML-KEM.

Table 1: ML-KEM$^+$ with Poly1305 achieves large performance improvements in decapsulation at the cost of minimal penalty in encapsulation performance and ciphertext size

	ML-KEM$^+$ 512	ML-KEM$^+$ 768	ML-KEM$^+$ 1024
Encap (ccl/tick)	93157 (+1.8%)	146405 (+7.3%)	205763 (+3.3%)
Decap (ccl/tick)	33733 (-72.2%)	43315 (-76.8%)	51375 (-79.1%)
CT size (bytes)	784 (+2.1%)	1104 (+1.5%)	1584 (+1.0%)

We also measured the round trip time of key exchange protocols with various modes of authentication. When compared to ML-KEM, ML-KEM$^+$ achieves 24%-28% reduction of round trip time in unauthenticated key exchange (KE), 29%-35% in unilaterally authenticated key exchange (UAKE), and 35%-48% reduction in mutually authenticated key exchange (AKE). See Table 2 for a summary of round trip times.

Table 2: ML-KEM$^+$ achieves substantial RTT savings despite increased encapsulation cost and ciphertext size

	ML-KEM$^+$ 512	ML-KEM$^+$ 768	ML-KEM$^+$ 1024
KE RTT (μs)	70 (-23.9%)	99 (-26.7%)	138 (-28.5%)
UAKE RTT (μs)	103 (-29.0%)	144 (-33.0%)	202 (-34.8%)
AKE RTT (μs)	133 (-39.5%)	190 (-35.4%)	266 (-48.0%)

1.2 Related works

Optimal Asymmetric Encryption Padding (OAEP) [BR94][BDPR98] is a generic chosen-ciphertext secure PKE. Under the random oracle model, the chosen-ciphertext security of the OAEP encryption scheme reduces to the one-wayness of the input trapdoor permutation. Although it was discovered that there exist trapdoor permutations with which the OAEP encryption scheme does not achieve full IND-CCA security [Sho02], Fujisaki et al. later proved that the OAEP is IND-CCA secure when combined with the RSA trapdoor permutation [FOPS01][RSA78]. RSA-OAEP was standardized in PKCS#1 v2 [MKJR16] and is currently the most recommended of all RSA-based encryption schemes. Unfortunately, OAEP's requirement for a trapdoor permutation is immensely difficult to satisfy, and no other practical instantiation saw widespread adoption to this day.

The **Fujisaki-Okamoto transformation** [FO99][FO13] is another generic chosen-ciphertext secure transformation. While Fujisaki and Okamoto originally proposed a hybrid public-key encryption scheme whose IND-CCA security reduces non-tightly to the OW-CPA security of the underlying PKE and the IND-CPA security of the symmetric encryption scheme. Later works [Den03][HHK17][DNR04][HHM22][BP18] tightened the

```
$\texttt{KEM}_m^{\not\perp}.\texttt{KeyGen}()$
1: $(\texttt{pk}, \texttt{sk}') \xleftarrow{\$} \texttt{PKE.KeyGen}()$
2: $z \xleftarrow{\$} \mathcal{M}$
3: $\texttt{sk} \leftarrow (\texttt{sk}', \texttt{pk}, z)$
4: return $(\texttt{pk}, \texttt{sk})$
```

```
$\texttt{KEM}_m^{\not\perp}.\texttt{Encap}(\texttt{pk})$
1: $m \xleftarrow{\$} \mathcal{M}$
2: $r \leftarrow G(m)$
3: $c \leftarrow \texttt{PKE.Enc}(\texttt{pk}, m, r)$
4: $K \leftarrow H(m)$
5: return $(c, K)$
```

```
$\texttt{KEM}_m^{\not\perp}.\texttt{Decap}(\texttt{sk}, c)$
1: $\hat{m} \leftarrow \texttt{PKE.Dec}(\texttt{sk}', c)$
2: $\hat{r} \leftarrow G(m)$
3: $\hat{c} \leftarrow \texttt{PKE.Enc}(\texttt{pk}, \hat{m}, \hat{r})$
4: if $\hat{c} = c$ then
5:     $K \leftarrow H(\hat{m})$
6: else
7:     $K \leftarrow H(z, c)$
8: end if
9: return $K$
```

Figure 2: The $U_m^{\not\perp}$ variant of Fujisaki-Okamoto transformation is used in ML-KEM

security reduction, accounted for imperfect correctness, adapted the original proposal to build a KEM, and proved its security in the quantum random oracle model (QROM).

The modular Fujisaki-Okamoto KEM transformation is remarkably successful because of the simplicity of its construction, the tightness of the security bound, and the proven (though non-tight) security against quantum adversaries. It was adopted by many submissions to NIST's post-quantum cryptography competition, including Kyber [BDK+18], Saber [DKRV18], FrodoKEM [BCD+16], and classic McEliece [ABC+20] among others. The $U_m^{\not\perp}$ variant of the Fujisaki-Okamoto transformation (see Figure 2) is adopted in ML-KEM.

However, the Fujisaki-Okamoto transformation is not perfect. Because it uses derandomization and re-encryption for achieving ciphertext non-malleability, the Fujisaki-Okamoto transformation suffers from the following two problems:

- **Computational inefficiency**. The decapsulation routine needs to re-encrypt the decryption to ensure ciphertext has not been tempered with. For input PKE whose encryption routine carries substantial computational cost, such as most lattice-based cryptosystems, re-encryption slows down decapsulation significantly.

- **Side-channel vulnerability**. Re-encryption also introduces side-channels that can leak information about the decrypted PKE plaintext. As demonstrated in [UXT+22][RRCB19][TUX+23], these side-channels can be converted into efficient plaintext-checking attacks that can fully recover the secret key.

2 Preliminaries

2.1 Public-key encryption scheme

Syntax. A public-key encryption scheme $\texttt{PKE}(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ is a collection of three routines defined over some plaintext space $\mathcal{M}$ and some ciphertext space $\mathcal{C}$. Key generation $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}()$ is a randomized routine that returns a keypair. The encryption routine $\texttt{Enc} : (\texttt{pk}, m) \mapsto c$ encrypts the input plaintext under the input public key. The decryption routine $\texttt{Dec} : (\texttt{sk}, c) \mapsto m$ decrypts the input ciphertext under the input secret key. Where the encryption routine is randomized, we denote the randomness by a coin $r \in \mathcal{R}$, where $\mathcal{R}$ is called the coin space. The decryption routine is assumed to always be deterministic. Some decryption routines can detect malformed ciphertext and output the rejection symbol $\perp$ accordingly.

Correctness. Following the definition in [DNR04], a $\texttt{PKE}$ is δ-correct if:

$$E\left[\max_{m\in\mathcal{M}} P\left[\texttt{Dec}(\texttt{sk}, c) \neq m \mid c \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m)\right]\right] \leq \delta$$

Where the expectation is taken with respect to the probability distribution of all possible keypairs $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{PKE.KeyGen()}$. For many lattice-based cryptosystems, including ML-KEM, decryption failures could leak information about the secret key, although the probability of a decryption failure is low enough that classical adversaries cannot exploit decryption failure more than they can defeat the underlying lattice problem.

Security. The security of public-key encryption is conventionally discussed within the context of adversarial games played between a challenger and an adversary [GM82]. There are two main types of games: in the one-wayness game, the adversary is given a random encryption, then asked to produce the correct decryption; in the indistinguishability game, the adversary is given the encryption of one of two adversary-chosen plaintexts, then asked to decide which of the plaintexts coresopnds with the given encryption. Depending on the attack model, the adversary may have access to various oracles. Within the context of public-key cryptography, adversaries are always assumed to have the public key with which they can mount chosen-plaintext attack (CPA). If the adversary has access to a plaintext-checking oracle (PCO) [OP01b] then it can mount plaintext-checking attack (PCA). Where the adversary has access to a decryption oracle, it can mount chosen-ciphertext attacks (CCA).

```
OW-ATK Game
1: $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda)$
2: $m^* \xleftarrow{\$} \mathcal{M}$
3: $c^* \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m^*)$
4: $\hat{m} \xleftarrow{\$} A^{\mathcal{O}_{\texttt{ATK}}}(1^\lambda, \texttt{pk}, c^*)$
5: return $[\![\hat{m} = m^*]\!]$
```

```
IND-ATK Game
1: $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda)$
2: $(m_0, m_1) \xleftarrow{\$} A^{\mathcal{O}_{\texttt{ATK}}}(1^\lambda, \texttt{pk})$
3: $b \xleftarrow{\$} \{0, 1\}$
4: $c^* \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m_b)$
5: $\hat{b} \xleftarrow{\$} A^{\mathcal{O}_{\texttt{ATK}}}(1^\lambda, \texttt{pk}, c^*)$
6: return $[\![\hat{b} = b]\!]$
```

```
$\mathcal{O}_{\texttt{PCO}}(m, c)$
1: return $[\![m = \texttt{Dec}(\texttt{sk}, c)]\!]$
```

```
$\mathcal{O}_{\texttt{Dec}}(c)$
1: return $\texttt{Dec}(\texttt{sk}, c)$
```

Figure 3: The one-way game, indistinguishability game, plaintext-checking oracle (PCO), and decryption oracle. $\texttt{ATK} \in \{\texttt{CPA}, \texttt{PCA}, \texttt{CCA}\}$

The advantage of an adversary in the `OW-ATK` game is the probability that it ouputs the correct decryption. The advantage of an adversary in the `IND-ATK` game is defined below. A PKE is `OW-ATK`/`IND-ATK` secure if no efficient adversary has non-negligible advantage in the corresponding security game.

$$\texttt{Adv}_{\texttt{IND-ATK}}(A) = \left| P\left[\hat{b} = b\right] - \frac{1}{2} \right|$$

2.2 Key encapsulation mechanism (KEM)

Syntax A key encapsulation mechanism $\texttt{KEM}(\texttt{KeyGen}, \texttt{Encap}, \texttt{Decap})$ is a collection of three routines defined over some ciphertext space $\mathcal{C}$ and some key space $\mathcal{K}$. The key generation routine takes the security parameter 1^λ and outputs a keypair $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda)$. $\texttt{Encap}(\texttt{pk})$ is a probabilistic routine that takes a public key pk and outputs a pair of values (c, K) where $c \in \mathcal{C}$ is the ciphertext (also called encapsulation) and $K \in \mathcal{K}$ is the shared secret (also called session key). $\texttt{Decap}(\texttt{sk}, c)$ is a deterministic routine that takes the secret key sk and the encapsulation c and returns the shared secret K if the ciphertext is valid.

Some KEM constructions use explicit rejection, where if c is invalid then `Decap` will return a rejection symbol $\perp$; other KEM constructions use implicit rejection, where if c is invalid then `Decap` will return a fake session key that depends on the ciphertext and some other secret values.

Security The security of KEMs is similarly discussed in adversarial games (Figure 4), although the win conditions differ slightly from the win conditions of a PKE's indistinguishability game. In a KEM's indistinguishability game, an adversary is given the public key and a challenge ciphertext, then asked to distinguish a pseudorandom shared secret K_0 associated with the challenge ciphertext from a truly random bit string of equal length.

```
IND-ATK game
1: $(\texttt{pk}, \texttt{sk}) \overset{\$}{\leftarrow} \texttt{KeyGen}(1^\lambda)$
2: $(c^*, K_0) \overset{\$}{\leftarrow} \texttt{Encap}(\texttt{pk})$
3: $K_1 \overset{\$}{\leftarrow} \mathcal{K}$
4: $b \overset{\$}{\leftarrow} \{0, 1\}$
5: $\hat{b} \overset{\$}{\leftarrow} A^{\mathcal{O}_{\texttt{ATK}}}(1^\lambda, \texttt{pk}, c^*, K_b)$
6: return $[\![\hat{b} = b]\!]$
```

```
$\mathcal{O}_{\texttt{Decap}}(c)$
1: return $\texttt{Decap}(\texttt{sk}, c)$
```

Figure 4: `IND-ATK` game for KEM and decapsulation oracle $\mathcal{O}_{\texttt{Decap}}$

The decapsulation oracle $\mathcal{O}^{\texttt{Decap}}$ takes a ciphertext c and returns the output of the `Decap` routine using the secret key. The advantage $\epsilon_{\texttt{IND-CCA}}$ of an IND-CCA adversary $\mathcal{A}_{\texttt{IND-CCA}}$ is defined by the adversary's ability to correctly distinguish the two cases beyond blind guess:

$$\texttt{Adv}_{\texttt{IND-CCA}}(A) = \left| P[A^{\mathcal{O}_{\texttt{Decap}}}(a^\lambda, \texttt{pk}, c^*, K_b) = b] - \frac{1}{2} \right|$$

A KEM is `IND-ATK` secure if no efficient adversary has non-negligible advantage in the corresponding security game.

2.3 Message authentication code (MAC)

A message authentication code $\texttt{MAC}(\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$ is a collection of routines defined over some key space $\mathcal{K}$, some message space $\mathcal{M}$, and some tag space $\mathcal{T}$. The signing routine $\texttt{Sign}(k, m)$ authenticates the message m under the secret key k by producing a tag t (also called digest). The verification routine $\texttt{Verify}(k, m, t)$ takes the triplet of secret key, message, and tag, and outputs `1` if the message-tag pair is valid under the secret key, or `0` otherwise. Many MAC constructions are deterministic. For these constructions it is simpler to denote the signing routine by $t \leftarrow \texttt{MAC}(k, m)$ and perform verification using a simple comparison.

The security of a MAC is defined in an adversarial game in which an adversary, with access to some signing oracle $\mathcal{O}_{\texttt{Sign}}(m)$, tries to forge a new valid message-tag pair that has never been queried before. The existential unforgeability under chosen message attack (EUF-CMA) game is shown below:

```
EUF-CMA game
1: $k^* \xleftarrow{\$} \mathcal{K}$
2: $(\hat{m}, \hat{t}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\texttt{Sign}}}()$
3: return $[\![\texttt{Verify}(k^*, \hat{m}, \hat{t}) \land (\hat{m}, \hat{t}) \notin \mathcal{O}_{\texttt{Sign}}]\!]$
```

Figure 5: The existential forgery game

The advantage $\texttt{Adv}_{\texttt{EUF-CMA}}$ of the existential forgery adversary is the probability that it wins the EUF-CMA game. Some MACs are one-time existentially unforgeable, meaning that each secret key can be used to authenticate only a single message. The corresponding security game is modified such that the signing oracle will only answer a single signing query.

3 The encrypt-then-MAC transformation

In this section we introduce the encrypt-then-MAC transformation that transforms an OW-PCA secure PKE and an one-time existentially unforgeable MAC into an IND-CCA secure KEM. In Section 3.1 we reduce the IND-CCA security of the KEM tightly to the OW-PCA security of the underlying PKE, and non-tightly to the unforgeability of the MAC. In Section 3.2, we show that the "Hashed ElGamal" cryptosystem in [ABR01] is a special case of the encrypt-then-MAC transformation by reducing the OW-PCA security of the ElGamal cryptosystem to the Gap Diffie-Hellman assumption.

Let $\mathcal{B}^*$ denote the set of finite bit strings. Let $\texttt{PKE}(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ be a public-key encryption scheme defined over message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Let $\texttt{MAC} : \mathcal{K}_{\texttt{MAC}} \times \mathcal{B}^* \to \mathcal{T}$ be a deterministic message authentication code that takes a key $k \in \mathcal{K}_{\texttt{MAC}}$, some message $m \in \mathcal{B}^*$, and outputs a digest $t \in \mathcal{T}$. Let $G : \mathcal{M} \to \mathcal{K}_{\texttt{MAC}}$ be a hash function that maps from PKE's plaintext space to MAC's key space. Let $H : \mathcal{B}^* \to \mathcal{K}_{\texttt{KEM}}$ be a hash function that maps bit strings into the set of possible shared secrets. The encrypt-then-MAC transformation $\texttt{EtM}[\texttt{PKE}, \texttt{MAC}, G, H]$ constructs a key encapsulation mechanism $\texttt{KEM}_{\texttt{EtM}}(\texttt{KeyGen}, \texttt{Encap}, \texttt{Decap})$, whose routines are described in Figure 6.

The key generation routine of $\texttt{KEM}_{\texttt{EtM}}$ is largely identical to that of the PKE, only a secret value z is sampled as the implicit rejection symbol. In the encapsulation routine, a MAC key is derived from the randomly sampled plaintext $k \leftarrow G(m)$, then used to sign the unauthenticated ciphertext c'. Because the encryption routine might be randomized, the session key is derived from both the message and the ciphertext. Finally, the unauthenticated ciphertext c' and the tag t combine into the authenticated ciphertext c that would be transmitted to the peer. In the decapsulation routine, the decryption $\hat{m}$ of the unauthenticated ciphertext is used to re-derive the MAC key $\hat{k}$, which is then used to re-compute the tag $\hat{t}$. The ciphertext is considered valid if and only if the recomputed tag is identical to the input tag.

For an adversary A to produce a valid tag t for some unauthenticated ciphertext c' under the symmetric key $k \leftarrow G(\texttt{Dec}(\texttt{sk}', c'))$ implies that A must either know the symmetric key k or produce a forgery. Under the random oracle model, A also cannot know k without knowing its preimage $\texttt{Dec}(\texttt{sk}', c')$, so A must either have produced c' honestly, or have broken the one-way security of PKE. This means that the decapsulation oracle will not give out information on decryptions that the adversary does not already know.

However, a decapsulation oracle can still give out some information: for a known plaintext m, all possible encryptions $c' \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m)$ can be correctly signed, while ciphertexts that don't decrypt back to m cannot be correctly signed. This means that a

```
$\texttt{KEM}_{\texttt{EtM}}.\texttt{KeyGen}()$
1: $(\texttt{pk}, \texttt{sk}') \xleftarrow{\$} \texttt{PKE.KeyGen}()$
2: $z \xleftarrow{\$} \mathcal{M}$
3: $\texttt{sk} \leftarrow (\texttt{sk}', z)$
4: return $(\texttt{pk}, \texttt{sk})$
```

```
$\texttt{KEM}_{\texttt{EtM}}.\texttt{Encap}(\texttt{pk})$
1: $m \xleftarrow{\$} \mathcal{M}$
2: $k \leftarrow G(m)$
3: $c' \xleftarrow{\$} \texttt{PKE.Enc}(\texttt{pk}, m)$
4: $t \leftarrow \texttt{MAC}(k, c')$
5: $K \leftarrow H(m, c')$
6: $c \leftarrow (c', t)$
7: return $(c, K)$
```

```
$\texttt{KEM}_{\texttt{EtM}}.\texttt{Decap}(\texttt{sk}, c)$
1: $(c', t) \leftarrow c$
2: $(\texttt{sk}', z) \leftarrow \texttt{sk}$
3: $\hat{m} \leftarrow \texttt{PKE.Dec}(\texttt{sk}', c')$
4: $\hat{k} \leftarrow G(\hat{m})$
5: if $\texttt{MAC}(\hat{k}, c') \neq t$ then
6:     $K \leftarrow H(z, c')$
7: else
8:     $K \leftarrow H(\hat{m}, c')$
9: end if
10: return $K$
```

Figure 6: The encrypt-then-MAC transformation builds a KEM $\texttt{KEM}_{\texttt{EtM}}$ using a $\texttt{PKE}(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$, a $\texttt{MAC}$, and two hash functions G, H

decapsulation oracle can be converted into a plaintext-checking oracle (see Figure 7), so every chosen-ciphertext attack against the KEM can be converted into a plaintext-checking attack against the underlying PKE.

```
$\texttt{PCO}(m, c)$
1: $k \leftarrow G(m)$
2: $t \leftarrow \texttt{MAC}(k, c)$
3: return $[\![\mathcal{O}^{\texttt{Decap}}((c, t)) = H(m, c)]\!]$
```

Figure 7: With the encrypt-then-MAC transformation, every chosen-ciphertext attack against the KEM can be converted into a plaintext-checking attack against the underlying PKE

On the other hand, if the underlying PKE is OW-PCA secure and the underlying MAC is one-time existentially unforgeable, then the encrypt-then-MAC KEM is IND-CCA secure:

Theorem 1. *For every IND-CCA adversary A against $\texttt{KEM}_{\texttt{EtM}}$ that makes q decapsulation queries, there exists an OW-PCA adversary B who makes at least q plaintext-checking queries against the underlying PKE, and an one-time existential forgery adversary C against the underlying MAC such that*

$$\mathit{Adv}_{\textit{IND-CCA}}(A) \leq q \cdot \mathit{Adv}_{\textit{OT-MAC}}(C) + 2 \cdot \mathit{Adv}_{\textit{OW-PCA}}(B)$$

Theorem 1 naturally flows into an equivalence relationship between the security of the KEM and the security of the PKE:

Lemma 1. *$\texttt{KEM}_{\texttt{EtM}}$ is IND-CCA secure if and only if the input PKE is OW-PCA secure*

3.1 Proof of theorem 1

We will prove Theorem 1 using a sequence of game. A summary of the the sequence of games can be found in Figure 8 and 9. From a high level we made three incremental modifications to the IND-CCA game for $\texttt{KEM}_{\texttt{EtM}}$: replace true decapsulation with simulated decapsulation, replace the pseudorandom MAC key $k^* \leftarrow G(m^*)$ with a truly random MAC key $k^* \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$, and finally replace pseudorandom shared secret $K_0 \leftarrow H(m^*, c')$ with a truly random shared secret $K_0 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$. A OW-PCA adversary can then simulate the modified IND-CCA game for the KEM adversary, and the advantage of the OW-PCA adversary is associated with the probability of certain behaviors of the KEM adversary.

```
IND-CCA game for $\texttt{KEM}_{\texttt{EtM}}$

 1: $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KEM}_{\texttt{EtM}}.\texttt{KeyGen}()$
 2: $m^* \xleftarrow{\$} \mathcal{M}$
 3: $c' \xleftarrow{\$} \texttt{PKE.Enc}(\texttt{pk}, m^*)$
 4: $k^* \leftarrow G(m^*)$                              ▷ Game 0-1
 5: $k^* \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$    ▷ Game 2-3
 6: $t \leftarrow \texttt{MAC}(k^*, c')$
 7: $c^* \leftarrow c' \| t$
 8: $K_0 \leftarrow H(m^*, c')$                         ▷ Game 0-2
 9: $K_0 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$    ▷ Game 3
10: $K_1 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$
11: $b \xleftarrow{\$} \{0, 1\}$
12: $\hat{b} \leftarrow A^{\mathcal{O}^{\texttt{Decap}}}(\texttt{pk}, c^*, K_b)$     ▷ Game 0
13: $\hat{b} \leftarrow A^{\mathcal{O}_1^{\texttt{Decap}}}(\texttt{pk}, c^*, K_b)$   ▷ Game 1-3
14: return $[\![\hat{b} = b]\!]$
```

```
$\mathcal{O}^G(m)$

1: if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \tilde{m} = m$ then
2:     return $\tilde{k}$
3: end if
4: $k \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$
5: $\mathcal{L}^G \leftarrow \mathcal{L}^G \cup \{(m, k)\}$
6: return $k$
```

```
$\mathcal{O}^{\texttt{Decap}}(c)$

1: $(c', t) \leftarrow c$
2: $\hat{m} = \texttt{Dec}(\texttt{sk}', c')$
3: $\hat{k} \leftarrow G(\hat{m})$
4: if $\texttt{MAC}(\hat{k}, c') = t$ then
5:     $K \leftarrow H(\hat{m}, c')$
6: else
7:     $K \leftarrow H(z, c')$
8: end if
9: return $K$
```

```
$\mathcal{O}_1^{\texttt{Decap}}(c)$

1: $(c', t) \leftarrow c$
2: if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \tilde{m} = \texttt{Dec}(\texttt{sk}', c') \wedge \texttt{MAC}(\tilde{k}, c') = t$ then
3:     $K \leftarrow H(\tilde{m}, c')$
4: else
5:     $K \leftarrow H(z, c')$
6: end if
7: return $K$
```

```
$\mathcal{O}^H(m, c)$

1: if $\exists(\tilde{m}, \tilde{c}, \tilde{K}) \in \mathcal{L}^H : \tilde{m} = m \wedge \tilde{c} = c$ then
2:     return $\tilde{K}$
3: end if
4: $K \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$
5: $\mathcal{L}^H \leftarrow \mathcal{L}^H \cup \{(m, c, K)\}$
6: return $K$
```

Figure 8: Sequence of games

Proof. Game 0 is the standard IND-CCA game for KEMs. The decapsulation oracle $\mathcal{O}^{\texttt{Decap}}$ executes the decapsulation routine using the challenge keypair and return the results faithfully. The queries made to the hash oracles $\mathcal{O}^G, \mathcal{O}^H$ are recorded to their respective tapes $\mathcal{L}^G, \mathcal{L}^H$.

Game 1 is identical to game 0 except that the true decapsulation oracle $\mathcal{O}^{\texttt{Decap}}$ is replaced with a simulated oracle $\mathcal{O}_1^{\texttt{Decap}}$. Instead of directly decrypting c' as in the decapsulation routine, the simulated oracle searches through the tape $\mathcal{L}^G$ to find a matching query $(\tilde{m}, \tilde{k})$ such that $\tilde{m}$ is the decryption of c'. The simulated oracle then uses $\tilde{k}$ to validate the tag t against c'.

If the simulated oracle accepts the queried ciphertext as valid, then there is a matching query that also validates the tag, which means that the queried ciphertext is honestly generated. Therefore, the true oracle must also accept the queried ciphertext. On the other hand, if the true oracle rejects the queried ciphertext (and output the implicit rejection $H(z, c')$), then the tag is simply invalid under the MAC key $k = G(\texttt{Dec}(\texttt{sk}', c'))$. Therefore, there could not have been a matching query that also validates the tag, and the simulated oracle must also rejects the queried ciphertext.

This means that from the adversary A's perspective, game 1 and game 0 differ only when the true oracle accepts while the simulated oracle rejects, which means that t is a valid tag for c' under $k = G(\texttt{Dec}(\texttt{sk}', c'))$, but k has never been queried. Under the random oracle model, such k is a uniformly random sample of $\mathcal{K}_{\texttt{MAC}}$ that the adversary does not know, so for A to produce a valid tag is to produce a forgery against the MAC under an unknown and uniformly random key. Furthermore, the security game does not include a signing oracle, so this is a zero-time forgery. While zero-time forgery is not a standard security definition for a MAC, we can bound it by the advantage of a one-time forgery adversary C:

$$P\left[\mathcal{O}^{\texttt{Decap}}(c) \neq \mathcal{O}_1^{\texttt{Decap}}(c)\right] \leq \texttt{Adv}_{\texttt{OT-MAC}}(C)$$

Across all q decapsulation queries, the probability that at least one query is a forgery is thus at most $q \cdot P\left[\mathcal{O}^{\texttt{Decap}}(c) \neq \mathcal{O}_1^{\texttt{Decap}}(c)\right]$. By the difference lemma:

$$\texttt{Adv}_{G_0}(A) - \texttt{Adv}_{G_1}(A) \leq q \cdot \texttt{Adv}_{\texttt{OT-MAC}}(C)$$

Game 2 is identical to game 1, except that the challenger samples a uniformly random MAC key $k^* \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$ instead of deriving it from m^*. From A's perspective the two games are indistinguishable, unless A queries G with the value of m^*. Denote the probability that A queries G with m^* by $P[\texttt{QUERY G}]$, then:

$$\texttt{Adv}_{G_1}(A) - \texttt{Adv}_{G_2}(A) \leq P\left[\texttt{QUERY G}\right]$$

Game 3 is identical to game 2, except that the challenger samples a uniformly random shared secret $K_0 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$ instead of deriving it from m^* and c'. From A's perspective the two games are indistinguishable, unless A queries H with $(m^*, \cdot)$. Denote the probability that A queries H with $(m^*, \cdot)$ by $P[\texttt{QUERY H}]$, then:

$$\texttt{Adv}_{G_2}(A) - \texttt{Adv}_{G_3}(A) \leq P\left[\texttt{QUERY H}\right]$$

Since in game 3, both K_0 and K_1 are uniformly random and independent of all other variables, no adversary can have any advantage: $\texttt{Adv}_{G_3}(A) = 0$.

We will bound $P[\texttt{QUERY G}]$ and $P[\texttt{QUERY H}]$ by constructing a OW-PCA adversary B against the underlying PKE that uses A as a sub-routine. B's behaviors are summarized in Figure 9.

B simulates game 3 for A: receiving the public key $\texttt{pk}$ and challenge encryption c'^*, B samples random MAC key and session key to produce the challenge encapsulation, then feeds it to A. When simulating the decapsulation oracle, B uses the plaintext-checking oracle to look for matching queries in $\mathcal{L}^G$. When simulating the hash oracles, B uses the plaintext-checking oracle to detect when $m^* = \texttt{Dec}(\texttt{sk}', c'^{\star})$ has been queried. When m^* is queried, B terminates A and returns m^* to win the OW-PCA game. In other words:

```
$B(\texttt{pk}, c'^*)$
1: $z \xleftarrow{\$} \mathcal{M}$
2: $k \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$
3: $t \leftarrow \texttt{MAC}(k, c'^*)$
4: $c^* \leftarrow (c'^*, t)$
5: $K \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$
6: $\hat{b} \leftarrow A^{\mathcal{O}_B^{\texttt{Decap}}, \mathcal{O}_B^G, \mathcal{O}_B^H}(\texttt{pk}, c^*, K)$
7: if $\texttt{ABORT}(m)$ then
8:     return $m$
9: end if
```

```
$\mathcal{O}_B^{\texttt{Decap}}(c)$
1: $(c', t) \leftarrow c$
2: if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \texttt{PCO}(c', \tilde{m}) = 1 \land \texttt{MAC}(\tilde{k}, c') = t$ then
3:     $K \leftarrow H(\tilde{m}, c')$
4: else
5:     $K \leftarrow H(z, c')$
6: end if
7: return $K$
```

```
$\mathcal{O}_B^H(m, c)$
if $\texttt{PCO}(m, c'^*) = 1$ then
    $\texttt{ABORT}(m)$
end if
if $\exists(\tilde{m}, \tilde{c}, \tilde{K}) \in \mathcal{L}^H : \tilde{m} = m \land \tilde{c} = c$ then
    return $\tilde{K}$
end if
$K \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$
$\mathcal{L}^H \leftarrow \mathcal{L}^H \cup \{(m, c, K)\}$
return $K$
```

```
$\mathcal{O}_B^G(m)$
1: if $\texttt{PCO}(m, c'^*) = 1$ then
2:     $\texttt{ABORT}(m)$
3: end if
4: if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \tilde{m} = m$ then
5:     return $\tilde{k}$
6: end if
7: $k \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$
8: $\mathcal{L}^G \leftarrow \mathcal{L}^G \cup \{(m, k)\}$
9: return $k$
```

Figure 9: OW-PCA adversary B simulates game 3 for IND-CCA adversary A

$$P[\texttt{QUERY G}] \leq \texttt{Adv}_{\texttt{OW-PCA}}(B)$$
$$P[\texttt{QUERY H}] \leq \texttt{Adv}_{\texttt{OW-PCA}}(B)$$

Combining all equations above produce the desired security bound. □

3.2 ElGamal is OW-PCA secure

We show that the DHAES/DHIES hybrid encryption scheme is a special case of the encrypt-then-MAC transformation. Specifically, we will sketch a proof of the following lemma:

Lemma 2. *For every OW-PCA adversary A against the ElGamal cryptosystem, there exists a Gap Diffie-Hellman problem solver B such that:*

$$Adv_{GapDH}(B) = Adv_{OW\text{-}PCA}(A)$$

In other words, ElGamal is OW-PCA secure under the Gap Diffie-Hellman assumption.

Each ElGamal cryptosystem [Gam85] is parameterized by a cyclic group $G = \langle g \rangle$ of prime order $q > 2$. A summary of the routine is shown in Figure 10:

```
KeyGen()
1: $x \overset{\$}{\leftarrow} \mathbb{Z}_q$
2: $\mathtt{sk} \leftarrow x$
3: $\mathtt{pk} \leftarrow g^x$
4: return $(\mathtt{pk}, \mathtt{sk})$
```

```
Enc(pk = $g^x$, $m \in G$)
Require: $m \in G$
1: $y \overset{\$}{\leftarrow} \mathbb{Z}_q$
2: $w \leftarrow g^y$
3: $v \leftarrow m \cdot (g^x)^y$
4: return $c = (w, v)$
```

```
Dec(sk = $x$, $c = (w, v) \in G^2$)
1: $\hat{m} \leftarrow (w^x)^{-1} \cdot v$
2: return $\hat{m}$
```

Figure 10: ElGamal cryptosystem

The security of ElGamal cryptosystem reduces to the conjectured intractability of the computational Diffie-Hellman problem and the decisional Diffie-Hellman problem:

Definition 1 (**computational Diffie-Hellman problem (CDH)**)**.** Let $x, y \overset{\$}{\leftarrow} \mathbb{Z}_q$ be uniformly random samples. Given (g, g^x, g^y), compute g^{xy}.

Definition 2 (**decisional Diffie-Hellman problem (DDH)**)**.** Let $x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_q$ be uniformly random samples. Let $h \overset{\$}{\leftarrow} \{g^z, g^{xy}\}$ be randomly chosen between g^z and g^{xy}. Given (g, g^x, g^y, h), determine whether h is g^{xy} or g^z

It is also conjectured in [ABR01] that for certain choice of cyclic group G, the computational Diffie-Hellman problem remains intractable even if the adversary as access to a restricted decisional Diffie-Hellman oracle. This assumption is captured in the Gap Diffie-Hellman problem:

Definition 3 (**Gap Diffie-Hellman problem**)**.** Let $G = \langle g \rangle$ be a cyclic group of prime order $q > 2$. Let $x, y \overset{\$}{\leftarrow} \mathbb{Z}_q$ be uniformly random samples. Given (g, g^x, g^y) and a restricted DDH oracle $\mathcal{O}^{\text{DDH}} : (u, v) \mapsto [\![u^x = v]\!]$, compute g^{xy}.

We now present the proof for Lemma 2

Proof. We will prove by sequence of game. A summary can be found in Figure 11

```
$G_0 - G_2$
1: $x \overset{\$}{\leftarrow} \mathbb{Z}_q$
2: $m^* \overset{\$}{\leftarrow} G$
3: $y \overset{\$}{\leftarrow} \mathbb{Z}_q, w \leftarrow g^y$
4: $v \leftarrow m^* \cdot (g^x)^y$                    ▷ $G_0$ - $G_1$
5: $v \overset{\$}{\leftarrow} G$                           ▷ $G_2$
6: $c^* \leftarrow (w, v)$
7: $\hat{m} \overset{\$}{\leftarrow} A^{\mathtt{PCO}}(g^x, c^*)$                ▷ $G_0$
8: $\hat{m} \overset{\$}{\leftarrow} A^{\mathtt{PCO}_1}(g^x, c^*)$               ▷ $G_1$ - $G_2$
9: return $[\![\hat{m} = m^*]\!]$                 ▷ $G_0$ - $G_1$
10: return $[\![\hat{m} = w^{-x} \cdot v]\!]$         ▷ $G_2$
```

```
PCO($m, c = (w, v)$)
1: return $[\![m = (w^x)^{-1} \cdot v]\!]$
```

```
PCO$_1$($m, c = (w, v)$)
1: return $[\![(w^x) = m^{-1} \cdot v]\!]$
```

Figure 11: Lemma 2 sequence of games

Game 0 is the OW-PCA game. Adversary A has access to the plaintext-checking oracle PCO and wins the game if it can correctly recover the challenge plaintext m^*.

Game 1 is identical to game 0, except that the formulation of the PCO is changed. When servicing the plaintext-checking query $(m, c = (w, v))$, $\texttt{PCO}_1$ checks whether w^x is equal to $m^{-1} \cdot v$. Observe that in the cyclic group G, the algebraic expressions in PCO and $\texttt{PCO}_1$ are equivalent, which means that $\texttt{PCO}_1$ behaves identically to PCO.

Game 2 is identical to game 1 except for two modifications: first, when computing the challenge ciphertext, v is no longer computed from m^* but is randomly sampled; second, the win condition changed from $\hat{m} = m^*$ to $\hat{m} = w^{-x} \cdot v$. Notice that in game 0-1, $v \overset{\$}{\leftarrow} m^* \cdot (g^x)^y$ follows uniform random distribution on the cyclic group G because m^* is uniformly random in the cyclic group, so adversary A retains its advantage in "recovering the decryption" when v becomes truly uniformly random in the cyclic group. It is easy to verify that the two win conditions are equivalent. Up to this point, we have simply moved things around, and game 0 through game 2 are algebraically equivalent:

$$\texttt{Adv}_0(A) = \texttt{Adv}_1(A) = \texttt{Adv}_2(A)$$

The Gap Diffie-Hellman adversary B can perfectly simulate game 2 for A (see Figure 12): B receives as the Gap Diffie-Hellman problem inputs g^x and g^y. g^x simulates an ElGamal public key, where as g^y simulates the first component of the challenge ciphertext. As in game 2, the second component of the challenge ciphertext can be randomly sampled. Finally, the $\texttt{PCO}_1$ from game 2 can be perfectly simulated using the restricted DDH oracle $\mathcal{O}^{\text{DDH}}$.

```
$B^{\mathcal{O}^{\text{DDH}}}(g, g^x, g^y)$
1: $w \leftarrow g^y$
2: $v \overset{\$}{\leftarrow} G$
3: $c^* \leftarrow (w, v)$
4: $\hat{m} \overset{\$}{\leftarrow} A^{\texttt{PCO}_2}(g^x, c^*)$
5: return $\hat{m}^{-1} \cdot v$
```

```
$\mathcal{O}^{\text{DDH}}(u, v)$
1: return $[\![u^x = v]\!]$
```

```
$\texttt{PCO}_2(m, c = (w, v))$
1: return $\mathcal{O}^{\text{DDH}}(w, m^{-1} \cdot v)$
```

Figure 12: Gap Diffie-Hellman adversary B simulates game 2 for A

If A wins game 2, then its output is $\hat{m} = w^{-x} \cdot v = g^{-xy} \cdot v$, so $m^{-1} \cdot v$ is g^{xy}, the correct answer to the Gap Diffie-Hellman problem. In other words, B solves its Gap Diffie-Hellman problem if and only if A wins the simulated game 2.

$$\texttt{Adv}_2(A) = \texttt{Adv}_{\text{GapDH}}(B)$$

□

4 Practical instantiation with ML-KEM

ML-KEM is an IND-CCA secure key encapsulation mechanism standardized by NIST in FIPS 203 [oST24]. The chosen-ciphertext security of ML-KEM is achieved in two steps. First, ML-KEM constructs a PKE whose IND-CPA security reduces to the conjectured intractability of the decisional Module Learning with Error (MLWE) problem. Then, the $U_m^{\not\perp}$ variant of the Fujisaki-Okamoto transformation is applied to convert the IND-CPA secure PKE into an IND-CCA secure KEM. A summary of the IND-CPA PKE routines can be found in Figure 13. We refer readers to [HHK17] and [oST24] for more details of the $U_m^{\not\perp}$ variant of the Fujisaki-Okamoto transformation.

```
K-PKE.KeyGen()
1: $A \xleftarrow{\$} R_q^{k \times k}$
2: $\mathbf{s} \xleftarrow{\$} \mathcal{X}_{\eta_1}^k$
3: $\mathbf{e} \xleftarrow{\$} \mathcal{X}_{\eta_1}^k$
4: $\mathbf{t} \leftarrow A\mathbf{s} + \mathbf{e}$
5: $\texttt{pk} \leftarrow (A, \mathbf{t})$
6: $\texttt{sk} \leftarrow \mathbf{s}$
7: return $(\texttt{pk}, \texttt{sk})$
```

```
K-PKE.Enc(pk, m)
Ensure: $\texttt{pk} = (A, \mathbf{t})$
Ensure: $m \in R_2$
1: $\mathbf{r} \xleftarrow{\$} \mathcal{X}_{\eta_1}^k$
2: $\mathbf{e}_1 \xleftarrow{\$} \mathcal{X}_{\eta_2}^k$
3: $e_2 \xleftarrow{\$} \mathcal{X}_{\eta_2}$
4: $\mathbf{c}_1 \leftarrow A\mathbf{r} + \mathbf{e}_1$
5: $c_2 \leftarrow \mathbf{t}^\intercal \mathbf{r} + e_2 + m \cdot \lfloor \frac{q}{2} \rceil$
6: return $(\mathbf{c}_1, c_2)$
```

```
K-PKE.Dec(sk, c)
Ensure: $c = (\mathbf{c}_1, c_2)$
Ensure: $\texttt{sk} = \mathbf{s}$
1: $\hat{m} \leftarrow c_2 - \mathbf{c}_1^\intercal \cdot \mathbf{s}$
2: $\hat{m} \leftarrow \texttt{Round}(\hat{m})$
3: return $\hat{m}$
```

Figure 13: K-PKE is an IND-CPA secure PKE based on the conjectured intractability of the Module Learning with Error (MWLE) problem

We applied the encrypt-then-MAC transformation to the K-PKE routines of ML-KEM. The resulting KEM is called ML-KEM$^+$. A complete description of the KEM routines can be found in Figure 14

```
ML-KEM+.KeyGen()
1: $z \xleftarrow{\$} \{0,1\}^{256}$
2: $(\texttt{pk}, \texttt{sk}') \xleftarrow{\$} \texttt{K-PKE.KeyGen}()$
3: $h \leftarrow H(\texttt{pk})$
4: $\texttt{sk} \leftarrow (sk' \| \texttt{pk} \| h \| z)$
5: return $(\texttt{pk}, \texttt{sk})$
```

```
ML-KEM+.Encap(pk)
Require: Public key pk
1: $m \xleftarrow{\$} \{0,1\}^{256}$
2: $(\overline{K}, k) \leftarrow G(m \| H(\texttt{pk}))$
3: $c' \xleftarrow{\$} \texttt{K-PKE.Enc}(\texttt{pk}, m)$
4: $t \leftarrow \texttt{MAC}(k, c')$
5: $K \leftarrow \texttt{KDF}(\overline{K} \| c')$
6: $c \leftarrow (c', t)$
7: return $(c, K)$
```

```
ML-KEM+.Decap(sk, c)
Require: Secret key $\texttt{sk} = (\texttt{sk}' \| \texttt{pk} \| h \| z)$
Require: Ciphertext $c = (c' \| t)$
1: $(\texttt{sk}', \texttt{pk}, h, z) \leftarrow \texttt{sk}$
2: $(c', t) \leftarrow c$
3: $\hat{m} \leftarrow \texttt{K-PKE.Dec}(\texttt{sk}', c')$
4: $(\overline{K}, \hat{k}) \leftarrow G(\hat{m} \| h)$
5: $\hat{t} \leftarrow \texttt{MAC}(\hat{k}, c')$
6: if $\hat{t} = t$ then
7:     $K \leftarrow \texttt{KDF}(\overline{K} \| t)$
8: else
9:     $K \leftarrow \texttt{KDF}(z \| t)$
10: end if
11: return $K$
```

Figure 14: ML-KEM$^+$ applies our encrypt-then-MAC transformation to K-PKE

When applying our encrypt-then-MAC transformation to K-PKE, we modified the routines described in Figure 6 such that the public key is added into the derivation of the MAC key and the shared secret. There are two main reasons behind this modification. First, in a realistic key exchange scenario, the public key will be generated by one party while the random PKE plaintext is sampled by the other party. Hashing both the public key and the plaintext ensures that both parties in a key exchange have inputs into deriving the shared secret. The second reason is to prevent a dictionary attack on the MAC key. If the MAC key is derived on plaintext alone, then an adversary can pre-compute a large dictionary mapping MAC keys to the corresponding PKE plaintext. Upon receiving the ciphertext, the adversary can search through the MAC keys and recover the PKE plaintext.

Adding the public key into the derivation of the MAC key will dramatically increase the space requirement and render such dictionary attack infeasible in practice.

4.1 Choosing a message authenticator

For implementation, we instantiated the MAC with a selection that covered a wide range of MAC designs, including Poly1305 [Ber05], GMAC [MV04], CMAC [IK03][BR05], and KMAC [Gro13].

Poly1305 and GMAC are both Carter-Wegman style authenticators [WC81] that compute the tag using finite field arithmetic. Generically speaking, Carter-Wegman MAC operates by breaking the message into message blocks that can then be parsed into finite field elements. The tag is computed by evaluating a polynomial whose coefficients are the message blocks and whose interdeterminate is the secret key.

Specifically, Poly1035 operates in the prime field $\mathbb{F}_q$ where $q = 2^{130} - 5$ whereas GMAC operates in the binary field $\mathbb{F}_{2^{128}}$. In OpenSSL's implementation, standalone Poly1305 is a one-time secure MAC, whereas GMAC uses a nonce and AES as the PRF and is thus many-time secure (in OpenSSL GMAC is AES-256-GCM except all data is fed into the "associated data" section and thus not encrypted).

CMAC is based on the CBC-MAC with the block cipher instantiated from AES-256. To compute a CMAC tag, the message is first broke into 128-bit blocks with appropriate padding. Each block is first XOR'd with the previous block's output, then encrypted under AES using the symmetric key. The final output is XOR'd with a sub key derived from the symmetric key, before being encrypted for one last time.

KMAC is based on the SHA-3 family of sponge functions [oST15]. We chose KMAC-256, which uses Shake256 as the underlying extendable output functions. KMAC allows variable-length key and tag, but we chose the 256 bits for key length and 128 bits for tag size for consistency with other authenticators.

5 Performance comparison

We measured the real-world performance of the individual routines of our ML-KEM$^+$ construction, as well as the round trip time of performing key exchange over a network in a variety of authentication modes. Our implementation extended from the reference implementation by the PQCrystals team [BDK$^+$24]. All C code is compiled with `GCC 11.4.1` and `OpenSSL 3.0.8`. All binaries are executed on an AWS c7a.medium instance (AMD EPYC 9R14 CPU at 3.7 GHz and 1 GB of RAM) in the `us-west-2` region.

5.1 MAC performance

To isolate the performance characteristics of each authenticator in our instantiation of ML-KEM$^+$ we measured the CPU cycles needed for each authenticator to compute a tag on random inputs whose sizes correspond to the ciphertext sizes of ML-KEM. The measurements are summarized in Table 3.

Table 3: CPU cycles needed to compute tag on various input sizes

Input size: 768 bytes			**Input size: 1088 bytes**			**Input size: 1568 bytes**		
MAC	Median	Average	MAC	Median	Average	MAC	Median	Average
Poly1305	909	2823	Poly1305	961	2704	Poly1305	1065	1809
GMAC	3899	4859	GMAC	3899	4827	GMAC	4055	5026
CMAC	6291	6373	CMAC	7305	7588	CMAC	8735	8772
KMAC	6373	7791	KMAC	9697	9928	KMAC	11647	12186

5.2 Individual routine performance

Compared to the $U_m^{\not\perp}$ variant of Fujisaki-Okamoto transformed used in ML-KEM, the encrypt-then-MAC transformation achieves massive CPU cycle count reduction in decapsulation while incurring only a minimal increase of encapsulation cycle count and ciphertext size. Since `K-PKE.Enc` carries significantly more computational complexity than `K-PKE.Dec` or any MAC we chose, the performance advantage of the encrypt-then-MAC transformation over the $U_m^{\not\perp}$ transformation is dominated by the runtime saving gained from replacing *re-encryption* with MAC. A comparison between ML-KEM and variations of the ML-KEM$^+$ can be found in Table 4.

Remark. We also included the performance of Kyber as it is submitted to the third round of the NIST PQC standardization project, which differs from ML-KEM by deriving the shared secret from hashing the public key, the PKE plaintext, and the PKE ciphertext, while ML-KEM derives its shared secret by hashing only the public key and the PKE plaintext. This results in ML-KEM having meaningful performance improvement in encapsulation over Kyber, though the performance difference in decapsulation is negligible.

Table 4: CPU cycles of each KEM routine

128-bit security	
size parameters (bytes)	
`pk` size	800
`sk` size	1632
ct size	768
`KeyGen` cycles/tick	
Median	75945
Average	76171

KEM variant	`Encap` cycles/tick		`Decap` cycles/tick	
	Median	Average	Median	Average
ML-KEM-512	91467	92065	121185	121650
Kyber512	97811	98090	119937	120299
ML-KEM$^+$-512 w/ Poly1305	93157	93626	33733	33908
ML-KEM$^+$-512 w/ GMAC	97369	97766	37725	37831
ML-KEM$^+$-512 w/ CMAC	99739	99959	40117	39943
ML-KEM$^+$-512 w/ KMAC	101009	101313	40741	40916

192-bit security	
size parameters (bytes)	
`pk` size	1184
`sk` size	2400
ct size	1088
`KeyGen` cycles/tick	
Median	129895
Average	130650

KEM variant	`Encap` cycles/tick		`Decap` cycles/tick	
	Median	Average	Median	Average
ML-KEM-768	136405	147400	186445	187529
Kyber768	153061	153670	182129	182755
ML-KEM$^+$-768 w/ Poly1305	146405	146860	43315	43463
ML-KEM$^+$-768 w/ GMAC	149525	150128	46513	46706
ML-KEM$^+$-768 w/ CMAC	153139	153735	49841	50074
ML-KEM$^+$-768 w/ KMAC	155219	155848	52415	52611

256-bit security	
size parameters (bytes)	
`pk` size	1568
`sk` size	3168
ct size	1568
`KeyGen` cycles/tick	
Median	194921
Average	195465

KEM variant	`Encap` cycles/tick		`Decap` cycles/tick	
	Median	Average	Median	Average
ML-KEM-1024	199185	199903	246245	247320
Kyber1024	222351	223260	258231	259067
ML-KEM$^+$-1024 w/ Poly1305	205763	206499	51375	51562
ML-KEM$^+$-1024 w/ GMAC	208805	209681	54573	54780
ML-KEM$^+$-1024 w/ CMAC	213667	214483	59175	59408
ML-KEM$^+$-1024 w/ KMAC	216761	217468	62269	62516

5.3 Key exchange protocols

A common application of key encapsulation mechanism is key exchange protocols, where two parties establish a shared secret using a public channel. We used ML-KEM$^+$ to implement the key exchange protocols described in [BDK+18] and compared the round trip time (RTT) of ML-KEM$^+$ key exchange with ML-KEM key exchange. We denote the party who sends the first message to be the client and the other party to be the server. Round trip time (RTT) is defined to be the time interval between the moment before the client starts generating ephemeral keypairs and the moment after the client derives the final session key. All experiements are run on a pair of AWS c7a.medium instances both located in the `us-west-2` region. For each experiment, a total of 10,000 rounds of key exchange are performed, with the median and average round trip time (measured in microsecond) recorded.

Remark. Using KEM for authentication is especially interesting within the context of post-quantum cryptography: post-quantum KEM schemes usually enjoy better performance characteristics than post-quantum signature schemes with faster runtime, smaller memory footprint, and smaller communication sizes. KEMTLS was proposed in 2020 as an alternative to existing TLS handshake protocols, and many experimental implementations have demonstrated the performance advantage [SSW20].

5.3.1 Unauthenticated key exchange

In unauthenticated key exchange (KE), a single pair of ephemeral keypair $(\texttt{pk}_e, \texttt{sk}_e) \xleftarrow{\$} \texttt{KeyGen}()$ is generated by the client. The client transmits the ephemeral public key $\texttt{pk}_e$ to the server, who runs the encapsulation routine $(c_e, K_e) \xleftarrow{\$} \texttt{Encap}(\texttt{pk}_e)$ and transmits the ciphertext c_e back to the client. The client finally decapsulates the ciphertext to recover the shared secret $K_e \leftarrow \texttt{Decap}(\texttt{sk}_e, c_e)$. The key exchange routines are summarized in Figure 15. The key derivation function is instantiated using Shake256, and the final session key is 256 bits in length. The RTT comparison is summarized in Table 5

```
KE_C()
1: $(\texttt{pk}_e, \texttt{sk}_e) \xleftarrow{\$} \texttt{KeyGen}()$
2: $\texttt{send}(\texttt{pk}_e)$
3: $c_e \leftarrow \texttt{read}()$
4: $K_e \leftarrow \texttt{Decap}(\texttt{sk}_e, c_e)$
5: $K \leftarrow \texttt{KDF}(K)$
6: return $K$
```

```
KE_S()
1: $\texttt{pk}_e \leftarrow \texttt{read}()$
2: $(c_e, K_e) \xleftarrow{\$} \texttt{Encap}(\texttt{pk}_e)$
3: $\texttt{send}(c_e)$
4: $K \leftarrow \texttt{KDF}(K_e)$
5: return $K$
```

Figure 15: Unauthenticated key exchange (KE) routines

Table 5: KE RTT comparison

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-512	800	768	92	97
ML-KEM-512$^+$ w/ Poly1305	800	784	70	72
ML-KEM-512$^+$ w/ GMAC	800	784	73	76
ML-KEM-512$^+$ w/ CMAC	800	784	75	79
ML-KEM-512$^+$ w/ KMAC	800	784	76	78

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-768	1184	1088	135	140
ML-KEM-768$^+$ w/ Poly1305	1184	1104	99	104
ML-KEM-768$^+$ w/ GMAC	1184	1104	101	105
ML-KEM-768$^+$ w/ CMAC	1184	1104	103	109
ML-KEM-768$^+$ w/ KMAC	1184	1104	103	107

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-1024	1568	1568	193	199
ML-KEM-1024$^+$ w/ Poly1305	1568	1584	138	141
ML-KEM-1024$^+$ w/ GMAC	1568	1584	140	145
ML-KEM-1024$^+$ w/ CMAC	1568	1584	143	148
ML-KEM-1024$^+$ w/ KMAC	1568	1584	144	149

5.3.2 Unilaterally authenticated key exchange

In unilaterally authenticated key exchange (UAKE), the authenticating party proves its identity to the other party by demonstrating possession of a secret key that corresponds to a published long-term public key. In this implementation, the client possesses the long-term public key $\texttt{pk}_S$ of the server, and the server authenticates itself by correctly decrypting the challenge encryption sent by the client. UAKE routines are summarized in Figure 16.

```
UAKE_C(pk_S)
Require: Server's long-term $\texttt{pk}_S$
1: $(\texttt{pk}_e, \texttt{sk}_e) \xleftarrow{\$} \texttt{KeyGen}()$
2: $(c_S, K_S) \xleftarrow{\$} \texttt{Encap}(\texttt{pk}_S)$
3: $\texttt{send}(\texttt{pk}_e, c_S)$
4: $c_e \leftarrow \texttt{read}()$
5: $K_e \leftarrow \texttt{Decap}(\texttt{sk}_e, c_e)$
6: $K \leftarrow \texttt{KDF}(K_e \| K_S)$
7: return $K$
```

```
UAKE_S(sk_S)
Require: Server's long-term $\texttt{sk}_S$
1: $(\texttt{pk}_e, c_S) \leftarrow \texttt{read}()$
2: $K_S \leftarrow \texttt{Decap}(\texttt{sk}_S, c_S)$
3: $(c_e, K_e) \xleftarrow{\$} \texttt{Encap}(\texttt{pk}_e)$
4: $\texttt{send}(c_e)$
5: $K \leftarrow \texttt{KDF}(K_e \| K_S)$
6: return $K$
```

Figure 16: Unilaterally authenticated key exchange (UAKE) routines

In addition to the long-term key, the client will also generate an ephemeral keypair as it does in an unauthenticated key exchange, and the session key is derived by applying the KDF to the concatenation of both the ephemeral shared secret and the shared secret encapsulated under server's long-term key. This helps the key exchange to achieve weak forward secrecy [CK01]. Performance data is summaried in Table 6.

Table 6: UAKE RTT comparison

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-512	1568	768	145	151
ML-KEM-512$^+$ w/ Poly1305	1584	784	103	106
ML-KEM-512$^+$ w/ GMAC	1584	784	106	110
ML-KEM-512$^+$ w/ CMAC	1584	784	108	112
ML-KEM-512$^+$ w/ KMAC	1584	784	109	113

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-768	2272	1088	215	222
ML-KEM-768$^+$ w/ Poly1305	2288	1104	144	150
ML-KEM-768$^+$ w/ GMAC	2288	1104	149	156
ML-KEM-768$^+$ w/ CMAC	2288	1104	153	160
ML-KEM-768$^+$ w/ KMAC	2288	1104	154	159

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-1024	3136	1568	310	318
ML-KEM-1024$^+$ w/ Poly1305	3152	1584	202	209
ML-KEM-1024$^+$ w/ GMAC	3152	1584	212	228
ML-KEM-1024$^+$ w/ CMAC	3152	1584	212	218
ML-KEM-1024$^+$ w/ KMAC	3152	1584	213	220

5.3.3 Mutually authenticated key exchange (AKE)

Mutually authenticated key exchange is largely identical to unilaterally authenticated key exchange, except for that client authentication is required. This means that client possesses server's long-term public key and its own long-term secret key, while the server possesses client's long-term public key and its own long-term secret key. The session key is derived by applying KDF onto the concatenation of shared secrets produced under the ephemeral keypair, server's long-term keypair, and client's long-term keypair, in this order. the key exchange routines are described in Figure 17 and the performance data summarized in Table 7.

```
$\mathtt{AKE_C}(\mathtt{pk}_S, \mathtt{sk}_C)$
Require: Server's long-term $\mathtt{pk}_S$
Require: Client's long-term $\mathtt{sk}_C$
1: $(\mathtt{pk}_e, \mathtt{sk}_e) \xleftarrow{\$} \mathtt{KeyGen}()$
2: $(c_S, K_S) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk}_S)$
3: $\mathtt{send}(\mathtt{pk}_e, c_S)$
4: $(c_e, c_C) \leftarrow \mathtt{read}()$
5: $K_e \leftarrow \mathtt{Decap}(\mathtt{sk}_e, c_e)$
6: $K_C \leftarrow \mathtt{Decap}(\mathtt{sk}_e, c_C)$
7: $K \leftarrow \mathtt{KDF}(K_e \| K_S \| K_C)$
8: return $K$
```

```
$\mathtt{AKE_S}(\mathtt{sk}_S, \mathtt{pk}_C)$
Require: Server's long-term $\mathtt{sk}_S$
Require: Client's long-term $\mathtt{pk}_C$
1: $(\mathtt{pk}_e, c_S) \leftarrow \mathtt{read}()$
2: $K_S \leftarrow \mathtt{Decap}(\mathtt{sk}_S, c_S)$
3: $(c_e, K_e) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk}_e)$
4: $(c_C, K_C) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk}_C)$
5: $\mathtt{send}(c_e, c_C)$
6: $K \leftarrow \mathtt{KDF}(K_e \| K_S \| K_C)$
7: return $K$
```

Figure 17: Mutually authenticated key exchange (AKE) routines

Table 7: AKE RTT comparison

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-512	1568	1536	220	213
ML-KEM-512$^+$ w/ Poly1305	1584	1568	133	138
ML-KEM-512$^+$ w/ GMAC	1584	1568	139	143
ML-KEM-512$^+$ w/ CMAC	1584	1568	143	148
ML-KEM-512$^+$ w/ KMAC	1584	1568	145	151

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-768	2272	2176	294	301
ML-KEM-768$^+$ w/ Poly1305	2288	2208	190	196
ML-KEM-768$^+$ w/ GMAC	2288	2208	197	210
ML-KEM-768$^+$ w/ CMAC	2288	2208	202	208
ML-KEM-768$^+$ w/ KMAC	2288	2208	204	210

KEM variant	Client TX bytes	Server TX bytes	RTT (μs)	
			Median	Average
ML-KEM-1024	3136	3136	512	511
ML-KEM-1024$^+$ w/ Poly1305	3152	3168	266	273
ML-KEM-1024$^+$ w/ GMAC	3152	3168	273	282
ML-KEM-1024$^+$ w/ CMAC	3152	3168	280	287
ML-KEM-1024$^+$ w/ KMAC	3152	3168	282	288

6 Conclusions and future works

The encrypt-then-MAC transformation is a generic KEM construction whose IND-CCA security reduces to the OW-PCA security of the input PKE and the one-time existential unforgeability of the input MAC in the random oracle model. Compared to the Fujisaki-Okamoto transformation, the encrypt-then-MAC replaces the computationally expensive re-encryption with computing a MAC tag. At the cost of minimal increase in encapsulation cost and ciphertext size, the encrypt-then-MAC substantially improves the efficiency of the decapsulation routine. Where the input PKE's encryption is slower than decryption, the encrypt-then-MAC KEM achieves meaningful time savings in practical key exchange protocols.

Unfortunately, ML-KEM's K-PKE subroutines are not OW-PCA secure. In fact, as Chris Peikert pointed out in [Pei14], most lattice-based cryptosystems are not OW-PCA secure due to the search-decision equivalence of lattice problems. While we applied the encrypt-then-MAC transformation to K-PKE for performance comparison, ML-KEM$^+$ is not chosen-ciphertext secure, and should not be used in production systems. The natural questions is thus to find a suitable cryptosystem to apply the encrypt-then-MAC transformation to.

RSA and ElGamal. Combining encrypt-then-MAC with ElGamal results in the "Hashed ElGamal" scheme proposed in [ABR99][ABR01] which is proven IND-CCA secure. RSA is also known to be OW-PCA secure because it is a trapdoor permutation and thus a rigid PKE. However, because of RSA's rigidity, there exists even more efficient KEM transformation (such as RSA-KEM [Sho01]) that only adds a single hash to the base PKE routines. While applying encrypt-then-MAC to RSA will result in an IND-CCA secure KEM, such construction offers no meaningful advantage to RSA-KEM.

Code-based cryptosystems. Because the general problem of decoding a linear code is proven NP hard [BMvT78], code-based cryptosystems does not suffer from the inherent search-decision equivalence of lattice problems and thus be viable candidates with OW-PCA security. Unfortunately, among the code-based submissions to NIST PQC, HQC [MAB$^+$18] and BIKE [ABB$^+$22] are known to be vulnerable to key-recovery plaintext-checking attacks (KR-PCA) [TUX$^+$23]. On the other hand, classic McEliece [ABC$^+$20] seems to be PCA secure and thus a viable candidate, although in classic McEliece, the decoding routine is more expensive than the encryption routine, so applying encrypt-then-MAC may not yield meaningful performance gains.

Isogeny-based cryptosystems. The intractability assumptions of isogeny-based cryptography resemble the classical Diffie-Hellman assumptions, and it seems possible to formulate a "Gap Diffie-Hellman assumption" in supersingular isogeny [FTTY18]. In fact, SIKE [ACC$^+$17] also uses the Fujisaki-Okamoto transformation, and given the heavy computational cost of isogeny computation, replacing re-encryption with MAC will result in substantial performance improvements. While SIKE and SIDH were found to be insecure [CD23], other isogeny-based cryptosystem such as CSIDH [CLM$^+$18] remains unaffected by the aforementioned attack and might be suitable candidates.

References

[ABB$^+$22] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, et al. Bike: bit flipping key encapsulation. *NIST PQC Round 4*, 2022.

[ABC$^+$20] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, RubenNiederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter

Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic mceliece. Technical report, National Institute of Standards and Technology, 2020.

[ABR99] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHAES: an encryption scheme based on the diffie-hellman problem. *IACR Cryptol. ePrint Arch.*, page 7, 1999.

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[ACC+17] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 152:154–155, 2017.

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018. ACM, 2016.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

[BDK+24] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. official reference implementation of the kyber key encapsulation mechanism, 2024.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[Ber05] Daniel J. Bernstein. The poly1305-aes message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3):384–386, 1978.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[BP18] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. *IACR Cryptol. ePrint Arch.*, page 526, 2018.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[BR05] John Black and Phillip Rogaway. CBC macs for arbitrary-length messages: The three-key constructions. *J. Cryptol.*, 18(2):111–131, 2005.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 423–447. Springer, 2023.

[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, *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*, volume 2271 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2002.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

[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 D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[Den03] Alexander W. Dent. A designer's guide to kems. In Kenneth G. Paterson, editor, *Cryptography and Coding, 9th IMA International Conference, Cirencester, UK, December 16-18, 2003, Proceedings*, volume 2898 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2003.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*, volume 10831 of *Lecture Notes in Computer Science*, pages 282–305. Springer, 2018.

[DNR04] Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 342–360. Springer, 2004.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

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

[FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is secure under the RSA assumption. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2001.

[FTTY18] Atsushi Fujioka, Katsuyuki Takashima, Shintaro Terada, and Kazuki Yoneyama. Supersingular isogeny diffie-hellman authenticated key exchange. In Kwangsu Lee, editor, *Information Security and Cryptology - ICISC 2018 - 21st International Conference, Seoul, South Korea, November 28-30, 2018, Revised Selected Papers*, volume 11396 of *Lecture Notes in Computer Science*, pages 177–195. Springer, 2018.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.

[Gro13] Joint Task Force Transformation Initiative Interagency Working Group. Security and privacy controls for federal information systems and organizations. Technical Report NIST Special Publication (SP) 800-53, Rev. 4, Includes updates as of January 22, 2015, National Institute of Standards and Technology, Gaithersburg, MD, 2013.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore,* *MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.

[HHM22] Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Failing gracefully: Decryption failures and the fujisaki-okamoto transform. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 414–443. Springer, 2022.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: one-key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[Kal98] Burt Kaliski. PKCS #1: RSA encryption version 1.5. *RFC*, 2313:1–19, 1998.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, 2001.

[MAB$^+$18] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2(4):13, 2018.

[MKJR16] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[OP01a] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2001.

[OP01b] Tatsuaki Okamoto and David Pointcheval. REACT: rapid enhanced-security asymmetric cryptosystem transform. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175. Springer, 2001.

[oST15] National Institute of Standards and Technology. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical Report Federal Information Processing Standards Publication (FIPS) NIST FIPS 202, U.S. Department of Commerce, Washington, D.C., 2015.

[oST24] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report Federal Information Processing Standards Publication (FIPS) NIST FIPS 203, U.S. Department of Commerce, Washington, D.C., 2024.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2014.

[RRCB19] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based PKE and KEM schemes. *IACR Cryptol. ePrint Arch.*, page 948, 2019.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sho01] Victor Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptol. ePrint Arch.*, page 112, 2001.

[Sho02] Victor Shoup. OAEP reconsidered. *J. Cryptol.*, 15(4):223–249, 2002.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1461–1480. ACM, 2020.

[TUX$^+$23] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. Multiple-valued plaintext-checking side-channel attacks on post-quantum kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):473–503, 2023.

[UXT$^+$22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):296–322, 2022.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.