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

Anonymous submission

Abstract. TODO: write abstract later

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

Key encapsulation mechanism (KEM) is a public-key cryptographic primitive that allows two parties to establish a shared secret over an insecure communication channel. The accepted security requirement of a KEM is *Indistinguishability under adaptive chosen ciphertext attack (IND-CCA)*. Intuitively speaking, IND-CCA security implies that no efficient adversary (usually defined as probabilistic polynomial time Turing machine) can distinguish a pseudorandom shared secret from a uniformly random bit string of identical length even with access to a decapsulation oracle. Unfortunately, CCA security is difficult to achieve from scratch. Early attempts at constructing CCA secure public-key cryptosystems using only heuristics argument and without using formal proof, such as RSA encryption in PKCS #1 [40] and RSA signature in ISO 9796 [1], were badly broken with sophisticated cryptanalysis [15,29,21]. Afterwards, provable chosen ciphertext security became a necessity for new cryptographic protocols. There have been many provable CCA secure constructions since then. Notable examples include Optimal Asymmetric Encryption Padding (OAEP) [6], which is combined with RSA [26] into the widely adopted RSA-OAEP. The Fujisaki-Okamoto transformation [25,34] is another generic CCA secure transformation that was thoroughly studied and widely adopted, particularly by many KEM candidates in NIST's Post Quantum Cryptography (PQC) standardization project.

Chosen ciphertext security is a solved problem within the context of symmetric cryptography. It is well understood that authenticated encryption can be achieved by combining a semantically secure symmetric encryption scheme with an existentially unforgeable message authentication code (MAC) using either the "encrypt-then-MAC" (AES-GCM, ChaCha20-Poly1305) or "MAC-then-encrypt" pattern (AES-CCM)[5,42]. However, adapting this technique for public-key cryptosystems is challenging, since the two communicating parties do not have a pre-shared symmetric key. One attempt at such adaption is the Diffie-Hellman integrated encryption scheme (DHIES) [2,3] proposed by Abdalla, Bellare, and Rogaway, who proved its chosen ciphertext security under a non-standard but well studied assumption called "Gap Diffie-Hellman problem" [51].

1.1 Our contributions

Our contributions are as follows:

Generic CCA secure KEM transformation. We propose the encrypt-then-MAC KEM transformation. Our transformation constructs a KEM with provable CCA security under the random oracle model using a public-key encryption scheme (PKE) with one-wayness under plaintext-checking attack and a message authentication code with existential unforgeability. Compared to the Fujisaki-Okamoto transformation, which is widely adopted by many KEM candidates in NIST's Post Quantum Cryptography (PQC) standardization project, our transformation replaces *de-randomization* (which might degrade the security of a randomized cryptosystem) and *re-encryption* (which is computationally inefficient and introduces additional risk of side channels) with computing MAC tag.

Instantiation with Classic McEliece. We present McElieceEtM, a KEM constructed from applying the encrypt-then-MAC transformation to the CPA subroutines of Classic McEliece. We conjecture McElieceEtM to be IND-CCA secure as there is currently no known plaintext-checking attack against Classic McEliece. We implemented McElieceEtM by adapting the reference C implementation, which achieved 9-12% percent speed up in throughput (enc + dec).

Generalizing DHIES. We demonstrate that the encrypt-then-MAC KEM transformation is a generalization of DHIES [3,2], a well-known CCA secure hybrid public-key encryption scheme. Specifically, we prove that the Gap Diffie-Hellman assumption, which is used to prove the CCA security of DHIES, implies one-wayness under plaintext checking attack (OW-PCA) of the ElGamal cryptosystem.

1.2 Related works

OAEP *Optimal Asymmetric Encryption Padding (OAEP)* [6], proposed by Mihir Bellare and Phillip Rogaway in 1994, was one of the earliest provably secure CCA transformations. However, Victor Shoup identified a non-trivial gap in OAEP's security proof that cannot be filled under ROM[56], although Fujisaki et al. later proved that RSA-OAEP is secure under the RSA assumption [26]. RSA-OAEP is widely used in secure communication protocols such as TLS 1.2. The main drawback of OAEP is that it requires its input to be an one-way trapdoor permutation, which is difficult to find. To this day, RSA remains the only viable candidate to apply OAEP to.

REACT/GEM Okamoto and Pointchval proposed REACT [52] in 2001, followed by GEM [20] in 2002. Both are generic CCA transformation with security proved under ROM. Okamoto and Pointcheval first defined the security notion of

one-wayness under plaintext checking attack (OW-PCA) and reduced the CCA security of the transformation to the OW-PCA security of the input public-key cryptosystem.

Fujisaki-Okamoto transformation Fujisaki and Okamoto proposed to construct CCA secure hybrid PKE by combining a OW-CPA secure PKE and a semantically secure symmetric-key encryption (SKE) scheme [25]. The main techniques, namely *de-randomization* and *re-encrytion* were both introduced in the original proposal. Under ROM, Fujisaki and Okamoto reduced the CCA security of the hybrid PKE tightly to the semantic security of the input SKE and *non-tightly* to the OW-CPA security of the input PKE (with loss factor q, the number of hash oracle queries). Later works extended the original proposal to build CCA secure KEM: KEM's security model makes building secure KEM simpler than building secure PKE, and it is well-known that combining a CCA secure KEM with a CCA secure data encapsulation mechanism (DEM), such as some authenticated encryption scheme (e.g. AES-GCM, AES-CCM, ChaCha20-Poly1305), results in a CCA secure hybrid PKE [57,55]. Further studies [23,34,13,35,62,39] gave tighter security bounds, accounted for decryption failures in the underlying PKE, and analyzed the security under quantum random oracle model (QROM). To this day, the Fujisaki-Okamoto transformation is the only known generic CCA secure transformation that can convert OW-CPA/IND-CPA PKE into a CCA secure KEM. Because of the minimal input requirement and the simple construction, the Fujisaki-Okamoto transformation was widely adopted among post-quantum KEM candidates submitted to the PQC standardization project, including Kyber [17], Saber [22], FrodoKEM [16], and Classic McEliece [11,19,12].

Despite its widespread adoption, the Fujisaki-Okamoto transformation has many flaws:

- **Computational inefficiency.** In all variants of Fujisaki-Okamoto transformation, decapsulation routine needs to re-encrypt the decryption to ensure ciphertext non-malleability. For input PKE whose encryption routine carries significant computational cost, such as most lattice-based cryptosystems, re-encryption substantially slows down decapsulation.
- **Side-channel vulnerability.** Re-encryption introduces side-channels that can leak information about the decrypted PKE plaintext. As demonstrated in [59,58,36], these side-channels can be converted into efficient plaintext-checking attacks that can fully recover the secret key
- **Security degradation.** *de-randomization* can degrade the security of a randomized PKE. Where the security parameters did not account for this loss, the security of the KEM can fall below the expected level. Consequently, larger parameters are necessary to account for the security loss, which slows down the cryptosystem [8,9].

1.3 Paper organization

In Section 2, we review the preliminary definitions and theorems. In Section 3, we present the encrypt-then-MAC KEM transformation, proves its CCA security, and discusses practical attacks. In Section 4, we present McElieceEtM, an instantiation of the encrypt-then-MAC transformation using subroutines from Classic McEliece. In Section 5, we show that the encrypt-then-MAC transformation is a generalization of DHIES.

2 Preliminaries

Notations. For finite set S, we use $x \xleftarrow{\$} S$ to denote uniformly random sample from the set. For deterministic routine f, we use $x \leftarrow f()$ to denote assigning the output of f to x. For randomized routine g, we use $x \xleftarrow{\$} g()$ to denote assigning the randomized output of g to x. For boolean statement B, we denote $[\![B]\!]$ to be 1 if B is true and 0 otherwise. For probabilistic Turing machine A, we denote access to oracle $\mathcal{O}$ by $A^{\mathcal{O}}$. We sometimes model hash function H as a random oracle, in which case we will use $\mathcal{L}^H$ to denote the record of input-output queries $(x, H(x))$ made to the oracle.

2.1 Public-key encryption scheme

Syntax A public-key encryption scheme (PKE) is a collection fo three routines $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ defined over some plaintext space $\mathcal{M}$ and some ciphertext space $\mathcal{C}$. Key generation $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}(1^\lambda)$ is a randomized routine that returns a keypair consisting of a public encryption key and a secret decryption key. The encryption routine $\mathtt{Enc} : (\mathtt{pk}, m) \mapsto c$ encrypts the input plaintext m under the input public key $\mathtt{pk}$ and produces a ciphertext c. The decryption routine $\mathtt{Dec} : (\mathtt{sk}, c) \mapsto m$ decrypts the input ciphertext c under the input secret key and produces the corresponding plaintext. 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. Decryption routines are assumed to always be deterministic.

Correctness A PKE is δ-correct if

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

Where the expectation is taken with respect to the probability distribution of all possible keypairs. For many lattice-based cryptosystems, 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 problems.

Security The security of PKE's is conventionally discussed using adversarial games played between a challenger and an adversary [28]. In the OW-ATK game (Figure 1), the challenger samples a random keypair and a random encryption. The adversary is given the public key, the random encryption (also called the challenge ciphertext), and access to ATK, then asked to decrypt the challenge ciphertext.

OW-ATK game

$(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}(1^\lambda)$
$m^* \xleftarrow{\$} \mathcal{M}$
$c^* \xleftarrow{\$} \mathtt{Enc}(\mathtt{pk}, m)$
$\hat{m} \xleftarrow{\$} A^{\mathrm{ATK}}(1^\lambda, \mathtt{pk}, c^*)$
return $[\![\hat{m} = m^*]\!]$

Fig. 1: The one-wayness game: challenger samples a random keypair and a random encryption, and the adversary wins if it correctly produces the decryption

The advantage of an adversary is its probability of producing the correct decryption: $\mathrm{Adv}^{\text{OW-ATK}}_{\mathtt{PKE}}(A) = P[\hat{m} = m^*]$. A PKE is said to be OW-ATK secure if no efficient adversary can win the OW-ATK game with non-negligible probability.

IND-ATK game

$(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}(1^\lambda)$
$(m_0, m_1) \xleftarrow{\$} A^{\mathrm{ATK}}(1^\lambda, \mathtt{pk})$
$b \xleftarrow{\$} \{0, 1\}$
$c^* \xleftarrow{\$} \mathtt{Enc}(\mathtt{pk}, m_b)$
$\hat{b} \xleftarrow{\$} A^{\mathrm{ATK}}(1^\lambda, \mathtt{pk}, c^*)$
return $[\![\hat{b} = b]\!]$

Fig. 2: IND-ATK game: adversary is asked to distinguish the encryption of one message from another

In the IND-ATK game (Figure 2), the adversary chooses two distinct messages and receives the encryption of one of them, randomly selected by the challenger. The advantage of an adversary is its probability of correctly distinguishing the ciphertext of one message from the other beyond blind guess:

$\mathrm{Adv}^{\text{IND-ATK}}_{\mathtt{PKE}}(A) = |P[\hat{b} = b] - \frac{1}{2}|$. A PKE is said to be IND-ATK secure if no efficient adversary can win the IND-ATK game with non-negligible advantage.

$\mathcal{O}^{\mathtt{Dec}}(c)$
1: **return** $\mathtt{Dec}(\mathtt{sk}, c)$

$\mathcal{O}^{\mathtt{PCO}}(m, c)$
1: **return** $[\![\mathtt{Dec}(\mathtt{sk}, c) = m]\!]$

Fig. 3: Decryption oracle $\mathcal{O}^{\mathtt{Dec}}$ (left) answers decryption queries by returning the decryption of the queried ciphertext. Plaintext-checking oracle $\mathcal{O}^{\mathtt{PCO}}$ (right) answers whether the queried plaintext is the decryption of the queried ciphertext.

In public-key cryptography, all adversaries are assumed to have access to the public key (ATK = CPA). If the adversary has access to a decryption oracle, it is said to mount chosen-ciphertext attack (ATK = CCA). If the adversary has access to a plaintext-checking oracle (PCO) , then it is said to mount plaintext-checking attack (ATK = PCA). See Figure 3 for algorithmic description of the oracles.

$$\text{ATK} = \begin{cases} \text{CPA} & \mathcal{O}^{\text{ATK}} = \cdot \\ \text{PCA} & \mathcal{O}^{\text{ATK}} = \mathcal{O}^{\mathtt{PCO}} \\ \text{CCA} & \mathcal{O}^{\text{ATK}} = \mathcal{O}^{\mathtt{Dec}} \end{cases}$$

2.2 Key encapsulation mechanism (KEM)

Syntax A key encapsulation mechanism (KEM) is a collection of three routines $(\mathtt{KeyGen}, \mathtt{Encap}, \mathtt{Decap})$ defined over some ciphertext space $\mathcal{C}$ and some key space $\mathcal{K}$. Key generation $\mathtt{KeyGen} : 1^\lambda \mapsto (\mathtt{pk}, \mathtt{sk})$ is a randomized routine that returns a keypair. Encapsulation $\mathtt{Encap} : \mathtt{pk} \mapsto (c, K)$ is a randomized routine that takes a public encapsulation key and returns a pair of ciphertext c and shared secret K (also commonly referred to as session key). Decapsulation $\mathtt{Decap} : (\mathtt{sk}, c) \mapsto K$ is a deterministic routine that uses the secret key $\mathtt{sk}$ to recover the shared secret K from the input ciphertext c. Where the KEM chooses to reject invalid ciphertext explicitly, the decapsulation routine can also output the rejection symbol $\perp$. We assume a KEM to be perfectly correct:

$$P\left[\mathtt{Decap}(\mathtt{sk}, c) = K \mid (\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}(1^\lambda); (c, K) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk})\right] = 1$$

Security Similar to PKE security, the security of KEM is discussed using adversarial games. In the IND-ATK game (Figure 4), the challenger generates a random keypair and encapsulates a random secret; the adversary is given the

KEM IND-ATK Game

1: $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}((1^\lambda))$
2: $(c^*, K_0) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk})$
3: $K_1 \xleftarrow{\$} \mathcal{K}$
4: $b \xleftarrow{\$} \{0,1\}$
5: $\hat{b} \xleftarrow{\$} A^{\mathtt{ATK}}(1^\lambda, \mathtt{pk}, c^*, K_b)$
6: **return** $[\![\hat{b} = b]\!]$

Fig. 4: The IND-ATK game for KEM

public key and the ciphertext, then asked to distinguish the shared secret from a random bit string.

The advantage of an adversary is its probability of winning beyong blind guess. A KEM is said to be IND-ATK secure if no efficient adversary can win the IND-ATK game with non-negligible advantage.

$$\mathtt{Adv}^{\mathtt{IND\text{-}ATK}}(A) = \left| P\left[A^{\mathtt{ATK}}(1^\lambda, c^*, K_b) = b \mid \begin{array}{r} (\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}(1^\lambda); \\ (c^*, K_0) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk}); \\ K_1 \xleftarrow{\$} \mathcal{K}; b \xleftarrow{\$} \{0,1\} \end{array} \right] - \frac{1}{2} \right|$$

By default, all adversaries are assumed to have the public key, with which they can mount chosen plaintext attacks (ATK = CPA). If the adversary as access to a decapsulation oracle $\mathcal{O}^{\mathtt{Decap}} : c \mapsto \mathtt{Decap}(\mathtt{sk}, c)$, it is said to mount a chosen-ciphertext attack (ATK = CCA).

2.3 Message authentication code (MAC)

Syntax A message authentication code (MAC) is a collection of two routines $(\mathtt{Sign}, \mathtt{Verify})$ defined over some key space $\mathcal{K}$, some message space $\mathcal{M}$, and some tag space $\mathcal{T}$. The signing routine $\mathtt{Sign} : (k, m) \mapsto t$ authenticates the message m under the symmetric key k by producing a tag t. The verification routine $\mathtt{Verify}(k, m, t)$ outputs 1 if the message-tag pair (m, t) is authentic under the symmetric key k and 0 otherwise. Many MAC constructions are deterministic: for these constructions it is simpler to denote the signing routine by $t \leftarrow \mathtt{MAC}(k, m)$, and verification done using a simple comparison. Some MAC constructions require a distinct or randomized nonce $r \xleftarrow{\$} \mathcal{R}$, and the signing routine will take this additional argument $t \leftarrow \mathtt{MAC}(k, m; r)$.

Security The standard security notion for a MAC is *existential unforgeability under chosen message attack (EUF-CMA)*. We define it using an adversarial game in which an adversary has access to a signing oracle $\mathcal{O}^{\mathtt{Sign}} : m \mapsto$

$\texttt{Sign}(k, m)$ and tries to produce a valid message-tag pair that has not been queried from the signing oracle (Figure 5).

MAC EUF-CMA game

$k^* \xleftarrow{\$} \mathcal{K}$
$(\hat{m}, \hat{t}) \xleftarrow{\$} A^{\texttt{CMA}}()$
return $[\![\texttt{Verify}(k^*, \hat{m}, \hat{t}) \wedge (\hat{m}, \hat{t}) \notin \mathcal{O}^{\texttt{Sign}}]\!]$

Fig. 5: The signing oracle signs the queried message with the secret key. The adversary must produce a message-tag pair that has never been queried before

The advantage of the adversary is the probability that it successfully produces a valid message-tag pair. A MAC is said to be EUF-CMA secure if no efficient adversary has non-negligible advantage. Some MACs are *one-time existentially unforgeable* (we call them one-time MAC), meaning that each secret key can be used to authenticate exactly one message. The corresponding security game is identical to the EUF-CMA game except for that the signing oracle will only answer up to one query.

3 The encrypt-then-MAC transformation

In this section we present the encrypt-then-MAC KEM transformation. The transformation constructs an IND-CCA secure KEM using an OW-PCA secure PKE and an existentially unforgeable MAC. Our scheme is inspired by DHIES, but differs from it in two key aspects: whereas DHIES reduces its CCA security specifically to the Gap Diffie-Hellman assumption [51], our construction's CCA security reduces generically to the PCA security of the the input PKE; in addition, we argue that if the PKE's plaintext space is large and the sampling method has sufficient entropy, then the MAC only needs to be one-time existentially unforgeable (Abdalla, Rogaway, and Bellare originally proposed to use HMAC and CBC-MAC, which are many-time secure MAC but less efficient than one-time MAC). The data flow of the encapsulation is illustrated in Figure 6

In Section 3.1 we will describe the encrypt-then-MAC KEM routines and state the security reduction. In Section 3.2 we present the proof reducing 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 3.3 we discuss how the OW-PCA security of the PKE relates to the CCA security of the encrypt-then-MAC KEM. In Section 3.4 we discuss some generic attacks on our KEM transformation.

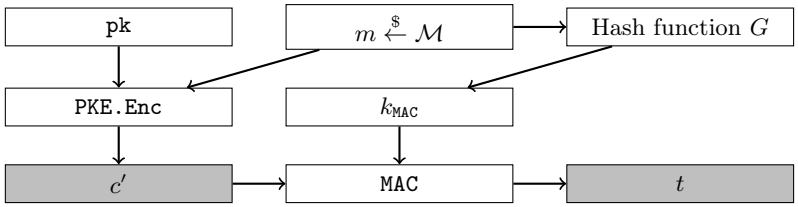

Fig. 6: Combining PKE with MAC using encrypt-then-MAC to ensure ciphertext integrity

3.1 The construction

Let $\mathcal{B}^*$ denote the set of finite bit strings. Let $\mathcal{K}_{\mathtt{KEM}}$ denote the set of all possible shared secrets. Let $(\mathtt{KeyGen}_{\mathtt{PKE}}, \mathtt{Enc}_{\mathtt{PKE}}, \mathtt{Dec}_{\mathtt{PKE}})$ be a PKE defined over message space $\mathcal{M}_{\mathtt{PKE}}$ and ciphertext space $\mathcal{C}_{\mathtt{PKE}}$. Let $\mathtt{MAC} : \mathcal{K}_{\mathtt{MAC}} \times \mathcal{B}^* \to \mathcal{T}$ be a MAC over key space $\mathcal{K}_{\mathtt{MAC}}$ and tag space $\mathcal{T}$. Let $G : \mathcal{B}^* \to \mathcal{K}_{\mathtt{MAC}}, H : \mathcal{B}^* \to \mathcal{K}_{\mathtt{KEM}}$ be hash functions. The encrypt-then-MAC transformation $\mathtt{EtM}[\mathtt{PKE}, \mathtt{MAC}, G, H]$ constructs a KEM $(\mathtt{KeyGen}_{\mathtt{EtM}}, \mathtt{Encap}_{\mathtt{EtM}}, \mathtt{Decap}_{\mathtt{EtM}})$ (Figure 7).

$\mathtt{KeyGen}_{\mathtt{EtM}}()$

1: $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}_{\mathtt{PKE}}()$
2: $s \xleftarrow{\$} \mathcal{M}_{\mathtt{PKE}}$
3: $\mathtt{sk} \leftarrow (\mathtt{sk}, s)$
4: **return** $(\mathtt{pk}, \mathtt{sk})$

$\mathtt{Encap}_{\mathtt{EtM}}(\mathtt{pk})$

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

$\mathtt{Decap}_{\mathtt{EtM}}(\mathtt{sk}, c)$

Require: $c = (c', t)$
Require: $\mathtt{sk} = (\mathtt{sk}', s)$
1: $\hat{m} \leftarrow \mathtt{Dec}_{\mathtt{PKE}}(\mathtt{sk}', c')$
2: $\hat{k} \leftarrow G(\hat{m})$
3: **if** $\mathtt{MAC}(\hat{k}, c') = t$ **then**
4: $\quad K \leftarrow H(\hat{m}, c)$
5: **else**
6: $\quad K \leftarrow H(s, c)$
7: **end if**
8: **return** K

Fig. 7: The encrypt-then-MAC KEM routines

We chose to construct $\mathtt{KEM}_{\mathtt{EtM}}$ using implicit rejection $K \leftarrow H(s, c)$: on invalid ciphertexts, the decapsulation routine returns a fake shared secret that depends on the ciphertext and some secret values, though choosing to use explicit rejection should not impact the security of the KEM. In addition, because the underlying PKE can be randomized, the shared secret $K \leftarrow H(m, c)$ must depend on both the plaintext and the ciphertext. According to [23,34], if the input PKE is *rigid* (i.e. $m = \mathtt{Dec}(\mathtt{sk}, c)$ if and only if $c = \mathtt{Enc}(\mathtt{pk}, m)$), such as with RSA, then the shared secret may be derived from the plaintext alone $K \leftarrow H(m)$.

The CCA security of $\texttt{KEM}_{\texttt{EtM}}$ can be intuitively argued through an adversary's inability to learn additional information from the decapsulation oracle. For an adversary A to produce a valid tag for some unauthenticated ciphertext c', it must either know the correct symmetric key or produce a forgery. Under the Random Oracle Model (ROM), A cannot know the symmetric key without knowing its pre-image under the hash function G, so A must either produced c' honestly, or have broken the one-wayness of the underlying PKE. This means that the decapsulation oracle will not leak information on decryption that the adversary does not already know. We formalize the security in Theorem 1

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

$$\mathbf{Adv}^{\textit{IND-CCA}}_{\textit{KEM}_{\texttt{EtM}}}(A) \leq q \cdot \mathbf{Adv}_{\textit{MAC}}(C) + 2 \cdot \mathbf{Adv}^{\textit{OW-PCA}}_{\textit{PKE}}(B)$$

3.2 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 $\texttt{KEM}_{\texttt{EtM}}$:

1. Replace the true decapsulation oracle with a simulated decapsulation oracle. The simulated decapsulation oracle does not directly decrypt the queried ciphertext. Instead it searches through the hash oracle and looks for matching queries using the plaintext-checking oracle. The true decapsulation oracle and the simulated decapsulation oracle disagree if and only if the adversary queries with a ciphertext that contains a forged MAC tag, so the adversary cannot distinguish the two games more than it can perform existential forgery against the underlying MAC.
2. Replace the pseudorandom MAC key $k^* \leftarrow G(m^*)$ with a uniformly random MAC key $k^* \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$. Under ROM, the adversary cannot distinguish this game from the previous one unless it queries the hash oracle with m^*, in which case a second adversary with access to a plaintext-checking oracle can win the OW-PCA game against the underlying PKE.
3. Replace the pseudorandom shared secret $K_0 \leftarrow H(m^*, c)$ with a truly random shared secret $K_0 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$. Similar to the point above, the adversary cannot distinguish the change more than it can break the one-wayness of the underlying PKE. Furthermore, since both K_0 and K_1 are uniformly random, no adversary can have any advantage.

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.

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

$(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}_{\texttt{EtM}}()$
$m^* \xleftarrow{\$} \mathcal{M}$
$c' \xleftarrow{\$} \texttt{Enc}_{\texttt{PKE}}(\texttt{pk}, m^*)$
$k^* \leftarrow G(m^*)$ ▷ Game 0-1
$k^* \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$ ▷ Game 2-3
$t \leftarrow \texttt{MAC}(k^*, c')$
$c^* \leftarrow (c', t)$
$K_0 \leftarrow H(m^*, c^*)$ ▷ Game 0-2
$K_0 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$ ▷ Game 3
$K_1 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$
$b \xleftarrow{\$} \{0, 1\}$
$\hat{b} \leftarrow A^{\mathcal{O}^{\texttt{Decap}}}(\texttt{pk}, c^*, K_b)$ ▷ Game 0
$\hat{b} \leftarrow A^{\mathcal{O}_1^{\texttt{Decap}}}(\texttt{pk}, c^*, K_b)$ ▷ Game 1-3
return $[\![\hat{b} = b]\!]$

Hash oracle $\mathcal{O}^G(m)$

if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \tilde{m} = m$ **then**
  return $\tilde{k}$
end if
$k \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$
$\mathcal{L}^G \leftarrow \mathcal{L}^G \cup \{(m, k)\}$
return k

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

$(c', t) \leftarrow c$
$\hat{m} = \texttt{Dec}_{\texttt{PKE}}(\texttt{sk}', c')$
$\hat{k} \leftarrow G(\hat{m})$
if $\texttt{MAC}(\hat{k}, c') = t$ **then**
  $K \leftarrow H(\hat{m}, c)$
else
  $K \leftarrow H(z, c)$
end if
return K

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

$(c', t) \leftarrow c$
if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \tilde{m} = \texttt{Dec}_{\texttt{PKE}}(\texttt{sk}', c') \wedge \texttt{MAC}(\tilde{k}, c') = t$ **then**
  $K \leftarrow H(\tilde{m}, c)$
else
  $K \leftarrow H(z, c)$
end if
return K

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

if $\exists(\tilde{m}, \tilde{c}, \tilde{K}) \in \mathcal{L}^H : \tilde{m} = m \wedge \tilde{c} = c$ **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

Fig. 8: Sequence of games in the proof of Theorem 1

Proof. *Game 0* is the standard KEM IND-CCA game. 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

$B(\mathtt{pk}, c'^*)$

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

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

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

$\mathcal{O}_B^{\mathtt{Decap}}(c)$

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

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

1: **if** $\mathcal{O}^{\mathtt{PCO}}(m, c'^*) = 1$ **then**
2: $\mathtt{ABORT}(m)$
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: $k \xleftarrow{\$} \mathcal{K}_{\mathtt{MAC}}$
8: $\mathcal{L}^G \leftarrow \mathcal{L}^G \cup \{(m, k)\}$
9: **return** k

Fig. 9: OW-PCA adversary B simulates game 3 for IND-CCA adversary A in the proof for Theorem 1

the queried ciphertext. On the other hand, if the true oracle rejects the queried ciphertext, then the tag is simply invalid under the MAC key $k = G(\mathtt{Dec}(\mathtt{sk}', c'))$. 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 $k = G(\mathtt{Dec}(\mathtt{sk}', c'))$, but k has never been queried. Under the random oracle model, such k is a uniformly random sample of $\mathcal{K}_{\mathtt{MAC}}$ that the adversary does not know, so for A to produce a valid tag is to produce a forgery against the $\mathtt{MAC}$ under an unknown and uniformly random key. Therefore, we can bound the probability that the true decapsulation oracle disagrees with the simulated oracle by the probability that some MAC adversary produces a forgery:

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

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:

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

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 t. 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: upon 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:

$$P\left[\texttt{QUERY G}\right] \leq \texttt{Adv}_{\texttt{PKE}}^{\texttt{OW-PCA}}(B),$$
$$P\left[\texttt{QUERY H}\right] \leq \texttt{Adv}_{\texttt{PKE}}^{\texttt{OW-PCA}}(B).$$

Combining all equations above produce the desired security bound.

3.3 OW-PCA security

The security notion of *One-wayness under plaintext-checking attack (OW-PCA)* was introduced by Okamoto and Pointcheval in [52], where the authors reduced the security of a generic CCA secure transformation (REACT) to the OW-PCA security of the input public-key cryptosystem. Following REACT, Pointcheval et al. proposed GEM [20], another generic CCA secure transformation whose security reduces to the OW-PCA security of the underlying PKE. Around the

time REACT and GEM were published, the best known CCA secure transformation is Optimal Asymmetric Encryption Padding (OAEP)[6]. Compared to OAEP's requirement for one-way trapdoor permutation, OW-PCA security is easier to achieve: any one-way trapdoor permutation (such as RSA) is automatically OW-PCA secure, while there are cryptosystems that are OW-PCA secure but not one-way trapdoor permutation (Table 1). More recently, the modular Fujisaki-Okamoto transformation [34] proposed CCA secure KEM whose security reduces to the OW-PCA security of the input PKE under both ROM and QROM.

Theorem 1 stated that if the underlying PKE is OW-PCA secure, then the encrypt-then-MAC KEM is IND-CCA secure. Conversely, if the underlying PKE is not OW-PCA secure, then the encrypt-then-MAC KEM is NOT IND-CCA secure. This is captured in Lemma 1

Lemma 1. *For every OW-PCA adversary A against the underlying PKE, there exists an IND-CCA adversary B against the encrypt-then-MAC KEM such that:*

$$\mathit{Adv}^{\mathit{IND\text{-}CCA}}_{\mathit{KEM}_{\mathit{EtM}}}(B) = \mathit{Adv}^{\mathit{OW\text{-}PCA}}_{\mathit{PKE}}(A)$$

For a sketch of proof, we observe that the IND-CCA adversary B can perfectly simulate the plaintext-checking oracle for the OW-PCA adversary A (Figure 10), and if A succeeds in breaking the one-wayness of the underlying PKE, then B can compute the shared secret associated with the challenge ciphertext, which allows B to distinguish true shared secret from random bit strings.

$\mathcal{O}^{\mathsf{PCO}}_{\mathsf{Decap}}(m, c)$

1: $k \leftarrow G(m)$
2: $t \leftarrow \mathtt{MAC}(k, c)$
3: $K \leftarrow H(m, c)$
4: **return** $[\![\mathcal{O}^{\mathsf{Decap}}(c, t) = K]\!]$

Fig. 10: Plaintext-checking oracle can be simulated using decapsulation oracle. If m is the decryption of c, then m will hash into the correct MAC key and produce the correct tag, and the decapsulation will accept (c, t) and return the true shared secret. If m is not the decryption of c, then the probability of producing the correct tag is negligible, and the decapsulation will reject (c, t) as invalid.

For the remainder of this section, we will review the OW-PCA security of some well-known cryptosystems. A summary can be found in Table 1.

Number-theoretic cryptosystems The RSA cryptosystem as it was originally proposed in [54] is OW-PCA secure. This is because the RSA cryptosystem

PKE	is it OW-PCA?
RSA	Yes [23,57]
ElGamal	If Gap Diffie-Hellman assumption holds [3,51]
FrodoKEM	No: full key recovery [30,4]
Kyber/Saber	No, full key recovery [36,61,32]
NTRU	No, full key recovery [33,38,59]
HQC	No, full key recovery [36,4]
BIKE (with MDPC)	No, partial key recovery [31]
Classic McEliece (binary Goppa code)	Unknown, but there is no known attack [59]

Table 1: The landscape of OW-PCA security

is a one-way trapdoor permutation [26], meaning that encryption and decryption are both injective. Given some public key $(N = pq, e)$ and a plaintext-ciphertext pair (m, c), one can check that m is the decryption of c by checking that $c \equiv m^e \mod N$, which renders a plaintext-checking oracle completely useless.

The ElGamal cryptosystem [27] is OW-PCA secure if the Gap Diffie-Hellman assumption [51] holds for the underlying group (finite field or elliptic curve). This is the basis on which the CCA security of DHIES [3] is proved. We will discuss in details in Section 5.

Lattice-based cryptosystems In many prominent cryptosystems based on the Learning with Error (LWE) problems, the plaintext is encrypted by adding noise. If an adversary adds additional noise to the ciphertext, the modified ciphertext may or may not decrypt back to the same plaintext. Querying the plaintext-checking oracle with modified ciphertexts containing varying amount of noise thus allows an adversary to recover the secret key. Such key-recovery plaintext-checking attacks (KR-PCA) were described for FrodoKEM in [30,4]. Similar attacks for Kyber and Saber were described in [36,61,32]. As Peikert pointed out in [53], because of the search-decision equivalence of the (Ring) Learning With Error problem, lattice-based cryptosystems are unlikely to have inherent OW-PCA security.

NTRU is another family of lattice-based cryptosystems. Hoffstein and Silverman [33] proposed KR-PCA for the original NTRU cryptosystem, which Ueno et al. [59] adapted into a KR-PCA against modern instantiations such as NTRU-HPS and NTRU-HRSS (also see [63]). Ueno et al. also adapted [38] into a KR-PCA against NTRU-Prime.

Code-based cryptosystems HQC, despite being based on hard coding problems, has a structure that is similar to the lattice cryptosystems. Consequently, KR-PCA for lattice cryptosystems can be easily adapted to work on HQC [36,4]. BIKE is based on the Niederreiter cryptosystem instantiated with quasi-cyclic moderate density parity check (QC-MDPC) code [45], and while [31] described

KR-PCA against QC-MDPC code, such attack can partially recover BIKE secret keys. There is no known adaptive attack against Classic McEliece [59].

3.4 Additional implementation notes

Securely deriving MAC key Because the MAC key is pseudorandomly derived from the PKE plaintext (instead of uniformly and independently sampled), it is possible to construct a large lookup table mapping each PKE plaintext to a MAC key, then check the KEM ciphertext (c, t) against each MAC key to recover the pre-image. As was pointed out in [9,8], this relationship between the MAC tag and the PKE plaintext could damage the security of the scheme. This can be mitigated by including the PKE public key when deriving the MAC key $k_{\mathtt{MAC}} \leftarrow G(\mathtt{pk}, m)$ or even including random salt at each encapsulation $k_{\mathtt{MAC}} \leftarrow G(\mathtt{pk}, m, \text{salt})$, which can increase the cost of such dictionary attacks.

One-time MAC When the encrypt-then-MAC KEM is instantiated with a PKE with a sufficiently large plaintext space, we expect the probability of two encapsulations sampling the same PKE plaintext to be negligible within some reasonable keypair lifetime, and if the hash function is collision resistant, then the probability of two encapsulations deriving the same MAC key is also negligible. Therefore, we speculate that the encrypt-then-MAC KEM can be instantiated with one-time MACs [18] with no security impact, while one-time MACs can be computationally more efficient than many-time secure MACs such as CBC-MAC and HMAC (as used in DHIES [3]).

Deriving shared secret If the decryption routine of the underlying PKE is not injective, then the shared secret must be derived from both the PKE plaintext and the ciphertext: if the shared secret is derived from the plaintext alone, and the KEM adversary A can find a modified ciphertext that decrypts back to the same PKE plaintext, then A can query the decapsulation oracle with the modified ciphertext and obtain the true decapsulation. However, the shared secret does not have to be derived from hashing the entire ciphertext. Instead, we propose to derive the shared secret from the MAC tag, which is functionally equivalent to a keyed hash of the ciphertext. Since the MAC tag is usually much smaller than the ciphertext, hashing the tag instead of the entire ciphertext can lead to meaningful speedup.

4 Applying encrypt-then-MAC to code-based cryptosystem

Code-based cryptography was first introduced by Robert J. McEliece in 1978 [44]. The McEliece cryptosystem samples a random (n, k)-linear code over some finite field $\mathbb{F}$ with generator matrix $G \in \mathbb{F}^{n \times k}$, then generates the public key

by scrambling the generator matrix $\mathtt{pk} = G' \leftarrow PGS$ using a secret permutation matrix $P \in \mathbb{F}^{n\times n}$ and a secret invertible matrix $S \in \mathbb{F}^{k\times k}$. Because P, S are both invertible, G' is also a generator matrix for an (n, k)-linear code. The secret key consists of the decoding routine, the secret permutation matrix, and the secret invertible matrix. A plaintext message m is encrypted by first encoding it using G', then adding an error vector to the codeword: $c \leftarrow G' \cdot m + \mathbf{e}$. To decrypt a ciphertext, the permutation is first removed, then the error vector is removed using the decoding routine, finally the message is recovered. The one-wayness of the McEliece cryptosystem reduces to the NP-hard problem of decoding a random linear code. Harald Niederreiter improved the efficiency of the McEliece cryptosystem in 1986 [50] by replacing the generator matrix of a random linear code with the parity-check matrix. Correspondingly, the plaintext space becomes the set of fixed-weight error vectors, which are encrypted by computing the syndrome under the parity-check matrix $c \leftarrow H\mathbf{e}$. The one-wayness of the Niederreiter variant reduces to the syndrome decoding problem, which is proven NP-hard. Compared to the McEliece formulation, the Niederreiter variant enjoys smaller public key and smaller ciphertexts.

4.1 Classic McEliece

Classic McEliece [12] is an IND-CCA secure post-quantum KEM submitted to the PQC standardization project and is currently one of three viable fourth-round KEM candidates. Classic McEliece is constructed in two layer. The first layer is a OW-CPA secure PKE based on the Niederreiter cryptosystem using a random binary Goppa code, and the second layer is a modified Fujisaki-Okamoto transformation. Each instance of Classic McEliece is parameterized by the base field size m (which induces a finite field with order $q = 2^m$), the codeword size n, and the error vector weight t.

Algorithm 1 describes the key generation routine of Classic McEliece. In step 4, `FieldOrdering` takes the input bits and outputs either $\perp$ or a sequence $(\alpha_0, \ldots, \alpha_{q-1})$ distinct elements of $\mathbb{F}_q$. In step 5, `Irreducible` takes the input bits and outputs either $\perp$ or a monic irreducible degree-t polynomial $g \in \mathbb{F}_q[x]$. In step 7, `MatGen` outputs either $\perp$ or an $mt \times k$ matrix over $\mathbb{F}_2$. We refer readers to [12] for details of these subroutines.

Algorithms 2 and 3 describe the encoding and decoding subroutines of the Niederreiter layer. Note that our description of the decoding routine differs from [12] in that we did not include the weight and/or re-encryption check. Instead, they are included in the description of the KEM subroutines. This is because we later plan to substitute these checks with computing a MAC tag when applying the encrypt-then-MAC KEM transformation.

Algorithms 4 and 5 describe Classic McEliece's KEM construction using the CPA secure Niedereitter cryptosystem descrbied above. In step 1 of Algorithm 4, `FixedWeight` samples a random vector $\mathbf{e} \in \mathbb{F}_2^n$ with Hamming weight t. In this KEM construction, Classic McEliece makes use of re-encryption (step 3 of Algorithm 5) to ensure the integrity of the ciphertext.

Algorithm 1 `SeededKeyGen`(δ)

Require: l-bit seed δ
1: Expand δ to $n + \sigma_2 q + \sigma_1 t + l$ bits, where $\delta_1 = 16, \delta_2 = 32$
2: Denote the last l bits by δ'
3: Denote the first n bits by s
4: Compute $\alpha_0, \ldots, \alpha_{q-1}$ from the next $\sigma_2 q$ bits using the `FieldOrdering` algorithm. If this fails, set $\delta \leftarrow \delta'$ and restart
5: Compute g from the next $\delta_1 t$ bits using the `Irreducible` algorithm. If this fails, set $\delta \leftarrow \delta'$ and restart
6: $\Gamma \leftarrow (g, \alpha_0, \alpha_1, \ldots, \alpha_{n-1})$
7: Compute $(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma') \leftarrow$ `MatGen`(Γ). If this fails, set $\delta \leftarrow \delta'$ and restart
8: Write Γ' as $(g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$
9: Set $\alpha = (\alpha'_0, \ldots, \alpha'_{n-1}, \alpha_n, \ldots, \alpha_{q-1})$ and $c = (c_{mt-\mu}, \ldots, c_{mt-1})$
10: Output T as the public key and $(\delta, c, g, \alpha, s)$ as private key

Algorithm 2 `Encode`$(T, \mathbf{e})$

Require: Public key $T \in \mathbb{F}_2^{mt \times (n-mt)}$
Require: Weight-t vector $\mathbf{e} \in \mathbb{F}_2^n$
1: Define $H = (I_{mt} \mid T)$
2: Compute and return $\mathbf{c} \leftarrow H\mathbf{e} \in \mathbb{F}_2^{mt}$

Algorithm 3 `Decode`$(\Gamma, \mathbf{c})$

Require: Secret key $\Gamma = (g, \alpha_0, \ldots, \alpha_{n-1})$
Require: Syndrome $\mathbf{c} \in \mathbb{F}_2^{mt}$
1: Extend $\mathbf{c}$ to $\mathbf{v} \in \mathbb{F}_2^n$ by appending 0's
2: Find the unique Goppa codeword $\mathbf{w} \in \mathbb{F}_2^n$ such that $H\mathbf{w} = 0$ and $\mathbf{w}$ has Hamming distance no more than t from $\mathbf{v}$. If there is no such $\mathbf{w}$, return $\perp$
3: Set $\mathbf{e} \leftarrow \mathbf{v} + \mathbf{w}$
4: **return** $\mathbf{e}$

Algorithm 4 `Encap`$(\mathtt{pk})$

Require: Public key $\mathtt{pk} = T \in \mathbb{F}_2^{mt \times (n-mt)}$
1: Use `FixedWeight` to generate a vector $\mathbf{e} \in \mathbf{F}_2^n$ with Hamming weight t
2: Compute $\mathbf{c} \leftarrow$ `Encode`$(T, \mathbf{e})$
3: Compute $K \leftarrow H(1, \mathbf{e}, \mathbf{c})$
4: **return** $(\mathbf{c}, K)$

Algorithm 5 `Decap`$(\mathtt{sk}, \mathbf{c})$

Require: Secret key `sk` contains $s \in \mathbb{F}_2^n$ and $\Gamma = (g, \alpha_0, \alpha_1, \ldots, \alpha_{n-1})$
Require: Ciphertext $\mathbf{c} \in \mathbb{F}_2^{mt}$
1: Set $b \leftarrow 1$
2: Compute $\mathbf{e} \leftarrow$ `Decode`$(\Gamma, \mathbf{c})$
3: **if** $wt(\mathbf{e}) \neq t \vee H\mathbf{e} \neq \mathbf{c}$ **then**
4: Set $\mathbf{e} \leftarrow s, b \leftarrow 0$
5: **end if**
6: **return** $K \leftarrow H(b, \mathbf{e}, \mathbf{c})$

We instantiated the encrypt-then-MAC KEM transformation using `SeededKeyGen`, `FixedWeight`, `Encode`, and `Decode` as subroutines. The result is called McElieceEtM. Its key generation routine is identical to the key generation of Classic McEliece. Its encapsulation derives a MAC key from the output of `FixedWeight` and signs the syndrome computed from `Encode`. In decapsulation, weight and syndrome check is replaced with re-deriving the MAC key and checking the tag.

Algorithm 6 $\texttt{Encap}_{\text{McElieceEtM}}(\texttt{pk})$

Require: Public key $\texttt{pk} = T \in \mathbb{F}_2^{mt \times n}$
1: $\mathbf{e} \xleftarrow{\$} \texttt{FixedWeight}()$
2: $k \leftarrow G(\mathbf{e})$
3: $\mathbf{c} \leftarrow \texttt{Encode}(T, \mathbf{e})$
4: $t \leftarrow \texttt{MAC}(k, c)$
5: $K \leftarrow H(\mathbf{e}, \mathbf{c})$
6: **return** $(\mathbf{c}, K)$

Algorithm 7 $\texttt{Decap}_{\text{McElieceEtM}}(\texttt{sk}, c)$

Require: Secret key $\texttt{sk} = (\Gamma, s)$
Require: Ciphertext $c = (\mathbf{c}, t)$
1: $\hat{\mathbf{e}} \leftarrow \texttt{Decode}(\Gamma, \mathbf{c})$
2: $\hat{k} \leftarrow G(\hat{\mathbf{e}})$
3: **if** $\texttt{MAC}(\hat{k}, \mathbf{c}) \neq t$ **then**
4: $\quad K \leftarrow H(s, \mathbf{c})$
5: **else**
6: $\quad K \leftarrow H(\mathbf{e}, \mathbf{c})$
7: **end if**
8: **return** K

Fig. 11: McElieceEtM: applying encrypt-then-MAC to the Niedereitter cryptosystem

4.2 Choosing MAC

For concrete instantiation of McElieceEtM, we chose four MACs covering a variety of architectures. All MACs are parameterized with a 256-bit key and 128-bit tag, except for KMAC256, which can have variable key and tag length.

Poly1305 [7] and GMAC [46] are both Carter-Wegman style MACs [18,60], which compute the tag using finite field arithmetic. It first parses the message into a sequence of finite field elements, then evalutes a polynomial whose coefficients are the message blocks and whose indeterminate is the secret key. Specifically, Poly1305 oerates in the prime field $\mathbb{F}_q$ where $q = 2^{130} - 5$. GMAC operates in the binary extension field $\mathbb{F}_{2^{128}}$.

In implementation, we used OpenSSL 3.3.1's `EVP_MAC` interface. Within this interface, the Poly1305 implementation does not include a nonce and is thus only one-time secure. On the other hand, GMAC is implemented by passing all data into the "associated data" field of the authenticated encryption scheme AES-256-GCM. Assuming that nonce does not repeat within the lifetime of the symmetric key, GMAC is many-time secure.

CMAC [48] is based on the CBC-MAC [14]. To compute a CMAC tag, the message is first padded and parsed into blocks. Each block is first XOR'ed with the previous block's output, then encrypted under a block cipher using the secret key. The final output is XOR'ed with a sub-key derived from the secret key before being encrypted for one last time. In our implementation, the block cipher is instantiated with AES-256, which makes it particularly suitable for embedded devices with constrainted computing capacity but hardware support for AES. CMAC is many-time secure.

KMAC256 [41] is a pseudorandom function and keyed hash function based on *KECCAK* [47], although unlike SHAKE, altering the requested output length generates a new unrelated output. KMAC256 generally has worse performance than other MACs, but it is the only construction with flexible key and tag length. We fixed the key length at 256 bits, and varied the tag length to 128, 192, and 256 bits depending on the desired security levels.

Remark There are new lightweight MACs such as EliMAC [24], LightMAC [43], and ZMAC [37], which may offer better performance than the MACs listed above. However, the performance of the encrypt-then-MAC KEM is dominated by the public-key cryptographic operations and the key-derivation functions. For KEMs with relatively short ciphertext, such as Classic McEliece, the performance impact of the MAC is insignificant.

4.3 Performance benchmark

The subroutines `SeededKeyGen`, `FixedWeight`, `Encode`, and `Decode` were taken from the `vec` implementation submitted to the fourth round of NIST PQC standardization project[1]. While there are a total of 10 sets of parameters in Classic McEliece (see Section A), we only modified the 5 standard parameter variants. The only difference between the standard variants and the f-variants is the key generation routine, which is not the focus of this paper. Key pairs are fully interoperable between the standard and the f variants, and the encapsulation/decapsulation routines are entirely identical.

MAC implementations are taken from OpenSSL 3.5.0. C code is compiled using Apple Clang 15.0.0. Performance measurement is run on Apple Silicon M1 chip. CPU clock is measured using `mach_absolute_time`. Each routine is run 10000 times, with median times reported in Table 2.

5 Application to ElGamal

Applying the encrypt-then-MAC KEM transformation to the ElGamal cryptosystem results in a construction that is highly similar to DHIES [3]. In this

[1] `https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/mceliece-Round4.tar.gz`

KEM	Enc	Dec	Enc + Dec
mceliece348864	215	2471	2686
mceliece348864 + poly1305	316 (+46.98%)	2074 (-16.07%)	2390 (-11.02%)
mceliece348864 + gmac	335 (+55.81%)	2087 (-15.54%)	2422 (-9.83%)
mceliece348864 + cmac	340 (+58.14%)	2092 (-15.34%)	2432 (-9.46%)
mceliece348864 + kmac256	304 (+41.40%)	2093 (-15.30%)	2397 (-10.76%)

KEM	Enc	Dec	Enc + Dec
mceliece460896	487	6694	7181
mceliece460896 + poly1305	514 (+5.54%)	5784 (-13.59%)	6298 (-12.30%)
mceliece460896 + gmac	565 (+16.02%)	5809 (-13.22%)	6374 (-11.24%)
mceliece460896 + cmac	544 (+11.70%)	5905 (-11.79%)	6449 (-10.19%)
mceliece460896 + kmac256	570 (+17.04%)	5760 (-13.95%)	6330 (-11.85%)

KEM	Enc	Dec	Enc + Dec
mceliece6688128	816	7500	8316
mceliece6688128 + poly1305	889 (+8.95%)	6509 (-13.21%)	7398 (-11.04%)
mceliece6688128 + gmac	890 (+9.07%)	6521 (-13.05%)	7411 (-10.88%)
mceliece6688128 + cmac	900 (+10.29%)	6540 (-12.80%)	7440 (-10.53%)
mceliece6688128 + kmac256	901 (+10.42%)	6546 (-12.72%)	7447 (-10.45%)

KEM	Enc	Dec	Enc + Dec
mceliece6960119	699	7262	7961
mceliece6960119 + poly1305	735 (+5.15%)	6389 (-12.02%)	7124 (-10.51%)
mceliece6960119 + gmac	753 (+7.73%)	6450 (-11.18%)	7203 (-9.52%)
mceliece6960119 + cmac	763 (+9.16%)	6428 (-11.48%)	7191 (-9.67%)
mceliece6960119 + kmac256	765 (+9.44%)	6303 (-13.21%)	7068 (-11.22%)

KEM	Enc	Dec	Enc + Dec
mceliece8192128	858	7464	8322
mceliece8192128 + poly1305	955 (+11.31%)	6547 (-12.29%)	7502 (-9.85%)
mceliece8192128 + gmac	957 (+11.54%)	6550 (-12.25%)	7507 (-9.79%)
mceliece8192128 + cmac	945 (+10.14%)	6546 (-12.30%)	7491 (-9.99%)
mceliece8192128 + kmac256	957 (+11.54%)	6574 (-11.92%)	7531 (-9.50%)

Table 2: McElieceEtM achieves 9-12% speedup in combined "encapsulate + decapsulate" compared to Classic McEliece

section, we will show that the encrypt-then-MAC KEM transformation is a generalization of DHIES by showing that the Gap Diffie-Hellman assumption is a special case of OW-PCA security. 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:*

$$Adv_{GapDH}(B) = Adv^{\texttt{OW-PCA}}_{ElGamal}(A).$$

Each ElGamal cryptosystem [27] is parameterized by a cyclic group $G = \langle g \rangle$ of prime order $q > 2$. A summary of the routine is shown in Figure 12:

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

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

`Dec`$(\mathtt{sk}, c)$
Require: $\mathtt{sk} = x \in \mathbb{Z}_q$
Require: $c = (w, v) \in G \times G$
1: $\hat{m} \leftarrow (w^x)^{-1} \cdot v$
2: **return** $\hat{m}$

Fig. 12: ElGamal cryptosystem over cyclic group $G = \langle g \rangle$ of prime order q

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

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

Definition 2 (decisional Diffie-Hellman problem). *Let $x, y, z \xleftarrow{\$} \mathbb{Z}_q$ be uniformly random samples. Let $h \xleftarrow{\$} \{g^z, g^{xy}\}$ 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 [3] (and later extensively studied in [51]) 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 $x, y \xleftarrow{\$} \mathbb{Z}_q$ be uniformly random samples. Given (g, g^x, g^y) and a restricted DDH oracle $\mathcal{O}^{DDH} : (u, v) \mapsto [\![u^x = v]\!]$, compute g^{xy}.*

We now present the proof for Lemma 2.

Proof. We will prove by a sequence of games. A summary can be found in Figure 13

$G_0 - G_2$

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

$\mathcal{O}^{\mathsf{PCO}}(m, c = (w, v))$

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

$\mathcal{O}_1^{\mathsf{PCO}}(m, c = (w, v))$

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

Fig. 13: The sequence of games in proving Lemma 2

Game 0 is the OW-PCA game. Adversary A has access to the plaintext-checking oracle $\mathcal{O}^{\mathsf{PCO}}$ 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 $\mathcal{O}^{\mathsf{PCO}}$ is changed. When servicing the plaintext-checking query $(m, c = (w, v))$, PCO_1 checks whether w^x is equal to $m^{-1} \cdot v$. Observe that in the cyclic group G, the algebraic expressions in $\mathcal{O}^{\mathsf{PCO}}$ and $\mathcal{O}_1^{\mathsf{PCO}}$ are equivalent, which means that $\mathcal{O}_1^{\mathsf{PCO}}$ behaves identically to $\mathcal{O}^{\mathsf{PCO}}$.

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 $\hat{m} = w^{-x} \cdot v$. It is easy to verify that Game 0 through Game 2 are algebraically equivalent:

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

The Gap Diffie-Hellman adversary B can perfectly simulate game 2 for A (see Figure 14): 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 $\mathcal{O}_1^{\mathsf{PCO}}$ from game 2 can be perfectly simulated using the restricted DDH oracle $\mathcal{O}^{\mathrm{DDH}}$.

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:

$B^{\mathcal{O}^{\mathrm{DDH}}}(g, g^x, g^y)$

1: $w \leftarrow g^y$
2: $v \xleftarrow{\$} G$
3: $c^* \leftarrow (w, v)$
4: $\hat{m} \xleftarrow{\$} A^{\mathtt{PCO}_2}(g^x, c^*)$
5: **return** $\hat{m}^{-1} \cdot v$

$\mathcal{O}^{\mathrm{DDH}}(u, v)$

1: **return** $[\![u^x = v]\!]$

$\mathcal{O}_2^{\mathtt{PCO}}(m, c = (w, v))$

1: **return** $\mathcal{O}^{\mathrm{DDH}}(w, m^{-1} \cdot v)$

Fig. 14: Gap Diffie-Hellman adversary B simulates game 2 for A

$$\mathtt{Adv}_2(A) = \mathtt{Adv}_{\mathrm{GapDH}}(B).$$

6 Concluding remarks

In this paper we presented "encrypt-then-MAC", a KEM constructed from a PKE and a MAC. We reduced the IND-CCA security of the KEM tightly to the OW-PCA security of the underlying PKE and non-tightly to the security of the underlying MAC. We also analyzed generic attacks on the KEM and proposed countermeasures. We then applie the encrypt-then-MAC transformation to the ElGamal cryptosystem and the McEliece cryptosystem, and implemented the transformed McElieceEtM in C using a variety of MACs. On average, McElieceEtM achieved 9-12% increase in throughput compared to Classic McEliece, which uses the Fujisaki-Okamoto transformation to convert OW-CPA encryption scheme into a CCA secure KEM.

In Section 3.3 we mentioned that Kyber/ML-KEM's underlying PKE subroutines are not OW-PCA secure, which means that applying the encrypt-then-MAC KEM transformation to the PKE subroutines will not achieve full IND-CCA security. However, the plaintext-checking attack against Kyber requires several thousands of oracle queries, which translates to equivalent complexity of any chosen-ciphertext attacks. Therefore, it may be safe to apply the encrypt-then-MAC transformation to Kyber/ML-KEM with the restriction that each key pair is used at most once, such as in TLS 1.3 handshake. In Kyber/ML-KEM, the encryption routine is significantly slower than the decryption routine, so the encrypt-then-MAC transformation achieves significant performance improvements over the Fujisaki-Okamoto transformation (See Appendix B).

References

1. ISO/IEC 9796: Information Technology — Security Techniques — Digital Signature Scheme Giving Message Recovery, Part 1: Mechanisms Using Redundancy (1999), part 1 of the ISO/IEC 9796 standard

2. Abdalla, M., Bellare, M., Rogaway, P.: DHAES: an encryption scheme based on the Diffie-Hellman problem. IACR Cryptol. ePrint Arch. p. 7 (1999), http://eprint.iacr.org/1999/007
3. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer (2001). https://doi.org/10.1007/3-540-45353-9_12, https://doi.org/10.1007/3-540-45353-9_12
4. Baetu, C., Durak, F.B., Huguenin-Dumittan, L., Talayhan, A., Vaudenay, S.: Misuse attacks on post-quantum cryptosystems. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 747–776. Springer (2019). https://doi.org/10.1007/978-3-030-17656-3_26, https://doi.org/10.1007/978-3-030-17656-3_26
5. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) 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. Lecture Notes in Computer Science, vol. 1976, pp. 531–545. Springer (2000). https://doi.org/10.1007/3-540-44448-3_41, https://doi.org/10.1007/3-540-44448-3_41
6. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: Santis, A.D. (ed.) Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings. Lecture Notes in Computer Science, vol. 950, pp. 92–111. Springer (1994). https://doi.org/10.1007/BFB0053428, https://doi.org/10.1007/BFb0053428
7. Bernstein, D.J.: The Poly1305-AES message authentication code. In: Gilbert, H., Handschuh, H. (eds.) Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3557, pp. 32–49. Springer (2005). https://doi.org/10.1007/11502760_3, https://doi.org/10.1007/11502760_3
8. Bernstein, D.J.: FO derandomization sometimes damages security. Cryptology ePrint Archive, Paper 2021/912 (2021), https://eprint.iacr.org/2021/912
9. Bernstein, D.J.: On the looseness of FO derandomization. IACR Cryptol. ePrint Arch. p. 912 (2021), https://eprint.iacr.org/2021/912
10. Bernstein, D.J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Wang, W., Albrecht, M., Cid, C., Paterson, K.G., Tjhai, C.J., Tomlinson, M.: Classic McEliece: Conservative code-based cryptography: Guide for security reviewers (October 23 2022), https://classic.mceliece.org/mceliece-security-20221023.pdf
11. Bernstein, D.J., Chou, T., Schwabe, P.: Mcbits: Fast constant-time code-based cryptography. In: Bertoni, G., Coron, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8086, pp. 250–272. Springer (2013). https://doi.org/10.1007/978-3-642-40349-1_15, https://doi.org/10.1007/978-3-642-40349-1_15

12. Bernstein, D.J., Heninger, N., Lange, T., van Beirendonck, M., et al.: Classic mceliece: Specification (October 2022), https://classic.mceliece.org/mceliece-spec-20221023.pdf, accessed: 2025-01-29
13. Bernstein, D.J., Persichetti, E.: Towards KEM unification. IACR Cryptol. ePrint Arch. p. 526 (2018), https://eprint.iacr.org/2018/526
14. Black, J., Rogaway, P.: CBC MACs for arbitrary-length messages: The three-key constructions. In: Bellare, M. (ed.) Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1880, pp. 197–215. Springer (2000). https://doi.org/10.1007/3-540-44598-6_12, https://doi.org/10.1007/3-540-44598-6_12
15. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: Krawczyk, H. (ed.) Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1462, pp. 1–12. Springer (1998). https://doi.org/10.1007/BFB0055716, https://doi.org/10.1007/BFb0055716
16. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 1006–1018. ACM (2016). https://doi.org/10.1145/2976749.2978425, https://doi.org/10.1145/2976749.2978425
17. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: 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. pp. 353–367. IEEE (2018). https://doi.org/10.1109/EUROSP.2018.00032, https://doi.org/10.1109/EuroSP.2018.00032
18. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979). https://doi.org/10.1016/0022-0000(79)90044-8, https://doi.org/10.1016/0022-0000(79)90044-8
19. Chou, T.: Mcbits revisited. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 213–231. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_11, https://doi.org/10.1007/978-3-319-66787-4_11
20. Coron, J., Handschuh, H., Joye, M., Paillier, P., Pointcheval, D., Tymen, C.: GEM: A generic chosen-ciphertext secure encryption method. In: Preneel, B. (ed.) Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2271, pp. 263–276. Springer (2002). https://doi.org/10.1007/3-540-45760-7_18, https://doi.org/10.1007/3-540-45760-7_18
21. Coron, J., Naccache, D., Stern, J.P.: On the security of RSA padding. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 1–18. Springer (1999). https://doi.org/10.1007/3-540-48405-1_1, https://doi.org/10.1007/3-540-48405-1_1

22. D'Anvers, J., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10831, pp. 282–305. Springer (2018). https://doi.org/10.1007/978-3-319-89339-6_16, https://doi.org/10.1007/978-3-319-89339-6_16
23. Dent, A.W.: A designer's guide to KEMs. In: Paterson, K.G. (ed.) Cryptography and Coding, 9th IMA International Conference, Cirencester, UK, December 16-18, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2898, pp. 133–151. Springer (2003). https://doi.org/10.1007/978-3-540-40974-8_12, https://doi.org/10.1007/978-3-540-40974-8_12
24. Dobraunig, C., Mennink, B., Neves, S.: EliMAC: Speeding up LightMAC by around 20%. IACR Trans. Symmetric Cryptol. **2023**(2), 69–93 (2023). https://doi.org/10.46586/TOSC.V2023.I2.69-93, https://doi.org/10.46586/tosc.v2023.i2.69-93
25. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 537–554. Springer (1999). https://doi.org/10.1007/3-540-48405-1_34, https://doi.org/10.1007/3-540-48405-1_34
26. Fujisaki, E., Okamoto, T., Pointcheval, D., Stern, J.: RSA-OAEP is secure under the RSA assumption. In: Kilian, J. (ed.) Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2139, pp. 260–274. Springer (2001). https://doi.org/10.1007/3-540-44647-8_16, https://doi.org/10.1007/3-540-44647-8_16
27. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074, https://doi.org/10.1109/TIT.1985.1057074
28. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: Lewis, H.R., Simons, B.B., Burkhard, W.A., Landweber, L.H. (eds.) Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA. pp. 365–377. ACM (1982). https://doi.org/10.1145/800070.802212, https://doi.org/10.1145/800070.802212
29. Grieu, F.: A chosen messages attack on the ISO/IEC 9796-1 signature scheme. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding. Lecture Notes in Computer Science, vol. 1807, pp. 70–80. Springer (2000). https://doi.org/10.1007/3-540-45539-6_5, https://doi.org/10.1007/3-540-45539-6_5
30. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. Cryptology ePrint Archive, Paper 2020/743 (2020), https://eprint.iacr.org/2020/743
31. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and

Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 789–815 (2016). https://doi.org/10.1007/978-3-662-53887-6_29, https://doi.org/10.1007/978-3-662-53887-6_29

32. Guo, Q., Mårtensson, E.: Do not bound to a single position: Near-optimal multi-positional mismatch attacks against Kyber and Saber. In: Johansson, T., Smith-Tone, D. (eds.) Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023, College Park, MD, USA, August 16-18, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14154, pp. 291–320. Springer (2023). https://doi.org/10.1007/978-3-031-40003-2_11, https://doi.org/10.1007/978-3-031-40003-2_11
33. Hoffstein, J., Silverman, J.H.: Reaction attacks against the NTRU public key cryptosystem. Tech. rep., Technical Report 15, NTRU Cryptosystems (1999)
34. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10677, pp. 341–371. Springer (2017). https://doi.org/10.1007/978-3-319-70500-2_12, https://doi.org/10.1007/978-3-319-70500-2_12
35. Hövelmanns, K., Hülsing, A., Majenz, C.: Failing gracefully: Decryption failures and the Fujisaki-Okamoto transform. In: Agrawal, S., Lin, D. (eds.) 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. Lecture Notes in Computer Science, vol. 13794, pp. 414–443. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_15, https://doi.org/10.1007/978-3-031-22972-5_15
36. Huguenin-Dumittan, L., Vaudenay, S.: Classical misuse attacks on NIST round 2 PQC - the power of rank-based schemes. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) Applied Cryptography and Network Security - 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12146, pp. 208–227. Springer (2020). https://doi.org/10.1007/978-3-030-57808-4_11, https://doi.org/10.1007/978-3-030-57808-4_11
37. Iwata, T., Minematsu, K., Peyrin, T., Seurin, Y.: ZMAC: A fast tweakable block cipher mode for highly secure message authentication. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 34–65. Springer (2017). https://doi.org/10.1007/978-3-319-63697-9_2, https://doi.org/10.1007/978-3-319-63697-9_2
38. Jaulmes, É., Joux, A.: A chosen-ciphertext attack against NTRU. In: Bellare, M. (ed.) Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1880, pp. 20–35. Springer (2000). https://doi.org/10.1007/3-540-44598-6_2, https://doi.org/10.1007/3-540-44598-6_2
39. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018,

Proceedings, Part III. Lecture Notes in Computer Science, vol. 10993, pp. 96–125. Springer (2018). https://doi.org/10.1007/978-3-319-96878-0_4, https://doi.org/10.1007/978-3-319-96878-0_4

40. Kaliski, B.: PKCS #1: RSA encryption version 1.5. RFC **2313**, 1–19 (1998). https://doi.org/10.17487/RFC2313, https://doi.org/10.17487/RFC2313
41. Kelsey, J., jen Chang, S., Perlner, R.: SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash. Tech. Rep. NIST Special Publication 800-185, National Institute of Standards and Technology, U.S. Department of Commerce (2016), https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-185.pdf
42. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: Kilian, J. (ed.) Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2139, pp. 310–331. Springer (2001). https://doi.org/10.1007/3-540-44647-8_19, https://doi.org/10.1007/3-540-44647-8_19
43. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC mode for lightweight block ciphers. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 43–59. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_3, https://doi.org/10.1007/978-3-662-52993-5_3
44. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Technical report, NASA (1978), https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF
45. Misoczki, R., Tillich, J., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In: Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013. pp. 2069–2073. IEEE (2013). https://doi.org/10.1109/ISIT.2013.6620590, https://doi.org/10.1109/ISIT.2013.6620590
46. National Institute of Standards and Technology: Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. Tech. Rep. NIST Special Publication 800-38D, U.S. Department of Commerce (2007), https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
47. National Institute of Standards and Technology: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Tech. rep., Federal Information Processing Standards (FIPS) Publication 202 (August 2015). https://doi.org/10.6028/NIST.FIPS.202, http://dx.doi.org/10.6028/NIST.FIPS.202
48. National Institute of Standards and Technology: Recommendation for block cipher modes of operation: The CMAC mode for authentication. Tech. Rep. NIST Special Publication 800-38B, U.S. Department of Commerce (2016), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf
49. National Institute of Standards and Technology (NIST): Post-quantum cryptography standardization: Security evaluation criteria (nd), https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria), accessed: 2025-01-20
50. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Prob. Contr. Inform. Theory **15**(2), 157–166 (1986)

51. Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings. Lecture Notes in Computer Science, vol. 1992, pp. 104–118. Springer (2001). https://doi.org/10.1007/3-540-44586-2_8, https://doi.org/10.1007/3-540-44586-2_8
52. Okamoto, T., Pointcheval, D.: REACT: rapid enhanced-security asymmetric cryptosystem transform. In: Naccache, D. (ed.) Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2020, pp. 159–175. Springer (2001). https://doi.org/10.1007/3-540-45353-9_13, https://doi.org/10.1007/3-540-45353-9_13
53. Peikert, C.: Lattice cryptography for the internet. Cryptology ePrint Archive, Paper 2014/070 (2014), https://eprint.iacr.org/2014/070
54. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342, https://doi.org/10.1145/359340.359342
55. Shoup, V.: Using hash functions as a hedge against chosen ciphertext attack. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding. Lecture Notes in Computer Science, vol. 1807, pp. 275–288. Springer (2000). https://doi.org/10.1007/3-540-45539-6_19, https://doi.org/10.1007/3-540-45539-6_19
56. Shoup, V.: OAEP reconsidered. In: Kilian, J. (ed.) Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2139, pp. 239–259. Springer (2001). https://doi.org/10.1007/3-540-44647-8_15, https://doi.org/10.1007/3-540-44647-8_15
57. Shoup, V.: A proposal for an ISO standard for public key encryption. IACR Cryptol. ePrint Arch. p. 112 (2001), http://eprint.iacr.org/2001/112
58. Tanaka, Y., Ueno, R., Xagawa, K., Ito, A., Takahashi, J., Homma, N.: Multiple-valued plaintext-checking side-channel attacks on post-quantum KEMs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(3), 473–503 (2023). https://doi.org/10.46586/TCHES.V2023.I3.473-503, https://doi.org/10.46586/tches.v2023.i3.473-503
59. Ueno, R., Xagawa, K., Tanaka, Y., Ito, A., Takahashi, J., Homma, N.: Curse of re-encryption: A generic power/em analysis on post-quantum KEMs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 296–322 (2022). https://doi.org/10.46586/TCHES.V2022.I1.296-322, https://doi.org/10.46586/tches.v2022.i1.296-322
60. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. **22**(3), 265–279 (1981). https://doi.org/10.1016/0022-0000(81)90033-7, https://doi.org/10.1016/0022-0000(81)90033-7
61. Xagawa, K., Ito, A., Ueno, R., Takahashi, J., Homma, N.: Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13091, pp. 33–61. Springer (2021).

https : / / doi . org / 10 . 1007 / 978-3-030-92075-3 _ 2, https://doi.org/10.1007/978-3-030-92075-3_2
62. Xagawa, K., Yamakawa, T.: (tightly) QCCA-secure key-encapsulation mechanism in the quantum random oracle model. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers. Lecture Notes in Computer Science, vol. 11505, pp. 249–268. Springer (2019). https : / / doi . org / 10 . 1007 / 978-3-030-25510-7 _ 14, https : / / doi . org / 10 . 1007 / 978-3-030-25510-7_14
63. Zhang, X., Cheng, C., Ding, R.: Small leaks sink a great ship: An evaluation of key reuse resilience of PQC third round finalist NTRU-HRSS. Cryptology ePrint Archive, Paper 2021/168 (2021), https://eprint.iacr.org/2021/168

A Classic McEliece parameter sets

Classic McEliece has a total of 10 parameter sets. Five of them are presented in Table 3. The other five (called "f-variants") are identical to the five presented here, except for that the key generation routines differ from their canonical variants for performance reasons: in fact, the f-variants are fully interoperable with their normal counterparts.

Parameter set	public key	secret key	ciphertext	security level [10]
mceliece348864	261120	6492	96	category 1
mceliece460896	524160	13608	156	category 3
mceliece6688128	1044992	13932	208	category 5
mceliece6960119	1047319	13948	194	category 5
mceliece8192128	1357824	14120	208	category 5

Table 3: Public key, secret key, and ciphertext sizes are measured in bytes.

NIST evaluates concrete security levels against the difficulty of AES brute-force attack and/or SHA-2 collision attacks [49]. There are five categories of concrete securities (in increasing security levels):

1. Key search on AES-128, 2^{143} classical gates.
2. Finding collision on SHA256/SHA3-256, 2^{146} classical gates.
3. Key search on AES-192, 2^{207} classical gates.
4. Finding collision on SHA384/SHA3-384, 2^{210} classical gates.
5. Key search on AES-256, 2^{272} classical gates.

B ML-KEM+: applying encrypt-then-MAC to ML-KEM

We apply the encrypt-then-MAC transformation to the K-PKE routines of ML-KEM. The resulting KEM is called ML-KEM$^+$ (See Figure 15). The key generation routines are identical between ML-KEM$^+$ and ML-KEM. The ML-KEM$^+$

ML-KEM$^+$.KeyGen()

$z \xleftarrow{\$} \{0,1\}^{256}$
$(\mathtt{pk}, \mathtt{sk}') \xleftarrow{\$} \mathtt{K\text{-}PKE.KeyGen}()$
$h \leftarrow H(\mathtt{pk})$
$\mathtt{sk} \leftarrow (sk' \| \mathtt{pk} \| h \| z)$
return $(\mathtt{pk}, \mathtt{sk})$

ML-KEM$^+$.Encap(pk)

$m \xleftarrow{\$} \{0,1\}^{256}$
$(\overline{K}, k) \leftarrow G(m \| H(\mathtt{pk}))$
$c' \xleftarrow{\$} \mathtt{K\text{-}PKE.Enc}(\mathtt{pk}, m)$
$t \leftarrow \mathtt{MAC}(k, c')$
$K \leftarrow \mathtt{KDF}(\overline{K} \| t)$
$c \leftarrow (c', t)$
return (c, K)

ML-KEM$^+$.Decap(sk, c)

Require: Secret key $\mathtt{sk} = (\mathtt{sk}' \| \mathtt{pk} \| h \| z)$
Require: Ciphertext $c = (c' \| t)$
$(\mathtt{sk}', \mathtt{pk}, h, z) \leftarrow \mathtt{sk}$
$(c', t) \leftarrow c$
$\hat{m} \leftarrow \mathtt{K\text{-}PKE.Dec}(\mathtt{sk}', c')$
$(\overline{K}, \hat{k}) \leftarrow G(\hat{m} \| h)$
$\hat{t} \leftarrow \mathtt{MAC}(\hat{k}, c')$
if $\hat{t} = t$ **then**
    $K \leftarrow \mathtt{KDF}(\overline{K} \| t)$
else
    $K \leftarrow \mathtt{KDF}(z \| c)$
end if
return K

Fig. 15: ML-KEM$^+$ applies our encrypt-then-MAC transformation to the K-PKE sub-routines of ML-KEM. G is SHA3-512, H is SHA3-256, KDF is SHAKE256

encapsulation and decapsulation algorithms make use of two additional algorithms, namely a MAC and a KDF.

We measured the CPU cycles of the individual routines of ML-KEM$^+$. Our implementation extended from Kyber's reference implementation [2]. 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.

Compared to the Fujisaki-Okamoto transformation 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 Fujisaki-Okamoto 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.

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

Table 4: CPU cycles of each KEM routine

KEM variant	`Encap` cycles/tick		`Decap` cycles/tick	
	Median	Average	Median	Average
ML-KEM-512	91467	92065	121185	121650
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/ KMAC256	101009	101313	40741	40916

KEM variant	`Encap` cycles/tick		`Decap` cycles/tick	
	Median	Average	Median	Average
ML-KEM-768	136405	147400	186445	187529
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/ KMAC256	155219	155848	52415	52611

KEM variant	`Encap` cycles/tick		`Decap` cycles/tick	
	Median	Average	Median	Average
ML-KEM-1024	199185	199903	246245	247320
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/ KMAC256	216761	217468	62269	62516