

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 due to using de-randomization and re-encryption for providing ciphertext integrity. In this work, we propose an alternative modular FO transformation that replaces re-encryption with a message authentication code (MAC) and prove the security bound of our construction. We then instantiate a concrete instance with ML-KEM and show that when re-encryption incurs significant computational cost, our construction provides substantial runtime speedup and reduced memory footprint.

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 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] (standardized by NIST into ML-KEM [KE23]).

The current state of the FO transformation enjoys tight security bound and quantum resistance [HHK17], but also leaves many open questions. One such problem is the use of re-encryption for providing ciphertext integrity [BP18], which requires the decryption/decapsulation to run the encryption routine as a subroutine. In many post-quantum schemes, such as Kyber, the encryption routine is substantially computationally more expensive than the decryption routine.

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 provide ciphertext integrity for a public-key encryption scheme, which then translates to an IND-CCA secure KEM. Using a message authentication code for ciphertext integrity replaces the re-encryption step in decryption with the computation of a tag, which should offer 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 MAC key. We proposed to derive the shared MAC key by hashing the plaintext message. We will prove in section 3 that, under the random oracle model, the MAC key is securely hidden behind the hash function, and producing a valid pair of ciphertext and tag without full knowledge of the plaintext constitutes a forgery attack on the message authentication code. Thanks to the modular construction in [HHK17],

providing ciphertext integrity in the underlying encryption scheme gives us an IND-CCA secure KEM for free.

In section 4.4, we instantiate concrete instances of our proposed transformation by modifying ML-KEM. We will demonstrate that, at the cost of small increase in encryption runtime and ciphertext size, our construction reduces both the runtime and memory footprint of the decryption routine.

2 Preliminaries and previous results

2.1 Public-key encryption scheme

We define a public key encryption scheme PKE to be a collection of three routines ($\text{Gen}, \text{Enc}, \text{Dec}$) defined over a finite message space \mathcal{M} and some ciphertext space \mathcal{C} . Many encryption routines are probabilistic, and we define their source of randomness to come from some coin space \mathcal{R} .

The encryption routine $\text{Enc}(\text{pk}, m)$ 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 $\text{Dec}(\text{sk}, c)$ 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.

2.1.1 Correctness

It is common to require a PKE to be perfectly correct, meaning that for all possible keypairs (pk, sk) and plaintext messages $m \in \mathcal{M}$, $\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, m)) = m$ at all times. However, some encryption schemes, including many popular lattice-based schemes, admit a non-zero probability of decryption failure: $\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, m)) \neq m$. Furthermore, [HHK17] and [ABD⁺19] explained how decryption failure played a role in an adversary's advantage. In this paper, we inherit the definition for correctness from [HHK17]:

Definition 1 (δ -correctness). A public key encryption scheme PKE is δ -correct if

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

Where the expectation is taken over the probability distribution of keypairs $(\text{pk}, \text{sk}) \xleftarrow{\$} \text{Gen}()$

2.1.2 Security

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

In the OW-ATK game, an adversary's goal is to recover the decryption of a randomly generated ciphertext.

The adversary \mathcal{A} with access to oracle(s) \mathcal{O}_{ATK} wins the game if its guess \hat{m} is equal to the challenge plaintext m^* . The *advantage* $\epsilon_{\text{OW-ATK}}$ of an adversary in this game is the probability that it wins the game.

The choice of oracle(s) \mathcal{O}_{ATK} depends on the choice of ATK. Specifically:

Algorithm 1 The OW-ATK game

```

1:  $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \text{Gen}(1^\lambda)$ 
2:  $m^* \xleftarrow{\$} \mathcal{M}$ 
3:  $c^* \xleftarrow{\$} \text{Enc}(\mathbf{pk}, m^*)$ 
4:  $\hat{m} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\text{ATK}}}(1^\lambda, \mathbf{pk}, c^*)$ 
5: return  $\llbracket m^* = \hat{m} \rrbracket$ 

```

Figure 1: The OW-ATK game**Algorithm 2** PCO($m \in \mathcal{M}, c \in \mathcal{C}$)

```

1: return  $\llbracket \text{Dec}(\mathbf{sk}, c) = m \rrbracket$ 

```

Algorithm 3 CVO($c \in \mathcal{C}$)

```

1: return  $\llbracket \text{Dec}(\mathbf{sk}, c) \in \mathcal{M} \rrbracket$ 

```

Figure 2: The Plaintext-Checking Oracle PCO**Figure 3:** the Ciphertext-Validation Oracle CVO

$$\mathcal{O}_{\text{ATK}} = \begin{cases} - & \text{ATK} = \text{CPA} \\ \text{PCO} & \text{ATK} = \text{PCA} \\ \text{CVO} & \text{ATK} = \text{VA} \\ \text{PCO}, \text{CVO} & \text{ATK} = \text{PCVA} \end{cases}$$

83 Where the definitions of plaintext-checking oracle PCO and the ciphertext-validation
84 oracle CVO are inherited from [HHK17]

85 In the IND-CPA game (algorithm 4), an adversary's goal is to distinguish the encryption
86 of one message from the encryption of another message. Given the public key, the adversary
87 outputs two adversarially chosen messages and obtains the encryption of a random choice
88 between these two messages. The adversary wins the IND-CPA game if it correctly identifies
89 which message the encryption is obtained from.

Algorithm 4 The IND-CPA game

```

1:  $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \text{Gen}(1^\lambda)$ 
2:  $(m_0, m_1) \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathbf{pk})$ 
3:  $b \xleftarrow{\$} \{0, 1\}$ 
4:  $c^* \xleftarrow{\$} \text{Enc}(\mathbf{pk}, m_b)$ 
5:  $\hat{b} \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathbf{pk}, c^*)$ 
6: return  $\llbracket b = \hat{b} \rrbracket$ 

```

Figure 4: The IND-CPA game

90 The *advantage* $\epsilon_{\text{IND-CPA}}$ of an IND-CPA adversary A is defined by

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

2.2 Key encapsulation mechanism

A key encapsulation mechanism KEM is a collection of three routines (**Gen**, **Encap**, **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 $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \text{Gen}(1^\lambda)$. **Encap**(\mathbf{pk}) is a probabilistic routine that takes a public key \mathbf{pk} and outputs a pair of values (c, K) where $c \in \mathcal{C}$ is the encapsulation (or ciphertext) of the shared secret $k \in \mathcal{K}$. **Decap**(\mathbf{sk}, c) is a deterministic routine that takes the secret key \mathbf{sk} and the encapsulation c and returns the shared secret k if the ciphertext is valid, or the rejection symbol \perp if the ciphertext is invalid.

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 5 IND-CCA game for KEM

```

1:  $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \text{Gen}(1^\lambda)$ 
2:  $(c^*, k_0) \xleftarrow{\$} \text{Encap}(\mathbf{pk})$ 
3:  $k_1 \xleftarrow{\$} \mathcal{K}$ 
4:  $b \xleftarrow{\$} \{0, 1\}$ 
5:  $\hat{b} \xleftarrow{\$} \mathcal{A}_{\text{IND-CCA}}^{\mathcal{O}^{\text{Decap}}}(1^\lambda, \mathbf{pk}, c^*, k_b)$ 
6: return  $\llbracket \hat{b} = b \rrbracket$ 

```

Figure 5: The KEM-IND-CCA2 game

The decapsulation oracle $\mathcal{O}^{\text{Decap}}$ takes a ciphertext c and returns the output of the **Decap** routine using the secret key. The advantage $\epsilon_{\text{IND-CCA}}$ of an IND-CCA adversary $\mathcal{A}_{\text{IND-CCA}}$ is defined by

$$\epsilon_{\text{IND-CCA}} = \left| P[\hat{b} = b] - \frac{1}{2} \right|$$

2.3 Message authentication code

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

The security of a MAC is defined in an adversarial game in which an adversary, with access to some signing oracle $\mathcal{O}_{\text{sign}}(m)$, 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 6 The EUF-CMA game

```

1:  $k^* \xleftarrow{\$} \mathcal{K}$ 
2:  $(\hat{m}, \hat{t}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\text{sign}}}()$ 
3: return  $\llbracket \text{Verify}(k^*, \hat{m}, \hat{t}) \wedge (\hat{m}, \hat{t}) \notin \mathcal{O}_{\text{sign}} \rrbracket$ 

```

Figure 6: The EUF-CMA game

116 The advantage $\epsilon_{\text{EUF-CMA}}$ of the existential forgery adversary is the probability that it
 117 wins the EUF-CMA game.

2.4 Modular Fujisaki-Okamoto transformation

119 The Fujisaki-Okamoto transformation (FOT) [FO99] is a generic transformation that
 120 takes a PKE with weaker security (such as OW-CPA or IND-CPA) and outputs a PKE with
 121 stronger security. A later variation [HHK17] improved the original construction in [FO99]
 122 by accounting for decryption failures, tightening security bounds, and providing a modular
 123 construction that first transforms OW-CPA/IND-CPA PKE into OW-PCVA PKE by providing
 124 ciphertext integrity through re-encryption (the T transformation), then transforming the
 125 OW-PCVA PKE into an IND-CCA KEM (the U transformation).

126 Particularly relevant to our results are two variations of the U transformation: U^\perp
 127 (KEM with explicit rejection) and U^\perp (KEM with implicit rejection). If PKE is OW-PCVA
 128 secure, then U^\perp transforms PKE into an IND-CCA secure KEM $^\perp$:

129 **Theorem 1.** *For any IND-CCA adversary \mathcal{A}_{KEM} against KEM $^\perp$ with advantage ϵ_{KEM} issuing*
 130 *at most q_D decapsulation queries and at most q_H hash queries, there exists an OW-PCVA*
 131 *adversary \mathcal{A}_{PKE} against the underlying PKE with advantage ϵ_{PKE} that makes at most q_H*
 132 *queries to PCO and CVO such that*

$$\epsilon_{\text{KEM}} \leq \epsilon_{\text{PKE}}$$

133 Similarly, if PKE is OW-PCA secure, then U^\perp transforms PKE into an IND-CCA secure KEM $^\perp$

134 **Theorem 2.** *For any IND-CCA adversary \mathcal{A}_{KEM} against KEM $^\perp$ with advantage ϵ_{KEM} issuing*
 135 *at most q_D decapsulation queries and at most q_H hash queries, there exists an OW-CPA*
 136 *adversary \mathcal{A}_{PKE} against the underlying PKE with advantage ϵ_{PKE} issuing at most q_H queries*
 137 *to PCO such that:*

$$\epsilon_{\text{KEM}} \leq \frac{q_H}{|\mathcal{M}_{\text{PKE}}|} + \epsilon_{\text{PKE}}$$

138 The modularity of the T and U transformation allows us to tweak only the T transfor-
 139 mation (see section 3), obtain OW-PCVA security, then automatically get IND-CCA security
 140 for free. This means that we can directly apply our contribution to existing KEM's already
 141 using this modular transformation, such as ML-KEM [KE23], and obtain performance
 142 improvements while maintaining comparable levels of security (see section 4.4).

3 The “encrypt-then-MAC” transformation

143 Let PKE(Gen, Enc, Dec) be a public-key encryption scheme. Let MAC be a deterministic
 144 message authentication code. Let $G : \mathcal{M}_{\text{PKE}} \rightarrow \mathcal{K}_{\text{MAC}}$ and $H : \{0, 1\}^* \rightarrow \mathcal{K}_{\text{KEM}}$ be hash
 145 functions, where \mathcal{K}_{KEM} denote the set of all possible session keys. The EtM transformation
 146

147 outputs a key encapsulation mechanism $\text{KEM}_{\text{EtM}}(\text{Gen}_{\text{EtM}}, \text{Encap}_{\text{EtM}}, \text{Decap}_{\text{EtM}})$. The three
 148 routines are described in figure 7.

Algorithm 7 Gen_{EtM}

```

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

Algorithm 8 $\text{Encap}_{\text{EtM}}(\text{pk})$

```

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

Algorithm 9 $\text{Decap}_{\text{EtM}}(\text{sk}, c)$

```

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

Figure 7: KEM_{EtM} routines

149 **Theorem 3.** For every *IND-CCA2* adversary A against KEM_{EtM} that makes at most q_D
 150 decapsulation queries, there exists an *OW-PCA* adversary B who makes q_P plaintext-checking
 151 queries against the underlying *PKE* such that

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

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

Algorithm 10 Sequence of games $G_0 - G_3$

```

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

```

Algorithm 11 $\mathcal{O}^{\text{Decap}}(c)$

```

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

```

Algorithm 12 $\mathcal{O}_1^{\text{Decap}}(c)$

```

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

```

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

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

155 *Game 1* is identical to *Game 0* except for that the decapsulation oracle $\mathcal{O}^{\text{Decap}}$ (algorithm
156 11) is replaced with a simulated decapsulation oracle $\mathcal{O}_1^{\text{Decap}}$ (algorithm 12). If $\mathcal{O}_1^{\text{Decap}}$
157 accepts the queried ciphertext $c = (c_{\text{PKE}}, t)$ and outputs the true session key $K \leftarrow H(\tilde{m}, c_{\text{PKE}})$,
158 then the queried ciphertext must be honestly generated, which means that $\mathcal{O}^{\text{Decap}}$ must
159 also accept the queried ciphertext and output the true session key. If $\mathcal{O}^{\text{Decap}}$ rejects the
160 queried ciphertext $c = (c_{\text{PKE}}, t)$ and outputs the implicit rejection $K \leftarrow H(z, c_{\text{PKE}})$, then the
161 tag t is invalid under the MAC key $k \leftarrow G(\text{Dec}(\text{sk}_{\text{PKE}}, c_{\text{PKE}}))$. Since for a given ciphertext
162 c_{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, $\mathcal{O}_1^{\text{Decap}}$ must also reject the queried ciphertext and output the implicit rejection.

This means that game 0 and game 1 differ when $\mathcal{O}^{\text{Decap}}$ accepts the queried ciphertext $c = (c_{\text{PKE}}, t)$ but $\mathcal{O}_1^{\text{Decap}}$ rejects it, which means that t is a valid tag for c_{PKE} under the correct MAC key $k \leftarrow G(\text{Dec}(\text{sk}_{\text{PKE}}, c_{\text{PKE}}))$ 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 ϵ_{MAC} , then the probability that the two decapsulation oracles disagree on one or more queries is at most $q_D \cdot \epsilon_{\text{MAC}}$. Finally, by the difference lemma,

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

□

3.1 If PKE is not OW-PCA, then EtM is not IND-CCA2 secure

More specifically, if there exists an efficient OW-PCA adversary B against the underlying PKE, then we can build an efficient IND-CCA2 adversary A who uses B as a sub-routine and wins the IND-CCA2 game.

To run B as a sub-routine, A needs to be able to simulate a plaintext-checking oracle, which can be done using the decapsulation oracle. When presented with a plaintext-checking query (\tilde{m}, \tilde{c}) , A derives the corresponding MAC key $\tilde{k} \leftarrow G(\tilde{m})$ and signs the ciphertext $\tilde{t} \leftarrow \text{MAC}(\tilde{k}, \tilde{c})$. A then queries the decapsulation oracle on the ciphertext-tag pair $\tilde{K} \leftarrow \mathcal{O}^{\text{Decap}}(\tilde{c}, \tilde{t})$.

Because the correct session key is deterministically derived from the public key, the plaintext, the ciphertext, and some other values derived from these three values, for each set of $(\tilde{m}, \tilde{c}, \tilde{t}, \tilde{K})$, A can correctly derive the session key if \tilde{m} is indeed the decryption of \tilde{c} . On the other hand, if $\tilde{m} \neq \text{Dec}_{\text{PKE}}(\text{sk}_{\text{PKE}}, \tilde{c})$, then the decapsulation oracle will return the implicit rejection value, which will not match the expected session key value.

By comparing the decapsulation oracle's output and the expected session key, A can correctly distinguish whether \tilde{m} is the decryption of \tilde{c} or not. Thus A can correctly simulate the plaintext-checking oracle for B .

When A receives the challenge ciphertext (c^*, t^*) and unknown session key $K_b \xleftarrow{\$} \{K_0, K_1\}$, A passes c^* as the challenge ciphertext to B as B 's challenge ciphertext. After B returns the guess \hat{m} , A computes the expected session key using \hat{m} and c^* . If \hat{m} is the correct decryption of c^* , then the expected session key should match the correct session key, in which case A will correctly distinguish the true session key from the random session key. If \hat{m} is not the correct decryption of c^* , then the expected session key will probably not match anything, so A will always claim K_b to be a random key. In other words, *if B wins the OW-PCA game, then A wins the IND-CCA2 game; if B does not win, then A 's chance of winning is exactly $\frac{1}{2}$* . Therefore, the advantage of A is at least that of B .

4 Application to Kyber

4.1 Kyber with AE

CRYSTALS-Kyber [BDK⁺18][ABD⁺19] and ML-KEM [KE23] are IND-CCA2 secure key encapsulation mechanisms whose security depends on the hardness of the Module Learning with Error (MLWE) problem. For the construction of the IND-CCA2 secure KEM, Kyber first constructs an IND-CPA public-key encryption scheme (which we will call CPAPKE), then applies the modular Fujisaki-Okamoto transformation [HHK17] to construct the key encapsulation mechanism (which we will call CCAKEM). Specifically, Kyber's round-3

208 submission uses the U^\perp transformation, while ML-KEM uses the U_m^\perp transformation. The
 209 routines of CPAPKE can be found in Algorithm 4, 5, 6 in [ABD⁺19] and are largely identical
 210 between Kyber and ML-KEM.

211 The routines of CCAKEM can be found in Algorithm 7, 8, 9 in [ABD⁺19]. We modify
 212 Algorithms 8 and 9 using authenticated encryption (AE) mode as follows where Algorithm
 213 7 is unchanged.

Algorithm 13 Kyber.CCAKEM.KeyGen()

```

1:  $z \xleftarrow{\$} \mathcal{B}^{32}$  ▷ Randomly sample 32 bytes (256 bits)
2:  $(pk, sk') \xleftarrow{\$} \text{Kyber.CPAPKE.KeyGen}()$ 
3:  $sk = (sk', pk, H(pk), z)$  ▷ H is instantiated with SHA3-256
4: return  $(pk, sk)$ 

```

Algorithm 14 Kyber.CCAKEM.Encap⁺(pk)

```

1:  $m \xleftarrow{\$} \mathcal{B}^{32}$  ▷ Do not output system RNG directly
2:  $(\bar{K}, r, K_{mac}) = G(m' || H(pk))$  ▷ G is instantiated with SHA3-512
3:  $c \leftarrow \text{Kyber.CPAPKE.Enc}(pk, m', r)$  ▷ Because r is set, CPAPKE is deterministic
4:  $t_1 = MAC(K_{mac}, c)$ 
5:  $K = KDF(\bar{K} || t_1)$  ▷ KDF is instantiated with Shake256
6:  $c \leftarrow (c, t_1)$ 
7: return  $(c, K)$ 

```

Algorithm 15 Kyber.CCAKEM.Decap⁺(sk, c, t₁, t₂)

Require: Secret key $sk = (sk', pk, H(pk), z)$

Require: Kyber.CPAPKE Ciphertext c and Tags t_1, t_2

```

1:  $(sk', pk, h, z) \leftarrow sk$  ▷ Unpack the secret key; h is the hash of pk
2:  $(c, t_1) \leftarrow c$ 
3:  $m = \text{Kyber.CPAPKE.Dec}(sk', c)$ 
4:  $(\bar{K}, r, K_{mac}) = G(m' || h)$ 
5:  $t'_1 = MAC(K_{mac}, c)$ 
6: if  $t'_1 = t_1$  then
7:    $K = KDF(\bar{K} || t_1)$ 
8:   return  $K$ 
9: else
10:   Abort
11: end if

```

214 *Remark 1.* If c is manipulated, then the verification of t_1 will be failed. In this case, there
 215 is no K outputted from the decap⁺. So the attacks described in the following subsections
 216 won't work. We also added the key confirmation, which is tag t_2 .

217 *Note for authenticated encryption, tags t_i 's are necessary for the inputs.*

218 *In fact, this is authenticated encryption instead of EtM. So we should change that,*
 219 *called authenticated encryption. Please do not change my notation. They have their*
 220 *meanings in AE mode.*

221 4.2 “encrypt-then-MAC” is not secure with Kyber

222 **Lemma 1.** *There exists efficient OW-PCA adversary against CPAPKE*

The main reason why a plaintext-checking attack works against CPAPKE is that for each known honest plaintext-ciphertext pair (m, c) , one can perturb the ciphertext and obtain a new ciphertext $c' \leftarrow \text{addNoise}(c)$. Where the perturbation is small, the perturbed ciphertext can still correctly decrypt to the original plaintext $\text{Dec}_{\text{CPAPKE}}(\text{sk}, c') = \text{Dec}_{\text{CPAPKE}}(\text{sk}, c) = m$. Where the perturbation is large, the perturbed ciphertext will not decrypt to the original plaintext. The boundary between "small perturbation" and "large perturbation" depends on the values of the secret \mathbf{s} , so an adversary can probe the boundary in each of the $k \times n$ dimensions and learn the value of \mathbf{s} coefficient-by-coefficient.

As described in section 3.1, when the integrity of the unauthenticated ciphertext is protected under a MAC key derived from the corresponding plaintext, **the MAC will not prevent an adversary from perturbing an honest ciphertext** since the adversary can use the same MAC key to compute tag t' on the perturbed unauthenticated ciphertext c' .

On the other hand, [HHK17] prevents tempering by the combination of *de-randomization* and *re-encryption*. Even if $c' \leftarrow \text{addNoise}(c)$ still decrypts back to the same plaintext, *de-randomization* and *re-encryption* will reveal that c' has been tempered with because $c' \neq \text{Enc}(\text{pk}, m; r \leftarrow G(m))$. Unlike KEM_{EtM} where an adversary can temper with the unauthenticated ciphertext and still produce some valid authenticated ciphertexts, with *re-encryption* there is no "other valid ciphertext" that correspond to the same plaintext, thus preventing all tempering.

4.3 Is "encrypt-then-MAC" useful?

From [HHK17] we know that if a PKE is rigid and OW-CPA secure, then the U_m^χ transformation is sufficient for constructing an IND-CCA2 secure KEM with tight security reduction. Furthermore, the performance overhead of U_m^χ over the rigid PKE is minimal. Note that rigidity and OW-CPA automatically implies OW-PCA security, so KEM_{EtM} will also construct an IND-CCA2 secure KEM, but with extra runtime and communication overhead, so there is practical point of using EtM over the U_m^χ transformation. On the other hand, if the PKE is not OW-PCA, then KEM_{EtM} is not IND-CCA2 secure.

If the PKE is not rigid but still OW-PCA secure, *de-randomization + re-encryption* is still an option, so we need to consider the performance trade-offs. *de-randomization + re-encryption* adds one addition hash (for deriving coin) into the encryption routine and add one hash call and one call to the input encryption routine in the decryption routine. "encrypt-then-MAC" introduces one hash call and one MAC call to the encryption routine, adds a tag to the ciphertext size, and adds one hash call and one MAC call to the decryption routine. Using a one-time MAC like GMAC or Poly1305, the computational cost of producing a tag and the communication cost of the tag should be minimal (1000-2000 CPU cycles + 128 bits of tag). Where appropriate, replacing *re-encryption* in the decryption routine with a MAC computation can lead to substantial saving in computational cost. This is especially true in protocols where the client (often constrained environments) is responsible for running decryption routine, such as hybrid key exchanges used in CECPQ2.

Table 1: Use cases

Input PKE	what should I use to build KEM
rigid	U_m^χ
OW-PCA but not rigid	EtM
OW-CPA but not OW-PCA	<i>de-randomization + re-encryption</i>

4.4 MAC Performance

We claim that the input MAC only needs to be one-time existentially unforgeable. This is because besides the challenge ciphertext (c^*, t^*) , the adversary has no external resources from which it can obtain authenticated ciphertexts for which it does not know the decryption. Here we compare the performance of a variety of MAC instantiations. Some of them are many-time secure while others are one-time secure. The standalone performance results are listed in table 2. For each choice of MAC, we checked the median (top) and average (bottom) CPU cycles (run on a 2019 MacBook Pro 16-inch) needed to sign 768, 1088, and 1568 bytes of data (respectively the ciphertext size for Kyber512, Kyber768, and Kyber1024).

Table 2: Standalone MAC performances

Name	Security	768 bytes	1088 bytes	1568 bytes
CMAC	many-time	5022	5442	6090
		5131	5578	6154
GMAC	one-time	2778	2756	2762
		2843	2780	2919
KMAC-256	many-time	7934	9862	11742
		8594	10693	12319
Poly1305	one-time	1128	1218	1338
		1435	1504	1625

5 Conclusions and future works

References

- [ABD⁺19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede 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.
- [BDK⁺18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede 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. *Cryptography ePrint Archive*, 2018.
- [FO99] Eiichi 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.

- 295 [KE23] NIST Module-Lattice-Based Key-Encapsulation. Mechanism standard. *NIST*
296 *Post-Quantum Cryptography Standardization Process; NIST: Gaithersburg, MD,*
297 *USA*, 2023.
- 298 [Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security
299 proofs. *cryptology eprint archive*, 2004.