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 4.3, 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 $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ 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 $\texttt{Enc}(\texttt{pk}, m)$ 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.

2.1.1 Correctness

It is common to require a PKE to be perfectly correct, meaning that for all possible keypairs $(\texttt{pk}, \texttt{sk})$ and plaintext messages $m \in \mathcal{M}$, $\texttt{Dec}(\texttt{sk}, \texttt{Enc}(\texttt{pk}, m)) = m$ at all times. However, some encryption schemes, including many popular lattice-based schemes, admit a non-zero probability of decryption failure: $\texttt{Dec}(\texttt{sk}, \texttt{Enc}(\texttt{pk}, m)) \neq m$. 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

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

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

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

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

Algorithm 1 The `OW-ATK` game

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

Figure 1: The `OW-ATK` game

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

return $[\![\mathtt{Dec}(\mathtt{sk}, c) = m]\!]$

Figure 2: The Plaintext-Checking Oracle `PCO`

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

return $[\![\mathtt{Dec}(\mathtt{sk}, c) \in \mathcal{M}]\!]$

Figure 3: the Ciphertext-Validation Oracle `CVO`

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

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

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

Algorithm 4 The `IND-CPA` game

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

Figure 4: The `IND-CPA` game

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

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

2.2 Key encapsulation mechanism

A key encapsulation mechanism KEM is a collection of three routines $(\mathtt{Gen}, \mathtt{Encap}, \mathtt{Decap})$ 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 $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{Gen}(1^\lambda)$. $\mathtt{Encap}(\mathtt{pk})$ 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 encapsulation (or ciphertext) of the shared secret $k \in \mathcal{K}$. $\mathtt{Decap}(\mathtt{sk}, c)$ 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, 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: $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{Gen}(1^\lambda)$
2: $(c^*, k_0) \xleftarrow{\$} \mathtt{Encap}(\mathtt{pk})$
3: $k_1 \xleftarrow{\$} \mathcal{K}$
4: $b \xleftarrow{\$} \{0, 1\}$
5: $\hat{b} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathtt{Decap}}}_{\mathtt{IND\text{-}CCA}}(1^\lambda, \mathtt{pk}, c^*, k_b)$
6: **return** $[\![\hat{b} = b]\!]$

Figure 5: The KEM-IND-CCA2 game

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

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

2.3 Message authentication code

A message authentication code MAC is a collection of routines $(\mathtt{Sign}, \mathtt{Verify})$ defined over some key space $\mathcal{K}$, some message space $\mathcal{M}$, and some tag space $\mathcal{T}$. The signing routine $\mathtt{Sign}(k, m)$ takes the secret key $k \in \mathcal{K}$ and some message, and outputs a tag t. The verification routine $\mathtt{Verify}(k, m, t)$ 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 $\mathcal{O}_{\mathtt{Sign}}(m)$, 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: $(\hat{m}, \hat{t}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\texttt{Sign}}}()$
3: **return** $[\![\texttt{Verify}(k^*, \hat{m}, \hat{t}) \wedge (\hat{m}, \hat{t}) \notin \mathcal{O}_{\texttt{Sign}}]\!]$

Figure 6: The `EUF-CMA` game

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

2.4 Modular Fujisaki-Okamoto transformation

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

Particularly relevant to our results are two variations of the `U` transformation: $\texttt{U}^{\perp}$ (KEM with explicit rejection) and $\texttt{U}^{\not\perp}$ (KEM with implicit rejection). If `PKE` is `OW-PCVA` secure, then $\texttt{U}^{\perp}$ transforms `PKE` into an `IND-CCA` secure $\texttt{KEM}^{\perp}$:

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

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

Similarly, if `PKE` is `OW-PCA` secure, then $\texttt{U}^{\not\perp}$ transforms `PKE` into an `IND-CCA` secure $\texttt{KEM}^{\not\perp}$

Theorem 2. *For any `IND-CCA` adversary $\mathcal{A}_{KEM}$ against $KEM^{\not\perp}$ with advantage ϵ_{KEM} issuing at most q_D decapsulation queries and at most q_H hash queries, there exists an `OW-CPA` adversary $\mathcal{A}_{PKE}$ against the underlying `PKE` with advantage ϵ_{PKE} issuing at most q_H queries to `PCO` such that:*

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

The modularity of the `T` and `U` transformation allows us to tweak only the `T` transformation (see section 3), obtain `OW-PCVA` security, then automatically get `IND-CCA` security for free. This means that we can directly apply our contribution to existing KEM's already using this modular transformation, such as ML-KEM [KE23], and obtain performance improvements while maintaining comparable levels of security (see section 4.3).

3 The "encrypt-then-MAC" transformation

Let $\texttt{PKE}(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ be a public-key encryption scheme. Let `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 `EtM` transformation

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

Algorithm 7 $\texttt{Gen}_{\texttt{EtM}}$

$(\texttt{pk}, \texttt{sk}_{\texttt{PKE}}) \xleftarrow{\$} \texttt{Gen}(1^\lambda)$
$z \xleftarrow{\$} \mathcal{M}_{\texttt{PKE}}$
$\texttt{sk} \leftarrow (\texttt{sk}_{\texttt{PKE}}, z)$
return $(\texttt{pk}, \texttt{sk})$

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

$m \xleftarrow{\$} \mathcal{M}_{\texttt{PKE}}$
$k_{\texttt{MAC}} \leftarrow G(\texttt{pk}, m)$
$c_{\texttt{PKE}} \xleftarrow{\$} \texttt{Enc}(\texttt{pk}, m)$
$t \leftarrow \texttt{MAC}(k, c_{\texttt{PKE}})$
$K \leftarrow H(\texttt{pk}, m, c_{\texttt{PKE}})$
$c \leftarrow (c_{\texttt{PKE}}, K)$
return (c, K)

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

$(c_{\texttt{PKE}}, t) \leftarrow c$
$(\texttt{sk}_{\texttt{PKE}}, z) \leftarrow \texttt{sk}$
$\hat{m} \leftarrow \texttt{Dec}(\texttt{sk}_{\texttt{PKE}}, c_{\texttt{PKE}})$
$\hat{k}_{\texttt{pk}} \leftarrow G(\texttt{pk}, \hat{m})$
if $\texttt{MAC}(\hat{k}, c_{\texttt{PKE}}) \neq t$ **then**
    return $H(\texttt{pk}, z, c_{\texttt{PKE}})$
end if
return $H(\texttt{pk}, \hat{m}, c_{\texttt{PKE}})$

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

Here are a few design rationale:

1. *Deriving MAC key and session key.* We choose to include the public key $\texttt{pk}$ when deriving the MAC key $k_{\texttt{MAC}}$ and the session key K for similar reason stated in [BDK+18]. If the MAC key is derived solely from the message, then an adversary can pre-compute a large lookup table mapping MAC key to the source plaintext that can applied to *all sessions*. When the adversary intercepts a ciphertext $c = (c_{\texttt{PKE}}, t)$, it can brute-force all possible MAC keys to recover the plaintext. Since brute-forcing MAC key on a known ciphertext-tag pair is an offline search, given a sufficiently large lookup table and a sufficiently large quantum computer, this search might be feasible. On the other hand, if the MAC key is derived from both the public key and the plaintext, then the adversary will need to compute the large lookup table and run the key search *per session*, which greatly increases the cost of the attack.

2. *Not hashing unauthenticated ciphertext c_{PKE}*: [ABD+19][BDK+18] derives the session key from a hash of the ciphertext $K \leftarrow \texttt{KDF}(\overline{K} \| H(c_{\texttt{PKE}}))$ to *"simplify implementation*

with non-incremental hash APIs". Since "hashing the ciphertext separately" vs "deriving session key directly from ciphertext" does not affect the security nor the performance (since the ciphertext will be hashed somewhere) of the scheme, we opt to not complicate the design with unnecessary hashes and leave the implementation considerations to concrete instantiations.

3. *Not adding key confirmation* $\sigma \leftarrow \mathit{MAC}(\cdot, K)$: within the security model of a KEM, there is no interaction between the KEM adversary and any other party (be it the challenger or any oracle) that involves the adversary sending a session key. In other words, there is no scenario in which tempering with any session key is meaningful, which means that the session key does not need protection.

3.1 If PKE is OW-PCA secure, then EtM is IND-CCA2 secure

Theorem 3. *For every* IND-CCA2 *adversary* A *against* $\mathit{KEM}_{\mathit{EtM}}$ *that makes at most* q_D *decapsulation queries, there exists an* OW-PCA *adversary* B *against the underlying* PKE *such that*

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

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

Algorithm 10 Sequence of games $G_0 - G_3$

$(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{Gen}(1^\lambda)$
$(m^*, z) \xleftarrow{\$} \mathcal{M}_{\mathtt{PKE}}$
$k^* \leftarrow G(m^*)$ ▷ Game 0-1
$k^* \xleftarrow{\$} \mathcal{K}_{\mathtt{MAC}}$ ▷ Game 2-3
$c^* \xleftarrow{\$} \mathtt{Enc}(\mathtt{pk}, m^*)$
$t^* \leftarrow \mathtt{MAC}(k^*, c^*)$
$K_0 \leftarrow H(m^*, c^*)$ ▷ Game 0-2
$K_0 \xleftarrow{\$} \mathcal{K}_{\mathtt{KEM}}$ ▷ Game 3
$K_1 \xleftarrow{\$} \mathcal{K}_{\mathtt{KEM}}$
$b \xleftarrow{\$} \{0, 1\}$
$\hat{b} \leftarrow A^{\mathcal{O}^{\mathtt{Decap}}}(1^\lambda, \mathtt{pk}, (c^*, t^*), K_b)$ ▷ Game 0
$\hat{b} \leftarrow A^{\mathcal{O}_1^{\mathtt{Decap}}}(1^\lambda, \mathtt{pk}, (c^*, t^*), K_b)$ ▷ Game 1-3
return $[\![\hat{b} = b]\!]$

Algorithm 11 $\mathcal{O}^{\mathtt{Decap}}(c, t)$

$\hat{m} \leftarrow \mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c)$
$\hat{k} \leftarrow G(\hat{m})$
if $\mathtt{MAC}(\hat{k}, c) = t$ **then**
  return $H(\hat{m}, c)$
end if
return $H(z, c)$

Algorithm 12 $\mathcal{O}_1^{\mathtt{Decap}}(c, t)$

if $\exists(\tilde{m}, \tilde{k}) \in \mathcal{L}^G$:
  $\mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c) = \tilde{m}$
  $\wedge\ \mathtt{MAC}(\tilde{k}, c) = t$ **then**
  return $H(\tilde{m}, c)$
end if
return $H(z, c)$

Figure 8: Sequence of games

Need to finish the rest of the proof □

3.2 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-chekcing query $(\tilde{m}, \tilde{c})$, A derives the corresponding $\texttt{MAC}$ key $\tilde{k} \leftarrow G(\tilde{m})$ and signs the ciphertext $\tilde{t} \leftarrow \texttt{MAC}(\tilde{k}, \tilde{c})$. A then queries the decapsulation oracle on the ciphertext-tag pair $\tilde{K} \leftarrow \mathcal{O}^{\texttt{Decap}}(\tilde{c}, \tilde{t})$.

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 $\tilde{m} \neq \texttt{Dec}_{\texttt{PKE}}(\texttt{sk}_{\texttt{PKE}}, \tilde{c})$, then the decapsulation oracle will return the implicit rejection value, which will not match the expected session key value.

By comparing the decapsulation orcale'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

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 submission uses the $U^{\not\perp}$ transformation, while ML-KEM uses the $U^{\not\perp}_m$ transformation. The routines of CPAPKE can be found in Algorithm 4, 5, 6 in [ABD+19] and are largely identical between Kyber and ML-KEM.

The routines of CCAKEM can be found in Algorithm 7, 8, 9 in [ABD+19]. We recap them here:

Algorithm 13 `Kyber.CCAKEM.KeyGen()`

1: $z \xleftarrow{\$} \mathcal{B}^{32}$ ▷ Randomly sample 32 bytes (256 bits)
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 14 `Kyber.CCAKEM.Encap(pk)`

$m \xleftarrow{\$} \mathcal{B}^{32}$
$m \leftarrow H(m)$ ▷ Do not output system RNG directly
$(\bar{K}, r) \leftarrow G(m \| H(\mathtt{pk}))$ ▷ G is instantiated with SHA3-512
$c \leftarrow \mathtt{Kyber.CPAPKE.Enc(pk,\ m,\ r)}$ ▷ Because r is set, `CPAPKE` is deterministic
$K \leftarrow \mathtt{KDF}(\bar{K} \| H(c))$ ▷ `KDF` is instantiated with Shake256
return (c, K)

Algorithm 15 `Kyber.CCAKEM.Decap(sk, c)`

Require: Secret key $\mathtt{sk} = (\mathtt{sk}', \mathtt{pk}, H(\mathtt{pk}), z)$
Require: `Kyber.CPAPKE` Ciphertext c
$(\mathtt{sk}', \mathtt{pk}, h, z) \leftarrow \mathtt{sk}$ ▷ Unpack the secret key; h is the hash of `pk`
$m' \leftarrow \mathtt{Kyber.CPAPKE.Dec}(\mathtt{sk}', c)$
$(\bar{K}', r') \leftarrow G(m' \| h)$
$c' = \mathtt{Kyber.CPAPKE.Enc}(\mathtt{pk}, m', r')$
if $c = c'$ **then**
    $K \leftarrow \mathtt{KDF}(\bar{K}' \| H(c))$
else
    $K \leftarrow \mathtt{KDF}(z \| H(c))$
end if
return K

4.1 "encrypt-then-MAC" is not secure with Kyber

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 $c' \leftarrow \mathtt{addNoise}(c)$. Where the perturbation is small, the perturbed ciphertext can still correctly decrypt to the original plaintext $\mathtt{Dec}_{\mathtt{CPAPKE}}(\mathtt{sk}, c') = \mathtt{Dec}_{\mathtt{CPAPKE}}(\mathtt{sk}, c) = m$. 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 3.2, 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 $c' \leftarrow \mathtt{addNoise}(c)$ still decrypts back to the same plaintext, *de-randomization* and *re-encryption* will reveal that c' has been tempered with because $c' \neq \mathtt{Enc}(\mathtt{pk}, m; r \leftarrow G(m))$. Unlike $\mathtt{KEM}_{\mathtt{EtM}}$ 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.

4.2 Is "encrypt-then-MAC" useful?

From [HHK17] we know that if a `PKE` is rigid and `OW-CPA` secure, then the $U^{\not\perp}_m$ transformation is sufficient for constructing an `IND-CCA2` secure `KEM` with tight security reduction.

Furtherfore, the performance overhead of $U_m^{\not\perp}$ over the rigid PKE is minimal. Note that rigidity and OW-CPA automatically implies OW-PCA security, so $\texttt{KEM}_{\texttt{EtM}}$ 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 $U_m^{\not\perp}$ transformation. On the other hand, if the PKE is not OW-PCA, then $\texttt{KEM}_{\texttt{EtM}}$ 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	$U_m^{\not\perp}$
OW-PCA but not rigid	EtM
OW-CPA but not OW-PCA	*de-randomization + re-encryption*

4.3 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 5131	5442 5578	6090 6154
GMAC	one-time	2778 2843	2756 2780	2762 2919
KMAC-256	many-time	7934 8594	9862 10693	11742 12319
Poly1305	one-time	1128 1435	1218 1504	1338 1625

5 Conclusions and future works

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.

[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.

[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.

[KE23] NIST Module-Lattice-Based Key-Encapsulation. Mechanism standard. *NIST Post-Quantum Cryptography Standardization Process; NIST: Gaithersburg, MD, USA*, 2023.

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