Fast Fujisaki-Okamoto transformation using encrypt-then-mac and applications to Kyber

Anonymous Submission

Abstract. The modular Fujisaki-Okamoto (FO) transformation takes public-key encryption with weaker security and constructs a key encapsulation mechanism (KEM) with indistinguishability under adaptive chosen ciphertext attacks. While the modular FO transform enjoys tight security bound and quantum resistance, it also suffers from computational inefficiency and vulnerabilities to side-channel attacks due to using de-randomization and re-encryption for providing ciphertext integrity. In this work, we propose an alternative KEM construction that achieves ciphertext integrity using a message authentication code (MAC) and instantiate a concrete instance using ML-KEM. Our experimental results showed that where the encryption routine incurs heavy computational cost, replacing re-encryption with MAC provides substantial performance improvements at comparable security level.

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

1 Introduction

The Fujisaki-Okamoto transformation [FO99] is a generic construction that takes cryptographic primitives of lesser security and constructs a public-key encryption scheme with indistinguishability under adaptive chosen ciphertext attacks. Later works [HHK17] extended the original transformation to the construction of key encapsulation mechanism, which has been adopted by many post-quantum schemes such as Kyber [BDK$^+$18], FrodoKEM [BCD$^+$16], and SABER [DKSRV18].

The current state of the FO transformation enjoys tight security bound and quantum resistance [HHK17], but also leaves many deficiencies to be improved on. One such shortcoming is the use of re-encryption for providing ciphertext integrity [BP18], which requires the decapsulation routine to run the encryption routine as a subroutine. In many post-quantum schemes, such as Kyber, the encryption routine is substantially more expensive than the decryption routine, so running the encryption routine in the decapsulation routine inflates the computational cost of the decapsulator. In addition, running the encryption as a subroutine introduces risks of side-channel vulnerabilities that may expose the plaintext or the secret key [RRCB19][UXT$^+$22].

The problem of ciphertext integrity was solved in symmetric cryptography: given a semantically secure symmetric cipher and an existentially unforgeable message authentication code, combining them using "encrypt-then-mac" provides authenticated encryption [BN00]. We took inspiration from this strategy and applied a similar technique to transform an IND-CPA secure public-key encryption scheme into an IND-CCA2 secure key encapsulation mechanism. Using a message authentication code for ciphertext integrity replaces the re-encryption step in decryption with the computation of an authenticator, which offers significant performance improvements while maintaining comparable level of security.

The main challenge in applying "encrypt-then-mac" to public-key cryptography is the lack of a pre-shared symmetric key. We proposed to derive the symmetric key by hashing the plaintext message. In section 3, we prove that under the random oracle model, if the input public-key encryption scheme is one-way secure against plaintext-checking attack and the input message authentication code is one-time existentially unforgeable, then the transformed key encapsulation mechanism is IND-CCA2 secure.

In section 4, we instantiate concrete instances of our constructions by combining Kyber with GMAC and Poly1305. Our experimental results showed that replacing re-encryption with computing authenticator leads to significant performance improvements in the decapsulation routine while incurring only minimal overhead in the encapsulation routine.

2 Preliminaries and previous results

2.1 Public-key encryption scheme

A public key encryption scheme $\texttt{PKE}$ is a collection of three routines $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ defined over some message space $\mathcal{M}$ and some ciphertext space $\mathcal{C}$. Where the encryption routine is probabilistic, the source of randomness is denoted by the coin space $\mathcal{R}$.

The encryption routine $\texttt{Enc}(\texttt{pk}, m)$ takes a public key, a plaintext message, and outputs a ciphertext $c \in \mathcal{C}$. Where the encryption routine is probabilistic, specifying a pseudorandom seed $r \in \mathcal{R}$ will make the encryption routine behave deterministically. The decryption routine $\texttt{Dec}(\texttt{sk}, c)$ takes a secret key, a ciphertext, and outputs the decryption $\hat{m}$ if the ciphertext is valid under the given secret key, or the rejection symbol $\perp$ if the ciphertext is invalid.

We discuss the security of a $\texttt{PKE}$ using the sequence of games described in [Sho04]. Specifically, we first define the $\texttt{OW-ATK}$ as they pertain to a public key encryption scheme. In later section we will define the $\texttt{IND-CCA}$ game as it pertains to a key encapsulation mechanism.

Algorithm 1 The $\texttt{OW-ATK}$ 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: $\hat{m} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\texttt{ATK}}}(1^\lambda, \texttt{pk}, c^*)$
5: return $[\![m^* = \hat{m}]\!]$
```

Figure 1: The $\texttt{OW-ATK}$ game

Algorithm 2 $\texttt{PCO}(m \in \mathcal{M}, c \in \mathcal{C})$

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

Figure 2: Plaintext-checking oracle

In the $\texttt{OW-ATK}$ game (see figure 1), an adversary's goal is to recover the decryption of a randomly generated ciphertext. A challenger randomly samples a keypair and a challenge plaintext m^*, encrypts the challenge plaintext $c^* \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m^*)$, then gives $\texttt{pk}$ and c^* to the adversary A. The adversary A, with access to some oracle $\mathcal{O}_{\texttt{ATK}}$, outputs a guess decryption $\hat{m}$. A wins the game if its guess $\hat{m}$ is equal to the challenge plaintext m^*. The *advantage* $\texttt{Adv}_{\texttt{OW-ATK}}$ of an adversary in this game is the probability that it wins the game:

$$\texttt{Adv}_{\texttt{OW-ATK}}(A) = P\left[A(\texttt{pk}, c^*) = m^* | (\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(); m^* \xleftarrow{\$} \mathcal{M}; c^* \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m^*)\right]$$

The capabilities of the oracle $\mathcal{O}_{\texttt{ATK}}$ depends on the choice of security goal $\texttt{ATK}$. Particularly relevant to our result is security against plaintext-checking attack (PCA), for which the adversary has access to a plaintext-checking oracle (PCO) (see figure 2). A PCO takes as input a plaintext-ciphertext pair (m, c) and returns `True` if m is the decryption of c or `False` otherwise.

2.2 Key encapsulation mechanism (KEM)

A key encapsulation mechanism is a collection of three routines $(\texttt{KeyGen}, \texttt{Encap}, \texttt{Decap})$ 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 $(\texttt{pk}, \texttt{sk}) \overset{\$}{\leftarrow} \texttt{KeyGen}(1^\lambda)$. $\texttt{Encap}(\texttt{pk})$ 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.

The IND-CCA security of a KEM is defined by an adversarial game in which an adversary's goal is to distinguish pseudorandom shared secret (generated by running the `Encap` routine) and a truly random value.

Algorithm 3 `IND-CCA` game for `KEM`

```
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: $\hat{b} \overset{\$}{\leftarrow} A^{\mathcal{O}_{\texttt{Decap}}}(1^\lambda, \texttt{pk}, c^*, K_b)$
6: return $[\![\hat{b} = b]\!]$
```

Figure 3: The `KEM-IND-CCA2` game

Algorithm 4 $\mathcal{O}_{\texttt{Decap}}(c)$

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

Figure 4: Decapsulation oracle

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

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

2.3 Message authentication code (MAC)

A message authentication code `MAC` is a collection of routines $(\texttt{Sign}, \texttt{Verify})$ defined over some key space $\mathcal{K}$, some message space $\mathcal{M}$, and some tag space $\mathcal{T}$. The signing routine $\texttt{Sign}(k, m)$ takes the secret key $k \in \mathcal{K}$ and some message, and outputs a tag t. 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:

Algorithm 5 The 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 EUF-CMA game

The advantage $\texttt{Adv}_{\texttt{EUF-CMA}}$ of the existential forgery adversary is the probability that it wins the EUF-CMA game.

We are specifically interested in one-time MAC, whose security goal is identical to EUF-CMA described above, except for the constraint that each secret key can be used to sign exactly one distinct message. This translates to an attack model in which the signing oracle will only answer one signing query. Restricting to one-time usage allows for more efficient MAC constructions. One popular way to build one-time MAC is with universal hash functions (UHF), where each instance is parameterized by a finite field $\mathbb{F}$ and a maximal message length $L \geq 0$. The secret key is a pair of field elements $(k_1, k_2) \in \mathbb{F} \times \mathbb{F}$, and each message is a tuple of up to L field elements $m = (m_1, m_2, \ldots, m_l) \in \mathbb{F}^{\leq L}$. To compute the tag:

$$\texttt{MAC}((k_1, k_2), m) = H_{\text{xpoly}}(k_1, m) + k_2$$

Where the H_{xpoly} is a universal hash function:

$$H_{\text{xpoly}}(k_1, (m_1, m_2, \ldots, m_l)) = k_1^l \cdot m_1 + k_1^{l-1} \cdot m_2 + \ldots + k_1 \cdot m_l$$

Lemma 1. *For all adversaries (including unbounded ones) against the MAC described above, the probability of winning the one-time EUF-CMA game is at most:*

$$\texttt{Adv}_{OT\text{-}EUF\text{-}CMA}(A) \leq \frac{L+1}{|\mathbb{F}|}$$

Proof. See [BS20] lemma 7.11 □

2.4 Related works

The Fujisaki-Okamoto transformation [FO99][HHK17] is a family of generic transformations that takes as input a PKE with weaker security, such as OW-CPA, and outputs a PKE or KEM with IND-CCA2 security. The key ingredient in achieving ciphertext non-malleability is with *de-randomization* and *re-encryption*, which first transform a OW-CPA PKE into a *rigid* PKE, then transform the rigid PKE into a KEM. More specifically:

1. *de-randomization* means that a randomized encryption routine $c \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m)$ is made into a deterministic encryption routine by deriving randomization coin pseudorandomly: $c \leftarrow \texttt{Enc}(\texttt{pk}, m, r = H(m))$ for some hash function H

2. *re-encryption* means that the transformed decryption routine will run the transformed encryption routine to verify the integrity of the ciphertext. Because after *de-randomization*, each plaintext strictly corresponds exacty one ciphertext, tempering with a ciphertext means that even if the ciphertext decrypts back to the same plaintext, the re-encryption will detect that the ciphertext has been tempered with.

3. *rigidity* means that the decryption routine is a perfect inverse of the encryption routine: $c = \texttt{Enc}(\texttt{pk}, m) \Leftrightarrow m = \texttt{Dec}(\texttt{sk}, c)$. Converting a one-way secure rigid PKE (which is essentially a trapdoor function) into a IND-CCA2 KEM is well solved problem. We refer readers to [BS20] for details on such constructions.

let $\texttt{PKE} = (\texttt{KeyGen}_{\texttt{PKE}}, \texttt{Enc}, \texttt{Dec})$ be defined over message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Let $G : \mathcal{M} \rightarrow \mathcal{R}$ hash plaintexts into coincs, and let $H : \{0,1\}^* \rightarrow \{0,1\}^*$ hash byte stream into session keys. Depending on whether the constructed KEM uses implicit or explicit rejection, and the security property of the $\texttt{PKE}$, [HHK17] described four variations. They are summarized in table 1 and figure 6.

Table 1: Variants of modular FO transforms

name	rejection	PKE security
$U^{\perp}$	explicit	OW-PCVA
$U^{\not\perp}$	implicit	OW-PCA
$U^{\perp}_m$	explicit	OW-VA + rigid
$U^{\not\perp}_m$	implicit	OW-CPA + rigid

Algorithm 6 $\texttt{KeyGen}()$

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

Algorithm 7 $\texttt{Encap}(\texttt{pk})$

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

Algorithm 8 $\texttt{Decap}(\texttt{sk} = (\texttt{sk}', z), c)$

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

Figure 6: Summary of the modular Fujisaki-Okamoto transformation variations

The modular FO transformations enjoy tight security bounds and proven quantum resistance. Variations have been deployed to many post-quantum KEMs submitted to NIST's post-quantum cryptography competition. Kyber, one of the round 3 finalists, uses the $U^{\not\perp}$ transformation. When it was later standardized into FIPS-203, it changed to use the $U^{\not\perp}_m$ transformation for computational efficiencies.

3 The “encrypt-then-MAC” transformation

Let $\texttt{PKE}(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ be a public-key encryption scheme. Let $\texttt{MAC}$ be a deterministic message authentication code. Let $G : \mathcal{M}_{\texttt{PKE}} \to \mathcal{K}_{\texttt{MAC}}$ and $H : \{0,1\}^* \to \mathcal{K}_{\texttt{KEM}}$ be hash functions, where $\mathcal{K}_{\texttt{KEM}}$ denote the set of all possible session keys. The $\texttt{EtM}$ transformation outputs a key encapsulation mechanism $\texttt{KEM}_{\texttt{EtM}}(\texttt{KeyGen}_{\texttt{EtM}}, \texttt{Encap}_{\texttt{EtM}}, \texttt{Decap}_{\texttt{EtM}})$. The three routines are described in figure 7.

Algorithm 9 $\texttt{KeyGen}_{\texttt{EtM}}$

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

Algorithm 10 $\texttt{Encap}_{\texttt{EtM}}(\texttt{pk})$

```
1: $m \xleftarrow{\$} \mathcal{M}_{\texttt{PKE}}$
2: $k \leftarrow G(m)$
3: $c_{\texttt{PKE}} \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m)$
4: $t \leftarrow \texttt{MAC}(k, c_{\texttt{PKE}})$
5: $K \leftarrow H(m, c_{\texttt{PKE}})$
6: $c \leftarrow (c_{\texttt{PKE}}, K)$
7: return $(c, K)$
```

Algorithm 11 $\texttt{Decap}_{\texttt{EtM}}(\texttt{sk}, c)$

```
1: $(c_{\texttt{PKE}}, t) \leftarrow c$
2: $(\texttt{sk}_{\texttt{PKE}}, z) \leftarrow \texttt{sk}$
3: $\hat{m} \leftarrow \texttt{Dec}(\texttt{sk}_{\texttt{PKE}}, c_{\texttt{PKE}})$
4: $\hat{k} \leftarrow G(\hat{m})$
5: if $\texttt{MAC}(\hat{k}, c_{\texttt{PKE}}) \neq t$ then
6:     return $H(z, c_{\texttt{PKE}})$
7: end if
8: return $H(\hat{m}, c_{\texttt{PKE}})$
```

Figure 7: $\texttt{KEM}_{\texttt{EtM}}$ routines

Theorem 1. *For every* $\texttt{IND-CCA2}$ *adversary* A *against* $\texttt{KEM}_{\texttt{EtM}}$ *that makes* q_D *decapsulation queries, there exists an* $\texttt{OW-PCA}$ *adversary* B *who makes at least* q_D *plaintext-checking queries against the underlying* $\texttt{PKE}$ *such that*

$$\texttt{Adv}_{\texttt{IND-CCA2}}(A) \leq q_D \cdot \epsilon_{\texttt{MAC}} + 2 \cdot \texttt{Adv}_{\texttt{OW-PCA}}(B)$$

Proof. We will prove using a sequence of games. The complete sequence of games is shown in figure 8

Algorithm 12 Sequence of games $G_0 - G_3$

```
1: $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{KeyGen}(1^\lambda)$
2: $(m^*, z) \xleftarrow{\$} \mathcal{M}_{\mathtt{PKE}}$
3: $k^* \leftarrow G(m^*)$                                    ▷ $G_0$-$G_1$
4: $k^* \xleftarrow{\$} \mathcal{K}_{\mathtt{MAC}}$                                    ▷ $G_2$-$G_3$
5: $c^*_{\mathtt{PKE}} \xleftarrow{\$} \mathtt{Enc}(\mathtt{pk}, m^*)$
6: $t^* \leftarrow \mathtt{MAC}(k^*, c^*_{\mathtt{PKE}})$
7: $c^* \leftarrow (c^*_{\mathtt{PKE}}, t^*)$
8: $K_0 \leftarrow H(m^*, c^*_{\mathtt{PKE}})$                                    ▷ $G_0$-$G_2$
9: $K_0 \xleftarrow{\$} \mathcal{K}_{\mathtt{KEM}}$                                    ▷ $G_3$
10: $K_1 \xleftarrow{\$} \mathcal{K}_{\mathtt{KEM}}$
11: $b \xleftarrow{\$} \{0, 1\}$
12: $\hat{b} \leftarrow A^{\mathcal{O}^{\mathsf{Decap}}}(1^\lambda, \mathtt{pk}, c^*, K_b)$                                    ▷ $G_0$
13: $\hat{b} \leftarrow A^{\mathcal{O}_1^{\mathsf{Decap}}}(1^\lambda, \mathtt{pk}, c^*, K_b)$                                    ▷ $G_1$-$G_3$
14: return $[\![\hat{b} = b]\!]$
```

Algorithm 13 $\mathcal{O}^{\mathsf{Decap}}(c)$

```
1: $(c_{\mathtt{PKE}}, t) \leftarrow c$
2: $\hat{m} \leftarrow \mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c_{\mathtt{PKE}})$
3: $\hat{k} \leftarrow G(\hat{m})$
4: if $\mathtt{MAC}(\hat{k}, c_{\mathtt{PKE}}) = t$ then
5:     return $H(\hat{m}, c_{\mathtt{PKE}})$
6: end if
7: return $H(z, c_{\mathtt{PKE}})$
```

Algorithm 14 $\mathcal{O}_1^{\mathsf{Decap}}(c)$

```
1: $(c_{\mathtt{PKE}}, t) \leftarrow c$
2: if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G : \mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c_{\mathtt{PKE}}) = \tilde{m} \land \mathtt{MAC}(\tilde{k}, c_{\mathtt{PKE}}) = t$ then
3:     return $H(\tilde{m}, c_{\mathtt{PKE}})$
4: end if
5: return $H(z, c_{\mathtt{PKE}})$
```

Figure 8: Sequence of games, true decap oracle $\mathcal{O}^{\mathsf{Decap}}$ and simulated oracle $\mathcal{O}_1^{\mathsf{Decap}}$

Game 0 is the standard `IND-CCA2` game for a key encapsulation mechanism.

Game 1 is identical to *Game 0* except for that the decapsulation oracle $\mathcal{O}^{\mathsf{Decap}}$ (algorithm 13) is replaced with a simulated decapsulation oracle $\mathcal{O}_1^{\mathsf{Decap}}$ (algorithm 14). If $\mathcal{O}_1^{\mathsf{Decap}}$ accepts the queried ciphertext $c = (c_{\mathtt{PKE}}, t)$ and outputs the true session key $K \leftarrow H(\tilde{m}, c_{\mathtt{PKE}})$, then the queried ciphertext must be honestly generated, which means that $\mathcal{O}^{\mathsf{Decap}}$ must also accept the queried ciphertext and output the true session key. If $\mathcal{O}^{\mathsf{Decap}}$ rejects the queried ciphertext $c = (c_{\mathtt{PKE}}, t)$ and outputs the implicit rejection $K \leftarrow H(z, c_{\mathtt{PKE}})$, then the tag t is invalid under the MAC key $k \leftarrow G(\mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c_{\mathtt{PKE}}))$. Since for a given ciphertext $c_{\mathtt{PKE}}$, the correct MAC key is fixed, there could not be a matching hash query (m, k) such that m is the correct decryption and k can validate the incorrect tag. Therefore, $\mathcal{O}_1^{\texttt{Decap}}$ must also reject the queried ciphertext and output the implicit rejection.

This means that game 0 and game 1 differ when $\mathcal{O}^{\texttt{Decap}}$ accepts the queried ciphertext $c = (c_{\texttt{PKE}}, t)$ but $\mathcal{O}_1^{\texttt{Decap}}$ rejects it, which means that t is a valid tag for $c_{\texttt{PKE}}$ under the correct MAC key $k \leftarrow G(\texttt{Dec}(\texttt{sk}_{\texttt{PKE}}, c_{\texttt{PKE}}))$ but such key is never queried by the adversary. Under the random oracle model, from the adversary's perspective, such k is an unknown and uniformly random key, so producing a valid tag under such key constitutes a forgery against the MAC. Denote the probability of forgery against unknown uniformly random MAC key by $\epsilon_{\texttt{MAC}}$, then the probability that the two decapsulation oracles disagree on one or more queries is at most $q_D \cdot \epsilon_{\texttt{MAC}}$. Finally, by the difference lemma,

$$\texttt{Adv}_0(A) - \texttt{Adv}_1(A) \leq q_D \cdot \epsilon_{\texttt{MAC}}$$

Note that $\epsilon_{\texttt{MAC}}$ quantifies the probability that an adversary can produce forgery for a unknown key without access to a signing oracles. While this is not a standard security definition for MAC, this probability is straightforward to estimate for some classes of MACs. As described in section 2.3, with a Carter-Wegman-like one-time MAC instantiated with message length L and a finite field with F elements, such probability is at most $\epsilon_{\texttt{MAC}} \leq \frac{L+1}{F}$

Game 2 is identical to *Game 1*, except for that when the challenger generates the challenge ciphertext $c^* = (c^*_{\texttt{PKE}}, t^*)$, the tag t^* is computed using a uniformly random key $k^* \leftarrow \mathcal{K}_{\texttt{MAC}}$ instead of a pseudorandom key derived from hashing the challenge plaintext.

Under the random oracle model, game 2 and game 1 are statistically identical to the adversary A, unless A queries G with m^*. Denote the probability that A queries G with m^* by $P[\texttt{QUERY G}^*]$, then:

$$\texttt{Adv}_1(A) - \texttt{Adv}_2(A) \leq P[\texttt{QUERY G}^*]$$

Game 3 is identical to *Game 2*, except for that K_0 is a uniformly random session key instead of a pseudorandom session key derived from the challenge plaintext-ciphertext pair. Under the random oracle model, game 3 and game 2 are statistically identical unless the adversary A queries H with $(m^*, \cdot)$. Denote the probability that A makes such H query by $P[\texttt{QUERY H}^*]$, then:

$$\texttt{Adv}_2(A) - \texttt{Adv}_3(A) \leq P[\texttt{QUERY H}^*]$$

In game 3, both K_0 and K_1 are uniformly random. There is no statistical difference between the two session keys, so no adversary can have any advantage: $\texttt{Adv}_3(A) = 0$.

Now consider an $\texttt{OW-PCA}$ adversary B simulating game 3 for A:

1. When B receives its public key $\texttt{pk}$, B passes $\texttt{pk}$ to A
2. B can sample the implicit rejection z by itself
3. B can simulate both hash oracles G and H for A
4. B can simulate $\mathcal{O}_1^{\texttt{Decap}}$ for A. Instead of checking if $\texttt{Dec}(\texttt{sk}_{\texttt{PKE}}, c) = \tilde{m}$, B can use its access to the plaintext-checking oracle and check if $\texttt{PCO}(\tilde{m}, c) = 1$. This means that for every decapsulation query B services, B needs to make at least one plaintext-checking query. Therefore, B needs to make at least q_D plaintext-checking query.
5. When B receives its challenge ciphertext $c^*_{\texttt{PKE}}$, it can sample a uniformly random key $k^* \xleftarrow{\$} \mathcal{K}_{\texttt{MAC}}$, produce the corresponding tag $t^* \leftarrow \texttt{MAC}(k^*, c^*_{\texttt{PKE}})$, and sample a uniformly random session keys $K_0, K_1 \xleftarrow{\$} \mathcal{K}_{\texttt{KEM}}$. B then passes $c^* = (c^*_{\texttt{PKE}}, t^*)$ as the challenge ciphertext and a coin-flip K_b as the challenge session key.

If A ever queries G or H with the decryption of $c^*_{\texttt{PKE}}$, B will be able to detect it using the plaintext-checking oracle. From where B is guaranteed to win the `OW-PCA` game. Therefore:

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

Combining all inequalities above, we have:

$$\texttt{Adv}_0(A) \leq q_D \cdot \epsilon_{\texttt{MAC}} + 2\texttt{Adv}_{\texttt{OW-PCA}}(B)$$

□

4 Application to Kyber

CRYSTALS-Kyber [BDK+18][ABD+19] is an IND-CCA2 secure key encapsulation mechanism whose security is based on the conjecture hardness of the decisional Module Learning with Error problem. To achieve the the IND-CCA2 security, Kyber first constructs an IND-CPA secure public key encryption scheme based on [LPR13], then apply a slightly modified variation of the Fujisaki-Okamoto transformation described in [HHK17]. The resulting decapsulation routine is especially inefficient because Kyber's IND-CPA encryption routine incurs significantly more computational cost than the decryption routine. This makes Kyber a prime target for demonstrating the performance improvements enjoyed by the "encrypt-then-MAC" KEM construction.

We took the IND-CPA PKE routines (algorithms 4, 5, 6 in [ABD+19]) and applied the "encrypt-then-MAC" transformation. The resulting KEM routines are described in algorithms 15, 16, and 17.

Algorithm 15 `Kyber.CCAKEM.KeyGen()`

```
1: $z \xleftarrow{\$} \{0,1\}^{256}$
2: $(\texttt{pk}, \texttt{sk}') \xleftarrow{\$}$ Kyber.CPAPKE.KeyGen()
3: $\texttt{sk} = (\texttt{sk}', \texttt{pk}, H(\texttt{pk}), z)$        ▷ H is instantiated with SHA3-256
4: return $(\texttt{pk}, \texttt{sk})$
```

Algorithm 16 `Kyber.CCAKEM.Encap`$^+$`(pk)`

```
1: $m \xleftarrow{\$} \{0,1\}^{256}$
2: $m' = H(m)$        ▷ Do not output system RNG directly
3: $(\bar{K}, K_{\texttt{MAC}}) = G(m' \| H(\texttt{pk}))$        ▷ G is instantiated with SHA3-512
4: $r \xleftarrow{\$} \mathcal{R}$
5: $c' \leftarrow$ Kyber.CPAPKE.Enc$(\texttt{pk}, m', r)$
6: $t = \texttt{MAC}(K_{\texttt{MAC}}, c')$
7: $K = \texttt{KDF}(\bar{K} \| t)$        ▷ KDF is instantiated with Shake256
8: $c \leftarrow (c', t)$
9: return $(c, K)$
```

Algorithm 17 $\texttt{Kyber.CCAKEM.Decap}^{+}(\texttt{sk}, c)$

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

Remark 1. We derive the MAC key from both the plaintext m and the public key $\texttt{pk}$ for the same reason [ABD+19] derives the pseudorandom coin from both m and $\texttt{pk}$. One is to allow both parties to participate in the encryption process, the other is to prevent a quantum adversary from pre-computing a large table of MAC keys that can then be brute-forced.

Remark 2. We chose to derive the shared secret K from the ciphertext digest t instead of the ciphertext itself, which saves a few Keccak permutations since t is only 128 bits while the full ciphertext could span more than a thousand bytes. This should not impact the security of the scheme since finding collision for an unknown MAC key constitutes a forgery attack on the MAC.

Remark 3. We constructed the "encrypt-then-MAC" transformation to use implicit rejection so that we can directly use existing implementation of Kyber with minimal modification. In principle, a construction with explicit rejection should also be equally secure and efficient.

Remark 4. We chose to use MAC with 256-bit key size and 128-bit tag size. When modeling the security threats, we assumed that the adversary may have access to quantum computers, making it necessary to use the maximal key size of common MACs while maintaining the minimum 128-bit security. On the other hand, the tag size can be relative small because the decapsulator (aka the decapsulation oracle) is assumed to be a classical computer, so there is no quantum speedup on brute-forcing a valid tag.

When instantiating an instance $\texttt{KEM}(\texttt{KeyGen}, \texttt{Encap}, \texttt{Decap})$, there are a variety of possible MAC's to choose from. For each of the security level and choice of MAC, we measured the number of CPU cycles needed to produce a digest of Kyber's ciphertext. The MAC implementations are taken from OpenSSL and run on an Intel MacBook Pro with i7-9700h (TODO: run it on a server). The median measurements are reported in table 2.

Table 2: Standalone MAC performances

Name	Security	768 bytes	1088 bytes	1568 bytes
CMAC	many-time	5022	5442	6090
KMAC-256	many-time	7934	9862	11742
GMAC	one-time	2778	2756	2762
Poly1305	one-time	1128	1218	1338

Based on the security reduction in section 3 we chose standalone GMAC and Poly1305 for their substantial performance advantage over other constructions. We modified the reference implementation (https://github.com/pq-crystals/kyber) to implement the proposed construction, then measured the CPU cycles needed to run each of routines. The median CPU cycles are reported in table 3.

Table 3: Performance measurements

Name	Security level	`KeyGen`	`Encap`	`Decap`
Kyber512	128 bits			
Kyber768	192 bits			
Kyber1024	256 bits			
Kyber512 + GMAC	128 bits			
Kyber768 + GMAC	192 bits			
Kyber1024 + GMAC	256 bits			
Kyber512 + Poly1305	128 bits			
Kyber768 + Poly1305	192 bits			
Kyber1024 + Poly1305	256 bits			

Compared to Kyber using the FO transform, our proposed construction adds a small amount of runtime overhead (for computing a digest) in the encapsulation routine and a small increase in ciphertext size (for the 128-bit tag). In exchange, we see significant performance runtime savings in the decapsulation routines. This trade-off is especially meaningful in a key exchange protocol where one party has substantially more computationl resource than the other. For example, in many experimental implementation of TLS with post-quantum KEMs (such as CECPQ2), the client (might be IoT devices) runs `KeyGen` and `Decap` while the server (usually data centers) runs `Encap`.

5 Conclusions and future works

Algorithm 18 $\texttt{PCO}(m, c)$

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

Figure 9: Every decapsulation oracle can be converted into a plaintext-checking oracle

We applied the "encrypt-then-MAC" transformation to Kyber, which we speculated to be safe to use in ephemeral key exchange where each decapsulation key can only be used to decapsulate one ciphertext, and saw meaningful performance improvements over using the Fujisaki-Okamoto transformation. Unfortunately, the resulting KEM does not achieve the desired full IND-CCA2 security. This is because every chosen-ciphertext attack agaisnt the "'encrypt-then-MAC"' KEM can be converted into a plaintext-checking attack against the the underlying PKE (see algorithm 18), and the IND-CPA subroutines of Kyber are known to be vulnerable to key-recovery plaintext-checking attack (KR-PCA) [RRCB19][UXT+22]. This observation, combined with theorem 1, results in the following security equivalence:

Lemma 2. *$\texttt{KEM}_{\texttt{EtM}}$ is IND-CCA2 secure if and only if the input $\texttt{PKE}$ is OW-PCA secure*

However, if the input `PKE` is OW-CPA secure and rigid (which implies OW-PCA security), then the $U_m^{\perp}, U_m^{\not\perp}$ transformations from [HHK17] can build an IND-CCA2 secure KEM with almost no overhead, as the encapsulation and decapsulation routine each only adds a hash of the plaintext on top of the input encryption and decryption routine. This leaves a gap between OW-PCA security and *OW-CPA + rigidity* in which the "encrypt-then-MAC" transformation is a superior strategy for building KEM from a PKE. It remains an open problem to find a suitable PKE that is OW-PCA but not rigid.

References

[ABD$^+$19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.

[BCD$^+$16] Joppe 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 *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1006–1018, 2016.

[BDK$^+$18] Joppe 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)*, pages 353–367. IEEE, 2018.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 531–545. Springer, 2000.

[BP18] Daniel J Bernstein and Edoardo Persichetti. Towards kem unification. *Cryptology ePrint Archive*, 2018.

[BS20] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

[DKSRV18] 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 *Progress in Cryptology–AFRICACRYPT 2018: 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, Proceedings 10*, pages 282–305. Springer, 2018.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, pages 537–554. Springer, 1999.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[RRCB19] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kem schemes. *Cryptology ePrint Archive*, 2019.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *cryptology eprint archive*, 2004.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 296–322, 2022.