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 4.2, 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 $\mathtt{PCO}(m \in \mathcal{M}, c \in \mathcal{C})$

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

Figure 2: The Plaintext-Checking Oracle `PCO`

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

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

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

2.2 Key encapsulation mechanism

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

Figure 5: The KEM-IND-CCA2 game

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

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

2.3 Message authentication code

A message authentication code MAC is a collection of routines $(\texttt{Sign}, \texttt{Verify})$ 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.

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 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 4.2).

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 $\mathtt{KEM}_{\mathtt{EtM}}(\mathtt{Gen}_{\mathtt{EtM}}, \mathtt{Encap}_{\mathtt{EtM}}, \mathtt{Decap}_{\mathtt{EtM}})$. The three routines are described in figure 7.

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

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

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

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

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

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

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

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

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

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

Algorithm 10 Sequence of games $G_0 - G_3$

$(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathtt{Gen}(1^\lambda)$
$(m^*, z) \xleftarrow{\$} \mathcal{M}_{\mathtt{PKE}}$
$k^* \leftarrow G(m^*)$ $\triangleright$ G_0-G_1
$k^* \xleftarrow{\$} \mathcal{K}_{\mathtt{MAC}}$ $\triangleright$ G_2-G_3
$c^*_{\mathtt{PKE}} \xleftarrow{\$} \mathtt{Enc}(\mathtt{pk}, m^*)$
$t^* \leftarrow \mathtt{MAC}(k^*, c^*_{\mathtt{PKE}})$
$c^* \leftarrow (c^*_{\mathtt{PKE}}, t^*)$
$K_0 \leftarrow H(m^*, c^*_{\mathtt{PKE}})$ $\triangleright$ G_0-G_2
$K_0 \xleftarrow{\$} \mathcal{K}_{\mathtt{KEM}}$ $\triangleright$ G_3
$K_1 \xleftarrow{\$} \mathcal{K}_{\mathtt{KEM}}$
$b \xleftarrow{\$} \{0, 1\}$
$\hat{b} \leftarrow A^{\mathcal{O}^{\mathtt{Decap}}}(1^\lambda, \mathtt{pk}, c^*, K_b)$ $\triangleright$ G_0
$\hat{b} \leftarrow A^{\mathcal{O}_1^{\mathtt{Decap}}}(1^\lambda, \mathtt{pk}, c^*, K_b)$ $\triangleright$ G_1-G_3
return $[\![\hat{b} = b]\!]$

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

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

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

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

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

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 $\mathcal{O}^{\mathtt{Decap}}$ (algorithm 11) is replaced with a simulated decapsulation oracle $\mathcal{O}_1^{\mathtt{Decap}}$ (algorithm 12). If $\mathcal{O}_1^{\mathtt{Decap}}$ 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 $\mathcal{O}^{\mathtt{Decap}}$ must also accept the queried ciphertext and output the true session key. If $\mathcal{O}^{\mathtt{Decap}}$ 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,

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

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 $\texttt{OW-PCA}$ 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:

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

$\square$

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 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 modify Algorithms 8 and 9 using authenticated encryption (AE) mode as follows where Algorithm 7 is unchanged.

Algorithm 13 `Kyber.CCAKEM.KeyGen()`

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

Algorithm 14 `Kyber.CCAKEM.Encap`$^+$`(pk)`

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

Algorithm 15 $\texttt{Kyber.CCAKEM.Decap}^{+}(\texttt{sk}, c, t_1, t_2)$

Require: Secret key $\texttt{sk} = (\texttt{sk}', \texttt{pk}, H(\texttt{pk}), z)$
Require: $\texttt{Kyber.CPAPKE}$ Ciphertext c and Tags t_1, t_2

$(\texttt{sk}', \texttt{pk}, h, z) \leftarrow \texttt{sk}$ ▷ Unpack the secret key; h is the hash of $\texttt{pk}$
$(c, t_1) \leftarrow c$
$m = \texttt{Kyber.CPAPKE.Dec}(\texttt{sk}', c)$
$(\overline{K}, r, K_{mac}) = G(m' \| h)$
$t_1' = \texttt{MAC}(K_{\texttt{MAC}}, c)$
if $t_1' = t_1$ **then**
    $K = \texttt{KDF}(\bar{K} \| t_1)$
    return K
else
    Abort
end if

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

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

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

4.2 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 1. 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 1: 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, 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.