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

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

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

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

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 $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda)$. $\texttt{Encap}(\texttt{pk})$ 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: $(\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda)$
2: $(c^*, K_0) \xleftarrow{\$} \texttt{Encap}(\texttt{pk})$
3: $K_1 \xleftarrow{\$} \mathcal{K}$
4: $b \xleftarrow{\$} \{0, 1\}$
5: $\hat{b} \xleftarrow{\$} A^{\mathcal{O}_{\texttt{Decap}}}(1^\lambda, \texttt{pk}, c^*, K_b)$
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 also interested in one-time MAC, whose security goals are identical to normal MAC, except the signing oracle will only answer one signing query. In practice this means that the adversary only gets to see one message-tag pair, and the owner of the secret key only signs one message. One way to construct an unforgeable one-time MAC is to use a universal hash function (UHF), which computes the digest using finite field arithmetic. Each instance of the one-time MAC is parameterized by some maximal message length L and the finite field $\mathbb{F}$ in which arithmetics are performed. The tag is computed by evaluating a polynomial whose coefficients are derived from the message:

$$H(k_1, (m_1, m_2, \ldots, m_l)) = k_1^{l+1} + k_1^l m_1 + k_1^{l-1} m_2 + \ldots + k_1 m_l$$
$$\texttt{MAC}((k_1, k_2), m) = H(k_1, m) + k_2$$

We claim without proof (though proof can be found in [BS20]) that the MAC construction above is one-time existentially unforgeable.

2.4 Related works

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

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

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

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

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 4.1).

3 The "encrypt-then-MAC" transformation

Let $\mathtt{PKE}(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ be a public-key encryption scheme. Let MAC be a deterministic message authentication code. Let $G : \mathcal{M}_{\mathtt{PKE}} \to \mathcal{K}_{\mathtt{MAC}}$ and $H : \{0,1\}^* \to \mathcal{K}_{\mathtt{KEM}}$ be hash functions, where $\mathcal{K}_{\mathtt{KEM}}$ denote the set of all possible session keys. The EtM transformation outputs a key encapsulation mechanism $\mathtt{KEM}_{\mathtt{EtM}}(\mathtt{KeyGen}_{\mathtt{EtM}}, \mathtt{Encap}_{\mathtt{EtM}}, \mathtt{Decap}_{\mathtt{EtM}})$. The three routines are described in figure 6.

Algorithm 6 $\mathtt{KeyGen}_{\mathtt{EtM}}$

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

Algorithm 7 $\mathtt{Encap}_{\mathtt{EtM}}(\mathtt{pk})$

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

Algorithm 8 $\mathtt{Decap}_{\mathtt{EtM}}(\mathtt{sk}, c)$

$(c_{\mathtt{PKE}}, t) \leftarrow c$
$(\mathtt{sk}_{\mathtt{PKE}}, z) \leftarrow \mathtt{sk}$
$\hat{m} \leftarrow \mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c_{\mathtt{PKE}})$
$\hat{k} \leftarrow G(\hat{m})$
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 6: $\mathtt{KEM}_{\mathtt{EtM}}$ 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 7

Algorithm 9 Sequence of games $G_0 - G_3$

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

Algorithm 10 $\mathcal{O}^{\mathsf{Decap}}(c)$

$(c_{\mathtt{PKE}}, t) \leftarrow c$
$\hat{m} \leftarrow \mathtt{Dec}(\mathtt{sk}_{\mathtt{PKE}}, c_{\mathtt{PKE}})$
$\hat{k} \leftarrow G(\hat{m})$
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 11 $\mathcal{O}_1^{\mathsf{Decap}}(c)$

$(c_{\mathtt{PKE}}, t) \leftarrow c$
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 7: 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 10) is replaced with a simulated decapsulation oracle $\mathcal{O}_1^{\mathsf{Decap}}$ (algorithm 11). 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 will be discussed in section 4.1, 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 that first constructs an IND-CPA secure public key encryption scheme whose security is based on the Module Learning with Error Problem (MLWE), then applies a generic transformation using *de-randomization* and *re-encryption* [HHK17]. The use of *re-encryption* for providing rigidity means that the decapsulation routine needs to run the encryption routine for verifying ciphertext integrity. Unfortunately for Kyber, the encryption routine consumes both more CPU cycles and carries a larger memory footprint. Therefore, applying an alternate transformation that replaces *re-encryption* with a MAC verification will bring substantial performance enhancement. This is particularly applicable to communication protocols such as TLS 1.3, where clients, often constrained environments, need to run the decapsulation routine.

The IND-CPA PKE of Kyber (algorithms 4, 5, 6 in [ABD+19]) is not OW-PCA secure. A plaintext-checking attack [RRCB19] can recover the secret key using a few maliciously constructed plaintext-checking queries for each coefficient of a Kyber secret key (there are 512, 768, and 1024 coefficients in the secret key depending on the desired security level). However, we propose mitigation for plaintext-checking attack at protocol level by requiring each keypair to be used for decryption only once. If a decryption fails, then the secret key should be discarded and the key exchange terminated, and a new keypair should be generated to restart the key exchange. In fact, such ephemeral key exchange is already required in TLS 1.3 for forward secrecy, so requiring the one-time use of keypair does not introduce additional operational cost for protocols such as TLS 1.3.

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 12 `Kyber.CCAKEM.KeyGen()`

1: $z \xleftarrow{\$} \mathcal{B}^{32}$
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 13 $\texttt{Kyber.CCAKEM.Encap}^{+}(\texttt{pk})$

$m \xleftarrow{\$} \mathcal{B}^{32}$
$m' = H(m)$ ▷ Do not output system RNG directly
$(\bar{K}, K_{\texttt{MAC}}) = G(m' \| H(\texttt{pk}))$ ▷ G is instantiated with SHA3-512
$r \xleftarrow{\$} \mathcal{R}$
$c' \leftarrow \texttt{Kyber.CPAPKE.Enc}(\texttt{pk}, m', r)$
$t = \texttt{MAC}(K_{\texttt{MAC}}, c')$
$K = \texttt{KDF}(\bar{K} \| t)$ ▷ KDF is instantiated with Shake256
$c \leftarrow (c', t)$
return (c, K)

Algorithm 14 $\texttt{Kyber.CCAKEM.Decap}^{+}(\texttt{sk}, c)$

Require: Secret key $\texttt{sk} = (\texttt{sk}', \texttt{pk}, H(\texttt{pk}), z)$
Require: Ciphertext $c = (c', t)$
$(\texttt{sk}', \texttt{pk}, h, z) \leftarrow \texttt{sk}$
$(c', t_1) \leftarrow c$
$\hat{m} = \texttt{Kyber.CPAPKE.Dec}(\texttt{sk}', c')$
$(\overline{K}, K_{\texttt{MAC}}) = G(m' \| h)$
$\hat{t} = \texttt{MAC}(K_{\texttt{MAC}}, c)$
if $\hat{t} = t$ **then**
    $K = \texttt{KDF}(\bar{K} \| t)$
else
    $K = \texttt{KDF}(z \| t)$
end if
return K

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.

4.1 MAC performance

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 the unauthenticated ciphertext. The median (top) and mean (bottom) measurements are reported in table 1.

Table 1: Standalone `MAC` performances

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

Based on the security reduction in section 3 we chose standalone GMAC and Poly1305 for their substantial performance advantage over other constructions such as CBC-MAC and KMAC. We then modified the reference implementation (TODO: citation needed https://github.com/pq-crystals/kyber) according to algorithms 12, 13, and 14. We measured the median and mean CPU cycles needed to run each of the routines. The measurements are listed in table 2

Table 2: Kyber-AE performance measurements

Name	Security level	measurement	`KeyGen`	`Encap`	`Decap`
Kyber512	128 bits	median			
		mean			
Kyber768	192 bits	median			
		mean			
Kyber1024	256 bits	median			
		mean			
KyberAE512 w/ GMAC	128 bits	median			
		mean			
KyberAE768 w/ GMAC	192 bits	median			
		mean			
KyberAE1024 w/ GMAC	256 bits	median			
		mean			
KyberAE512 w/ Poly1305	128 bits	median			
		mean			
KyberAE768 w/ Poly1305	192 bits	median			
		mean			
KyberAE1024 w/ Poly1305	256 bits	median			
		mean			

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.

[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.

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

[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.