Deniable Kyber: A Post-Quantum Deniable KEM Scheme[*]

Shih-Ming Sun and Po-Wen Chi[†]

National Taiwan Normal University, Taipei, Taiwan
{61147032s,neokent}@gapps.ntnu.edu.tw

Abstract

The concept of deniable encryption solves the situation that when a user is coerced to provide their secret key to decrypt the ciphertext, the user can mislead the outsider with forged data by providing fake proofs. However, the security of most of the deniable encryptions is based on integer factorization and discrete logarithm problems, which are considered to be insecure in the near future due to the rapid advance of quantum computing. In this work, we construct a deniable encryption scheme by modifying Kyber's key encapsulate mechanism (KEM), which won the NIST competition for the first post-quantum cryptography standard. We first construct two sets of Kyber's KEM under different modulo, merge the ciphertexts into one, and use chameleon hash instead of normal hash to the verification part in order to fulfill the deniability.

1 Introduction

The main goal of regular encryption schemes is to hide the contents of secret data. We expect that an encryption scheme can satisfy semantic security, so that an adversary will not be able to get any partial information from a ciphertext.

However, in the real world, semantic security may not be enough to protect data, and it would happen in some scenario like a user being coerced to provide its secret key to reveal its secret data. For example, the United States National Security Agency started a program known as PRISM in 2007, which was aimed at collecting data relative to potentially threat, including terrorism and cybersecurity threats. However, it cooperated with many famous companies and requested many privacy data such as chat, photo, stored data, etc. This seriously violates users' privacy; in this kind of case, the real content must be revealed since it is hard to provide another meaningful plaintext under regular encryption. Therefore, we may want to prepare forged data and convincible evidence of them to trick other people and protect our secret.

Deniable encryption, first proposed in [1], solves the problem mentioned. The main concept of it is to create a fake proof of forged data to trick the coercer. Specifically, there is a fake algorithm, when entering a message m using a randomness r and another fake message m^*, it returns a fake randomness r^*, and both $\mathrm{Enc}(m, r)$ and $\mathrm{Enc}(m^*, r^*)$ would be the same ciphertext. By doing this, the sender can claim that the ciphertext is encrypted from m^* and r^*, and the coercer cannot realize that this is a lie. In the PRISM program scenario mentioned above, we can use deniable encryption to encrypt a message or a file before upload to the internet. When we are coerced by a coercer, we provide our fake key to reveal the fake message or the fake file to keep the real one secret.

According to [1], deniable encryption can be used not only in this scenario, one of the applications is multiparty computation, which allows a set of parties to compute a common function with their input; deniable encryption can be applied here when those parties want to keep

[*]Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 21, December 16-18, 2025, Sapporo, Japan. © The copyright of this paper remains with the author(s).

[†]Corresponding Author

their input private even with coercer. Another application is electronic secret voting schemes; a coercer may bribe a voter and require a proof for their vote. With deniable encryption, the coercer cannot actually know whether the person voted for the candidate the coercer wanted or not. By using a deniable encryption scheme, users no longer need to be afraid of the threat of coercers.

Though we have the concept of deniable encryption, this is not enough, since the security of most deniable encryption schemes is based on the integer factorization and discrete logarithm problems, these problems are hard to solve by a normal computer, but they would be easily solved by quantum computing and Shor's algorithm, which is considered to be realized in the near future.

For this reason, NIST started a process to recruit some standardized quantum resist public-key cryptographic algorithms in 2016. In this process, lattice-based cryptography (LBC) received researchers' interest because security proofs were based on worst-case instances. The lattice-based encryption scheme was first proposed by Ajtai and Dwork in [2], and then simplified and improved by Regev in [3], [4]. Regev also proposed The learning with errors (LWE) problem, which was easy to use in cryptographic construction and was shown that the security is at least hard as some worst case of lattice problems. The LWE assumption states that it is hard to distinguish it from the uniform distribution of $\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}$, where $\mathbf{A}$ is a uniformly random matrix in $\mathbb{Z}_q^{m \times n}$, $\mathbf{s}$ is a uniformly random vector in $\mathbb{Z}_q^n$, and $\mathbf{e}$ is a small random error vector chosen from some distribution. Lyubashevsky et al. [5], followed this idea and a series of works, defined the Ring-LWE assumption, which is a variant of the LWE but the domain is a certain polynomial ring. Langlois et al. [6], showed a reduction from the worst-case to the average-case for the module lattices and the defined module-LWE.

By July 2022, NIST released the candidates for the post-quantum cryptographic standard, including three signatures and one key encapsulation mechanism (KEM) algorithm after three rounds of evaluation, Kyber [7] is the only KEM that is accepted, and the security is based on the Module-LWE problem mentioned above. It first constructs an IND-CPA-secure public key encryption, and then obtains a CCA-secure KEM by applying the Fujisaki-Okamoto transform to the public key encryption.

In this work, we construct a deniable encryption scheme by modifying Kyber KEM ones. We achieve deniability by building two sets of Kyber KEM under different prime modulo, using the chameleon hash and arithmetic property of modulo to merge two ciphertext of KEM into one, while we can still distinguish which set of KEM is for real key or fake key. After we have a deniable KEM scheme, we can use it to construct a deniable encryption. We first generate two keys from our deniable KEM scheme, then encrypt a real and a fake message with different keys, respectively. When being coerced, we can give the coercer a convincible fake key to reveal the fake message and leave the real message unreadable. By modifying Kyber KEM, we assure that this deniable encryption is quantum resistant according to the security of Kyber, and expect that this scheme can easily adapt to other properties.

1.1 Related work

1.1.1 Deniable Encryption

Canetti et al. [1] not only defined deniable encryption, but also constructed two practical deniable schemes using translucent sets; one is a sender-deniable scheme with $1/O(n)$-deniability and the other is a flexible sender-deniable encryption scheme with negligible deniability. Following that, O'Neill et al. [8] build a bi-deniable encryption scheme with a lattice-based bi-translucent set. Later, Sahai and Waters [9] constructed the first and only known deniable encryption achieving negligible deniability, which is based on indistinguishability obfuscation. Caro et al. [10] proposed a deniable FE and presented a receiver-deniable scheme based on indistinguishable obfuscation and delayed trapdoor circuits. An et al. [11] considered the possible damage from leakage in the real world, designed a bit-related scheme, and examined it in classic side-channel attacks. Using iO, Canetti et al. [12] achieved fully interactive deniability, which means that the scheme is not only bi-deniable, but both parties can lie with their claims being consistent.

1.1.2 Deniable encryption with quantum-resistance

Agrawal et al. [13] achieved deniable FHE, its construction is based only on the LWE polynomial hardness assumption, which is considered quantum-resistant. Their scheme can encrypt polynomial-size ciphertexts, though the running time of encryption is non-polynomial, which depends on the inverse of the detection probability. Recently, Coladangelo et al. [14] proposed a quantum analog of the deniable encryption definition, based on the definitions, restricting their attention to non-interactive public-key schemes and constructed an efficient sender-deniable encryption based on the hardness of LWE problem.

1.2 Our Contribution

We construct a bi-deniable quantum-resist KEM scheme by modifying Kyber KEM scheme, and showing that the key generated by this scheme will be safe if the secret key is not revealed even if using a quantum computer to attack.

We also claim our KEM scheme has adaptability, which means that we expect this scheme to be adapted to any other scheme based on Kyber KEM scheme, since this scheme can be considered as running two Kyber KEM schemes simultaneously, when we focus on one set of them, it is almost the same as the original Kyber KEM scheme.

2 Preliminaries

In this section, we introduce some important notations and functions in Kyber.

2.1 Mathematical Background

Let R and R_q be the rings $\mathbb{Z}[X]/(X^n+1)$ and $\mathbb{Z}_q[X]/(X^n+1)$, respectively, we use regular font letters to represent elements in R or R_q and bold lowercase letters to represent vectors with coefficients in R or R_q, bold uppercase letters are matrices. For an element $x \in \mathbb{Q}$, we write $\lceil x \rfloor$ to represent the rounding of x to the closest integer. For a set S, we write $s \leftarrow S$ to denote that s is chosen uniformly at random from S. We write $y \sim S := \mathrm{Sam}(x)$ if we produce y using an extendable output function Sam taking as input x.

2.2 Compression and Decompression

Let $x \in \mathbb{Z}_q$ and $d < \lceil \log_2(q) \rceil$, we define $\mathrm{Compress}_q(x, d) = \lceil (2^d/q) \cdot x \rfloor \bmod^+ 2^d$ and $\mathrm{Decompress}_q(x, d) = \lceil (q/2^d) \cdot x \rfloor$. When the input is a polynomial, the function applies to each coefficient individually. The main propose of these functions is to ”compress” the number into a small one, so that we can ignore some low order bit that do not affect too much with correctness of decryption. In this work, most of them are only for ”compress”, except for line 3 in decryption, which is equivalent to ”mod 2”.

2.3 Module-LWE

Let k be a positive parameter, this assumption states that it is hard to distinguish between uniform distributions $\mathbf{A}$ and $\mathbf{As}+\mathbf{e}$, where $\mathbf{A}$ is a uniform random matrix selected from $R_q^{k\times k}$, $\mathbf{s}$, and $\mathbf{e}$ are vectors selected from some distribution. Formally, for an algorithm A, the advantage is defined as

$$\begin{aligned}&\mathbf{Adv}^{\text{mlwe}}_{m,k,\eta}(A)\\ =&\left|\begin{array}{l}\Pr\left[b'=1: \begin{array}{l}\mathbf{A}\leftarrow R_q^{m\times k};\\ (\mathbf{s},\mathbf{e})\leftarrow \beta_\eta^k\times\beta_\eta^m;\\ \mathbf{b}=\mathbf{As}+\mathbf{e};\\ b'\leftarrow(\mathbf{A},\mathbf{b})\end{array}\right]\\ -\ \Pr\left[b'=1: \begin{array}{l}\mathbf{A}\leftarrow R_q^{m\times k};\\ \mathbf{b}\leftarrow R_q^m;\\ b'\leftarrow A(\mathbf{A},\mathbf{b})\end{array}\right]\end{array}\right|\end{aligned}$$

2.4 Kyber's IND-CPA-secure PKE and CCA-secure KEM

A public-key encryption scheme PKE = (KeyGen, Enc, Dec) consists of three algorithms and a message space $\mathcal{M}$, the key-generation algorithm KeyGen randomly generate a key pair (pk, sk) consisting of a public key and a secret key, the encryption algorithm Enc takes a message $m \in \mathcal{M}$ and a public key pk as input and outputs a ciphertext c, the decryption algorithm Dec takes a ciphertext c and a secret key sk as input, outputs either a message $m \in \mathcal{M}$ or a special symbol $\perp$ to indicate the decryption process failed. Following [15], we say that PKE is $(1-\delta)$-correct if $\mathbf{E}[\max_{m\in\mathcal{M}}\Pr[\text{Dec}(sk,\text{Enc}(pk,m))=m]]\geq 1-\delta$, where the expectation is taken over $(pk,sk)\leftarrow \text{KeyGen}()$ and the probability is taken over random seeds of Enc.

We recall that for public-key encryption of indistinguishability under chosen-plaintext attack (IND-CPA). The advantage of an adversary A is defined as

$$\begin{aligned}&\mathbf{Adv}^{\text{cpa}}_{\text{PKE}}(A)\\ =&\left|\ \Pr\left[b=b': \begin{array}{l}(pk,sk)\leftarrow\text{KeyGen}();\\ (m_0,m_1)\leftarrow A(pk);\\ b\leftarrow\{0,1\}; c^*\leftarrow\text{Enc}(pk,m_b);\\ b'\leftarrow A(c^*)\end{array}\right]-\frac{1}{2}\right|\end{aligned}$$

Here, we require that $|m_0|=|m_1|$.

A key-encapsulation scheme KEM = (KeyGen, Encaps, Decaps) consists of three algorithms and a key space $\mathcal{K}$, the key-generation algorithm KeyGen randomly generate a key pair (pk, sk) consisting of a public key and a secret key, the encapsulation algorithm Encaps takes a public key pk as input and outputs a ciphertext c and a key $k \in \mathcal{K}$, the decapsulation algorithm Decaps takes a ciphertext c and a secret key sk as input, outputs either a key $k \in \mathcal{K}$ or a special symbol $\perp$ to indicate the decapsulation process failed. We say that KEM is $(1-\delta)$-correct if $\Pr[\text{Decaps}(sk,c)=K : (c,K)\leftarrow\text{Encaps}(pk)]\geq 1-\delta$, where the probability takes over $(pk,sk)\leftarrow\text{KeyGen}()$ and the random seeds of Encaps.

We recall that for key-encapsulation of indistinguishability under chosen ciphertext attack.

The advantage of an adversary A is defined as

$$\mathbf{Adv}^{\text{cca}}_{\text{KEM}}(A) = \left| \Pr\left[b = b' : \begin{array}{l} (pk, sk) \leftarrow \text{KeyGen}(); \\ b \leftarrow \{0,1\}; \\ (c^*, K_0^*) \leftarrow \text{Encaps}(pk); \\ K_1^* \leftarrow \mathcal{K}; \\ b' \leftarrow A^{\text{DECAPS}(\cdot)}(pk, c^*, K_b^*) \end{array} \right] - \frac{1}{2} \right|$$

where the decapsulation oracle is defined as DECAPS$(\cdot)$ = Decaps$(sk, \cdot)$. Here, we require that the attacker is not allowed to query the decapsulation oracle with challenge ciphertext c^*.

Kyber is a quantum-resistant KEM scheme. It builds a post-quantum IND-CPA-secure PKE first and then construct a KEM for IND-CCA security bases on the PKE to . Here we introduce PKE and KEM, respectively. Kyber's IND-CPA-secure PKE consists of CPA.KeyGen, CPA.Enc, CPA.Dec, and k, d_t, d_u, d_v are positive integer parameters. In PKE key generation, the matrix $\mathbf{A}$ is randomly generated, $\mathbf{s}$ and $\mathbf{e}$ are sampled by β_η^k, then $\mathbf{t}$ is produced by computing $\mathbf{As} + \mathbf{e}$, we generate a public key $(\mathbf{t}, \rho)$ and a secret key $\mathbf{s}$ in this algorithm. In PKE encryption, m is a message that we want to encrypt, r is a random seed and vector $\mathbf{r}, \mathbf{e}_1, e_2$ are sampled by $\beta_\eta^k, \beta_\eta^k, \beta_\eta$, respectively. The ciphertext consists of $\mathbf{u}$ and v, where $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ and $v = \mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot m$. In PKE decryption, we only need to compute $v - \mathbf{s}^T\mathbf{u}$ and we can obtain the plaintext m. Note that in line 3, the second parameter of the Compress 1 function implies that the function will decrypt to 1 if the coefficient is closer to $\lceil \frac{q}{2} \rfloor$ than to 0, and decrypt to 0 otherwise. The functions are defined as follows.

- CPA.KeyGen() $\rightarrow (pk, sk)$:
  - $\mathbf{A} \sim R_q^{k\times k} := \text{Sam}(\rho)$
  - $(\mathbf{s}, \mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \text{Sam}(\sigma)$
  - $\mathbf{t} := \text{Compress}_q(\mathbf{As} + \mathbf{e}, d_t)$
  - **return** $(pk := (\mathbf{t}, \rho), sk := \mathbf{s})$
- CPA.Enc$(pk = (t, \rho), m \in \mathcal{M}) \rightarrow c$:
  - $r \leftarrow \{0,1\}^{256}$
  - $\mathbf{t} := \text{Decompress}_q(\mathbf{t}, d_t)$
  - $\mathbf{A} \sim R_q^{k\times k} := \text{Sam}(\rho)$
  - $(\mathbf{r}, \mathbf{e_1}, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \text{Sam}(r)$
  - $\mathbf{u} := \text{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e_1}, d_u)$
  - $v := \text{Compress}_q(\mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot m, d_v)$
  - **return** $c := (\mathbf{u}, v)$
- CPA.Dec$(sk = \mathbf{s}, c = (\mathbf{u}, v)) \rightarrow m$:
  - $\mathbf{u} := \text{Decompress}_q(\mathbf{u}, d_u)$
  - $v := \text{Decompress}_q(v, d_v)$
  - **return** $m = \text{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$

Kyber's CCA-secure KEM is constructed by applying the Fujisaki-Okamoto transform to Kyber-PKE, consisting of Kyber.KEM.KeyGen, Kyber.Encaps, Kyber.Decaps, $\mathrm{G} : \{0,1\}^* \to \{0,1\}^{256}$, and $\mathrm{H} : \{0,1\}^* \to \{0,1\}^{256}$ being hash functions. In key generation, it is almost the same as CPA.KeyGen, except that the secret key not only contains $\mathbf{s}$ but also contains $pk = (\mathbf{t}, \rho)$ and a random secret value z. In encapsulation, it hash a message m by $\mathrm{G}(\mathrm{H}(pk), m)$ and obtains random seeds $\hat{K}$ and r, then encrypt this message using Kyber.CPA.Enc and random seed r, obtains ciphertext c. The real key K is generated by $\mathrm{H}(\hat{K}, \mathrm{H}(c))$ In decapsulation, it first decrypts c by Kyber.CPA.Dec to get plaintext m', then **re-encrypts** m' to get $\hat{K}'$ and $(\mathbf{u}', v')$. If the result of re-encryption $(\mathbf{u}', v')$ is equal to the ciphertext $(\mathbf{u}, v)$, then it can get the real key $K := \mathrm{H}(\hat{K}', \mathrm{H}(c))$; otherwise, it outputs a random key by computing $\mathrm{H}(z, \mathrm{H}(c))$.

- Kyber.KEM.KeyGen() $\to (pk, sk)$:
  - $z \leftarrow \{0,1\}^{256}$
  - $\mathbf{A} \sim R_q^{k\times k} := \mathrm{Sam}(\rho)$
  - $(\mathbf{s}, \mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \mathrm{Sam}(\sigma)$
  - $\mathbf{t} := \mathrm{Compress}_q(\mathbf{As} + \mathbf{e}, d_t)$
  - **return** $(pk := (\mathbf{t}, \rho), sk := (\mathbf{s}, z, \mathbf{t}, \rho))$
- Kyber.Encaps$(pk = (\mathbf{t}, \rho)) \to (c, K)$:
  - $m \leftarrow \{0,1\}^{256}$
  - $(\hat{K}, r) := G(H(pk), m)$
  - $(\mathbf{u}, v) :=$ Kyber.CPA.Enc$((\mathbf{t}, \rho), m; r)$
  - $c := (\mathbf{u}, v)$
  - $K := H(\hat{K}, H(c))$
  - **return** (c, K)
- Kyber.Decaps$(sk = (\mathbf{s}, z, \mathbf{t}, \rho), c = (\mathbf{u}, v)) \to K$:
  - $m' :=$ Kyber.CPA.Dec$(\mathbf{s}, (\mathbf{u}, v))$
  - $(\hat{K}', r') := G(H(pk), m')$
  - $(\mathbf{u}', v') :=$ Kyber.CPA.Enc$((\mathbf{t}, \rho), m'; r')$
  - **if** $(\mathbf{u}', v') = (\mathbf{u}, v)$,

    return $K := H(\hat{K}', H(c))$;

    otherwise,

    return $K := H(z, H(c))$.

2.5 Chameleon Hash

Chameleon hash was first introduced in [16]. This hash function has three properties; two of them are like other hash functions that have *collision resistance* and *semantic security* properties, and the other is *trapdoor collisions*, an additional property.

We use CHS to represent the chameleon hash scheme, the scheme is composed of the following algorithms:

- CHS.KeyGen(1^λ) $\rightarrow (PK, SK)$. The algorithm takes a security parameter λ as input, outputs a public key PK and a secret key(trapdoor) SK.
- CHS.HashFunc(PK) $\rightarrow CH$. The algorithm takes a public key PK as input and outputs a hash function CH.
- CHS.Hash(CH, m, r) $\rightarrow V$. The algorithm takes a hash function CH, a message m and a random seed r as input and outputs a value V. Since the algorithm only calculates $CH(m, r)$, we denote $CH(m, r)$ when calculating the hash value in the rest of this work.
- CHS.Collision(CH, SK, m_0, m_1, r_0) $\rightarrow r_1$. The algorithm takes a hash function CH, a secret key SK, two messages m_0, m_1 and a random seed r_0 as input, outputs a seed r_1.

The details of each property are described below.

Definition 1 (Collision Resistance)**.** *Given chameleon hash scheme $\{PK, SK, CH(\cdot,\cdot)\}$, where PK is the public key, SK is a trapdoor and $CH(\cdot,\cdot)$ is the hash function. Let m, m' be two messages and r be a random value, we say the scheme is collision resistant if there is no efficient algorithm to output r' such that $CH(m, r) = CH(m', r')$ without SK.*

Definition 2 (Semantic Security)**.** *Given chameleon hash scheme $\{PK, SK, CH(\cdot,\cdot)\}$, where PK is the public key, SK is a trapdoor and $CH(\cdot,\cdot)$ is the hash function. We say that the scheme is semantically secure if for any pair of messages m, m' and a random value r, the probability distributions of $CH(m, r)$ and $CH(m', r)$ are computationally indistinguishable.*

Definition 3 (Trapdoor Collisions)**.** *Given chameleon hash scheme $\{PK, SK, CH(\cdot,\cdot)\}$, where PK is the public key, SK is a trapdoor and $CH(\cdot,\cdot)$ is the hash function. Let m, m' be two messages and r be a random value. We say that the trapdoor collision scheme is successful if there is an efficient algorithm such that, given SK, it can output a value r' such that $CH(m, r) = CH(m', r')$.*

3 Quantum-resist deniable key encapsulation scheme construction

3.1 An overview of our deniable KEM scheme

We first give an overview of how our deniable KEM scheme works. We use the multi-distributional deniable encryption concept here. That is, there are two KEM schemes, one is a normal KEM scheme and the other one is a deniable KEM scheme. So, the user can claim that it is using the normal KEM scheme while it is actually using the deniable one.

We first introduce our deniable KEM scheme, as in Fig. 1. We build two Kyber KEM schemes under different modulo and add an additional step called "merge process" to merge ciphertexts of two KEM into one. The merging process first finds the specific p_0, p_1, then merges $(\mathbf{u}_0, v_0)$ and $(\mathbf{u}_1, v_1)$ by applying the Chinese remainder theorem. To achieve deniability, we use a chameleon hash instead of normal hash as our comparator and generate a coin b at the end to be our encryption proof. In decapsulation, we take modulo q as input to specify which KEM we are going to decapsulate, and as in the merge process, we use chameleon hash to compare the re-encryption result and generate a coin at the end. In the normal KEM scheme, instead of building two sets of the Kyber KEM scheme, we randomly generate $(\mathbf{u}_1, v_1)$ and run the merge process. There is an algorithm called "Verify" for coercer to check whether the key is convincing.

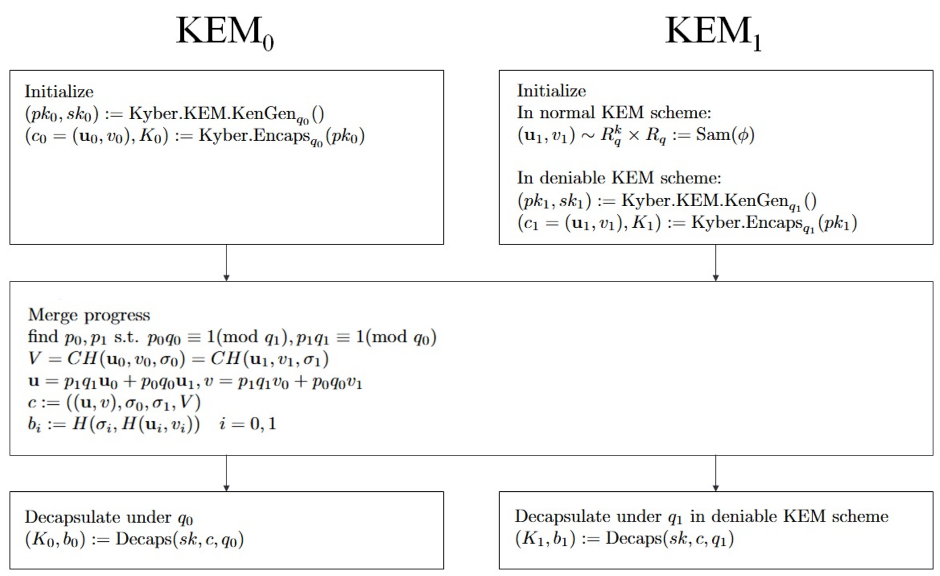
KEM_0
KEM_1
Initialize
$(pk_0, sk_0) := \text{Kyber.KEM.KenGen}_{q_0}()$
$(c_0 = (\mathbf{u}_0, v_0), K_0) := \text{Kyber.Encaps}_{q_0}(pk_0)$
Initialize
In normal KEM scheme:
$(\mathbf{u}_1, v_1) \sim R_q^k \times R_q := \text{Sam}(\phi)$
In deniable KEM scheme:
$(pk_1, sk_1) := \text{Kyber.KEM.KenGen}_{q_1}()$
$(c_1 = (\mathbf{u}_1, v_1), K_1) := \text{Kyber.Encaps}_{q_1}(pk_1)$
Merge progress
find p_0, p_1 s.t. $p_0 q_0 \equiv 1(\text{mod } q_1), p_1 q_1 \equiv 1(\text{mod } q_0)$
$V = CH(\mathbf{u}_0, v_0, \sigma_0) = CH(\mathbf{u}_1, v_1, \sigma_1)$
$\mathbf{u} = p_1 q_1 \mathbf{u}_0 + p_0 q_0 \mathbf{u}_1, v = p_1 q_1 v_0 + p_0 q_0 v_1$
$c := ((\mathbf{u}, v), \sigma_0, \sigma_1, V)$
$b_i := H(\sigma_i, H(\mathbf{u}_i, v_i)) \quad i = 0, 1$
Decapsulate under q_0
$(K_0, b_0) := \text{Decaps}(sk, c, q_0)$
Decapsulate under q_1 in deniable KEM scheme
$(K_1, b_1) := \text{Decaps}(sk, c, q_1)$

Figure 1: An overview of our deniable KEM scheme

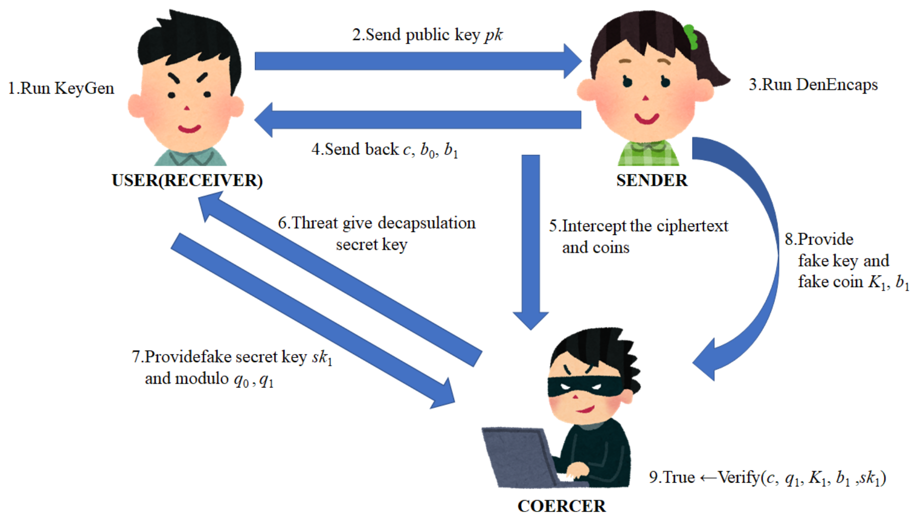
1.Run KeyGen
2.Send public key pk
3.Run DenEncaps
4.Send back c, b_0, b_1
USER(RECEIVER)
SENDER
6.Threat give decapsulation secret key
5.Intercept the ciphertext and coins
8.Provide fake key and fake coin K_1, b_1
7.Providefake secret key sk_1 and modulo q_0 , q_1
9.True $\leftarrow$Verify(c, q_1, K_1, b_1 ,sk_1)
COERCER

Figure 2: A coercion scenario

Let KEM_1 be the KEM that generates the fake key. In normal situations, the sender can release b_0 and the receiver can use sk_0 to verify the decapsulation is correct; when coerced, the sender will release b_1 and the receiver will release sk_1, the coercer will be convinced that the verification algorithm returns true. We demonstrate a situation where a user is being coerced, as in Fig. 2.

3.2 Definition of Deniable KEM

We first introduce the following property about deniable KEM:

- sender-, receiver-, and bi-deniability. The prefix indicates which role can fool the coercer with convincible evidence. Sender-deniability and receiver-deniability assume that the other role can not fool the coercer, while bi-deniability assumes that both of sender and receiver can fool the coercer.

- full deniability and multi-distributional deniability. Fully deniable KEM means that there is only one set of algorithm (KeyGen, Encaps, Decaps), all of senders, receivers, and coercers know the algorithm, and a sender or a receiver still can fool coercers with the KEM scheme. Multi-distributional deniable KEM means there are two sets of algorithm, one is for normal encapsulation and the other one is for deniable encapsulation. The normal one cannot be used to fool coercer, but the deniable one can. A sender or a receiver can fool a coercer by claiming that they are using the normal one to encapsulate key while they are using the deniable one indeed.

Based on the definition, our bi-deniable and multi-distributional KEM scheme is composed of the following algorithms:

- KeyGen$(1^\lambda) \rightarrow (pk, sk)$. This algorithm takes a security parameter λ as input, returns a public key pk and a secret key sk.

- Encaps$(pk) \rightarrow (c, K_0, b_0)$. This algorithm takes a public key pk as input and outputs a ciphertext c, a key K_0 and a coin b_0.

- Decaps$(sk, c) \rightarrow (K, b)$. This algorithm takes a secret key sk and a ciphertext c as input, outputs a key K and a coin b.

- Verify$(c, K, b_i, sk) \rightarrow \{T, F\}$. This algorithm takes a ciphertext c, a key K, a coin b_i, and a secret key sk as input and outputs a boolean value.

- DenKeyGen$(1^\lambda) \rightarrow (pk, sk_0, sk_1)$. This algorithm takes a security parameter λ as input, returns a public key pk and two secret keys sk_0, sk_1.

- DenEncaps$(pk) \rightarrow (c, K_0, K_1, b_0, b_1)$. This algorithm takes a public key pk as input, outputs a ciphertext c, two keys K_0, K_1, and two coins b_0, b_1.

- DenDecaps$(sk, c) \rightarrow (K, b)$. This algorithm takes a secret key sk and a ciphertext c as input, outputs a key K and a coin b.

We require the following properties:

- Security. The tuple (KeyGen, Encaps, Decaps) must form a secure KEM scheme. In this work, we require these algorithms to satisfy the CCA-secure KEM definition described in section 2.4.

- Bi-deniability. We say a KEM scheme is bi-deniable if the two distribution tuples (c, K, b, sk) and (c', K', b', sk') are computationally indistinguishable, where c, K, c', K' are ciphertexts and keys generated by the normal and deniable encapsulation algorithm, respectively, b and b' are coins generated by the normal and deniable encapsulation algorithm, respectively, sk and sk' are secret keys.

3.3 Deniable KEM construction

Let d_{t_0}, d_{t_1} be positive integer parameters, $G : \{0,1\}^* \to \{0,1\}^{256}$ and $H : \{0,1\}^* \to \{0,1\}^{256}$ be hash functions. Since we use two modulo to build our scheme, we let $\text{Algorithm}_q()$ denote the algorithm or function that runs under modulo q.

In KeyGen, the algorithm takes a security parameter λ as input, output pk, sk. We generate q_0, q_1 and find p_0, p_1 for the merge progress in advance, then follow the Kyber KEM steps.

- $\text{KeyGen}(1^\lambda) \to (pk, sk_0)$:
  - Generate two different primes q_0, q_1.
  - Find p_0, p_1 for the Chinese remainder theorem with the following condition:
$$p_0 q_0 \equiv 1 (\text{mod } q_1), p_1 q_1 \equiv 1 (\text{mod } q_0).$$
  - Set up the Module-LWE problem,
$$z, \rho, \phi_0, \phi_1 \leftarrow \{0,1\}^{256},$$
$$\mathbf{A} \sim R_{q_0}^{k \times k} := \text{Sam}(\rho),$$
$$(\mathbf{s}_0, \mathbf{e}_0) \sim \beta_\eta^k \times \beta_\eta^k := \text{Sam}(\phi_0),$$
$$\mathbf{t}_0 := \text{Compress}_{q_0}(\mathbf{A}\mathbf{s}_0 + \mathbf{e_0}, d_{t_0}).$$
  - Generate $\mathbf{t}_1$ for pk.
$$\mathbf{t}_1 \sim R_{q_0}^k := \text{Sam}(\phi_1).$$
  - Define the public key pk and the secret key sk_0,
$$pk := (\mathbf{t}_0, \mathbf{t}_1, \rho, q_0, q_1, p_0, p_1),$$
$$sk_0 := (\mathbf{s}_0, z, \mathbf{t}_0, \rho, q_0).$$
  - **return** (pk, sk_0)

In the encapsulation algorithm, we generate $(\mathbf{u}_0, v_0)$ and K_0 using the same process as Kyber.Encaps. Since this scheme is for normal encapsulation, we generate $(\mathbf{u}_1, v_1)$ randomly and then use the chameleon hash to generate the verifier and pseudo coin (σ_0, σ_1). Finally, the ciphertext c yielded by the Chinese remainder theorem and a coin for encryption proof generated by hashing, the algorithm then returns a ciphertext c, a key K_0 and a coin b_0.

- $\text{Encaps}(pk) \to (c, K_0, b_0)$:
  - Generate a random message m_0, a random seed ϕ and a random value σ_1,
$$m_0, \phi, \sigma_1 \leftarrow \{0,1\}^{256}.$$
  - Use the hash function to produce the pseudo key $\hat{K}_0$ and the random seed r_0,
$$(\hat{K}_0, r_0) := G(H(\mathbf{t}_0, \rho), m_0).$$

- Encrypt ciphertext for KEM_0 and randomly assign a ciphertext for KEM_1,

$$(\mathbf{u}_0, v_0) := \text{CPA.Enc}_{q_0}((\mathbf{t}_0, \rho), m_0; r_0),$$

$$(\mathbf{u}_1, v_1) \sim R_q^k \times R_q := \text{Sam}(\phi).$$

- Generate a verifier for decapsulation,

$$(CHS.PK, CHS.SK) := \text{CHS.KeyGen}(1^\lambda),$$

$$CH := \text{CHS.HashFunc}(CHS.PK),$$
$$V = CH(\mathbf{u}_0, v_0, \sigma_0).$$

- Use the Chinese remainder theorem to merge the ciphertexts,

$$\mathbf{u} = p_1 q_1 \mathbf{u}_0 + p_0 q_0 \mathbf{u}_1,$$

$$v = p_1 q_1 v_0 + p_0 q_0 v_1.$$

- Define the ciphertext c, the key K_0 and the coin b_0

$$c := ((\mathbf{u}, v), \sigma_0, \sigma_1, CHS.PK, V),$$

$$K_0 := H(\hat{K}_0, H(\mathbf{u}_0, v_0)),$$
$$b_0 := H(\sigma_0, H(\mathbf{u}_0, v_0)).$$

- **return** (c, K_0, b_0).

In the decapsulation algorithm, we re-encrypt c to produce $(\mathbf{u}_0, v_0)$, then the comparator returns a key K and a coin b. Following the design of Kyber KEM, we let the coin b be a random value if the decapsulation failed.

- Decaps$(sk, c) \to (K, b)$:
  - Recover ciphertext $c_q = (\mathbf{u}_q, v_q)$ for KEM under modulo q

$$\mathbf{u}_q = \mathbf{u} \text{ (mod q)},$$

$$v_q = v \text{ (mod q)}.$$

  - Decrypt c_q to yield message m',

$$m' := \text{CPA.Dec}_q(\mathbf{s}, (\mathbf{u}_q, v_q)).$$

  - Re-encrypt m' to yield ciphertext $(\mathbf{u}', v')$,

$$(\hat{K}', r') := G(H(\mathbf{t}, \rho), m'),$$

$$(\mathbf{u}', v') := \text{CPA.Enc}_q((\mathbf{t}, \rho), m'; r').$$

  - Rebuild chameleon hash function CH

$$CH := \text{CHS.HashFunc}(CHS.PK).$$

- **if** $\exists i \in \{0,1\}, CH(\mathbf{u}', v', \sigma_i) = V$,

$$K = H(\hat{K}', H(\mathbf{u}', v')),$$
$$b = H(\sigma_i, H(\mathbf{u}', v'));$$

  otherwise,

$$K = H(z, H(\mathbf{u}', v')),$$
$$b = H(\sigma_0, H(\mathbf{u}', v')).$$

- **return** (K, b).

For the verification algorithm, the sender will provide K and b_i, the receiver will provide sk. This algorithm first decapsulates c using sk and q and produces (K', b'), then checks whether these values are the same as the values sent by the sender, returning true or false.

- Verify$(c, K, b_i, sk) \to \{T, F\}$:
  - $(K', b') :=$ Decaps(sk, c)
  - **if** $K' = K, b' = b$,

    return TRUE;

    otherwise,

    return FALSE.

In deniable version, we build two sets of Kyber KEM under different modulo with the same randomness ρ, output one public key, two secret keys and numbers for Chinese remainder theorem.

- DenKeyGen$(1^\lambda) \to (pk, sk_0, sk_1)$:
  - Generate two different primes q_0, q_1.
  - Find p_0, p_1 for the Chinese remainder theorem with the following condition:

$$p_0 q_0 \equiv 1 (\text{mod } q_1), p_1 q_1 \equiv 1 (\text{mod } q_0).$$

  - For $i = 0, 1$, set up the Module-LWE problem,

$$z, \rho, \phi_0, \phi_1 \leftarrow \{0,1\}^{256},$$
$$\mathbf{A}_i \sim R_{q_i}^{k \times k} := \text{Sam}(\rho),$$
$$(\mathbf{s}_i, \mathbf{e}_i) \sim \beta_\eta^k \times \beta_\eta^k := \text{Sam}(\phi_i),$$
$$\mathbf{t}_i := \text{Compress}_{q_i}(\mathbf{A}_i \mathbf{s}_i + \mathbf{e}_i, d_{t_i}).$$

  - Define the public key pk and secret keys sk_0, sk_1,

$$pk := (\mathbf{t}_0, \mathbf{t}_1, \rho, q_0, q_1, p_0, p_1),$$
$$sk_0 := (\mathbf{s}_0, z, \mathbf{t}_0, \rho, q_0),$$
$$sk_1 := (\mathbf{s}_1, z, \mathbf{t}_1, \rho, q_1).$$

- **return** (pk, sk_0, sk_1).

The deniable encapsulation algorithm generates $(\mathbf{u}_0, v_0), (\mathbf{u}_1, v_1)$ and K_0, K_1 by the same process as Kyber.Encaps, use chameleon hash to generate comparator and pseudo coin (σ_0, σ_1), then a ciphertext c is obtained by the Chinese remainder theorem and two coins for encryption proof generated by hashing, the algorithm then returns a ciphertext c, two keys K_0, K_1, and two coins b_0, b_1.

- DenEncaps$(pk) \to (c, K_0, K_1, b_0, b_1)$:
  - Generate random messages m_0, m_1 and a random seed σ_0,
  $$m_0, m_1, \sigma_0 \leftarrow \{0,1\}^{256}.$$
  - For $i = 0, 1$, use the hash function to produce a pseudo-key $\hat{K}_i$ and a random seed r_i,
  $$(\hat{K}_i, r_i) := G(H(\mathbf{t}_i, \rho), m_i).$$
  - For $i = 0, 1$, encrypt the ciphertext for KEM_i,
  $$(\mathbf{u}_i, v_i) := \text{CPA.Enc}_{q_i}((\mathbf{t}_i, \rho), m_i; r_i).$$
  - Generate a verifier for decapsulation,
  $$(CHS.PK, CHS.SK) := \text{CHS.KeyGen}(1^\lambda),$$
  $$CH := \text{CHS.HashFunc}(CHS.PK),$$
  $$\sigma_1 := \text{CHS.Collision}(CH, CHS.SK, (\mathbf{u}_0, v_0), (\mathbf{u}_1, v_1), \sigma_0),$$
  $$V = CH(\mathbf{u}_0, v_0, \sigma_0) = CH(\mathbf{u}_1, v_1, \sigma_1).$$
  - Use the Chinese remainder theorem to merge the ciphertexts
  $$\mathbf{u} = p_1 q_1 \mathbf{u}_0 + p_0 q_0 \mathbf{u}_1, v = p_1 q_1 v_0 + p_0 q_0 v_1.$$
  - Define the ciphertext c, the keys K_0, K_1 and the coins b_0, b_1,
  $$c := ((\mathbf{u}, v), \sigma_0, \sigma_1, CHS.PK, V),$$
  $$K_i := H(\hat{K}_i, H(\mathbf{u}_i, v_i)) \quad i = 0, 1,$$
  $$b_i := H(\sigma_i, H(\mathbf{u}_i, v_i)) \quad i = 0, 1.$$
  - **return** (c, K_0, K_1, b_0, b_1).

There should be an algorithm for deniable decapsulation; however, this algorithm is actually the same as Decaps, since both of them are doing re-encrypt and output a key and a coin with the same input, therefore we do not need a decapsulation algorithm for deniable version.

3.4 Correctness

To prove correctness, we follow the step of Decaps and compute it directly. Since

$$\mathbf{u}_{q_i} = (p_1 q_1 \mathbf{u}_0 + p_0 q_0 \mathbf{u}_1)_{q_i} = p_{1-q_i} q_{1-q_i} \mathbf{u}_{q_i} = \mathbf{u}_i,$$

$$v_{q_i} = (p_1 q_1 v_0 + p_0 q_0 v_1)_{q_i} = p_{1-q_i} q_{1-q_i} v_{q_i} = v_i,$$

When we use $\mathbf{s}_i, q_i$ to decapsulate, calculate Decaps(sk_i, c), we have:

$$m' := \text{CPA.Dec}_{q_i}(\mathbf{s}_i, (\mathbf{u}_{q_i}, v_{q_i})) = \text{CPA.Dec}_{q_i}(\mathbf{s}_i, (\mathbf{u}_i, v_i)) = m_i,$$

$$(\hat{K}', r') := \text{G}(\text{H}(\mathbf{t}, \rho), m') = \text{G}(\text{H}(pk), m) = (\hat{K}_i, r_i),$$

$$\begin{aligned}(\mathbf{u}', v') &= \text{CPA.Enc}_{q_i}((\mathbf{t}_i, \rho), m'; r') \\ &= \text{CPA.Enc}_{q_i}((\mathbf{t}_i, \rho), m_i; r_i) \\ &= (\mathbf{u_i}, v_i).\end{aligned}$$

So we can show $(K, b) = (K_i, b_i)$ as follows.

$$\begin{aligned}(K, b) &= (H(\hat{K}', H(\mathbf{u}', v')), H(\sigma_i, H(\mathbf{u}', v'))) \\ &= (H(\hat{K}_i', H(\mathbf{u}_i', v_i')), H(\sigma_i, H(\mathbf{u}_i', v_i'))) \\ &= (K_i, b_i)\end{aligned}$$

since there exists σ_i such that $\text{CH}(\mathbf{u}', v', \sigma_i) = (\mathbf{u}_i, v_i, \sigma_i) = V$. Therefore, we can have the same key K_i and coin b_i as in Encaps, which also means that Verify(c, K, b_i, sk) will return true no matter what the secret key and coin are for real or fake encapsulation.

On the other hand, when we use $\mathbf{s}_i, q_{1-i}$ to decapsulate, which means that $\mathbf{s}_i$ does not correspond to q_{1-i}, in the first step it will not get a valid m which is generated in Encaps, we will get a random key $H(z, H(\mathbf{u}', v'))$ and a random coin, and end up with the algorithm Verify returning False.

3.5 Security Proof

Since our scheme is a modified version of Kyber's KEM scheme, we prove that our deniable encryption scheme is secure by showing that the scheme is secure if the corresponding Kyber scheme is secure. For the CPA part, there is nothing to prove since they are actually the same in different notation, and hence we only need to prove the KEM part.

Theorem 1. *Our proposed deniable KEM scheme is CCA-secure if Kyber's KEM scheme is CCA-secure.*

Proof. We prove this theorem by showing that if there exists an algorithm that can break our deniable KEM scheme, then this algorithm can also break the original Kyber's KEM scheme. We recall that the main concept of our deniable scheme is to merge two sets of Kyber KEM; so our plan is to extend a Kyber's KEM to our scheme and then break the Kyber's KEM. Consider that a Kyber's KEM scheme is already set up, then we may have the following parameter: $pk_0 = (\mathbf{t}_0, \rho), q_0, c_0 = (\mathbf{u}_0, v_0), K_0$, we first choose another prime $q_1 \neq q_0$ and then extend the KEM scheme to our deniable KEM scheme by the algorithms ExtDenKeyGen and ExtDenEncaps.

In ExtDenKeyGen, the algorithm takes pk_0, q_0 as input, uses the same random seed ρ to set up the Module-LWE problem, and outputs the same parameters as those in DenKeyGen.

- ExtDenKeyGen$(pk_0, q_0) \rightarrow (pk, sk_1)$:

- Generate a prime q_1 different from q_0.
- Find p_0, p_1 for the Chinese remainder theorem with the following condition:

$$p_0 q_0 \equiv 1 (\text{mod } q_1), p_1 q_1 \equiv 1 (\text{mod } q_0)$$

- Set up the Module-LWE problem,

$$z, \phi \leftarrow \{0,1\}^{256}, \mathbf{A} \sim R_{q_1}^{k \times k} := \text{Sam}(\rho),$$

$$(\mathbf{s}_1, \mathbf{e}_1) \sim \beta_\eta^k \times \beta_\eta^k := \text{Sam}(\phi),$$
$$\mathbf{t}_1 := \text{Compress}_{q_1}(\mathbf{As} + \mathbf{e}, d_{t_1}).$$

- Define the public key pk and the secret key sk_1,

$$pk := (\mathbf{t}_0, \mathbf{t_1}, \rho, q_0, q_1, p_0, p_1), sk_1 := (\mathbf{s}_1, z, \mathbf{t}_1, \rho, q_1).$$

- **return** $(pk, sk_0, sk_1, p_0, p_1)$

In ExtDenEncaps, the algorithm takes pk, c, K_0 as input, completes the encapsulation step for the new KEM set, generates a verifier and coins for both sets of KEM and outputs the same parameters as in DenEncaps.

- ExtDenEncaps$(pk, c_0, K_0) \to (c, K_0, K_1, b_0, b_1)$:
  - Generate a random message m_1 and a random seed σ_0,

$$m_1 \leftarrow \{0,1\}^{256}.$$

  - Use the hash function to produce pseudo key $\hat{K}_1$ and random seed r_1,

$$(\hat{K}_1, r_1) := G(H(\mathbf{t}_1, \rho), m_1).$$

  - Encrypt ciphertext for KEM_1,

$$(\mathbf{u}_1, v_1) := \text{CPA.Enc}_{q_1}((\mathbf{t}_1, \rho), m_1; r_1).$$

  - Generate a verifier for decapsulation,

$$(CHS.PK, CHS.SK) := \text{CHS.KeyGen}(1^\lambda),$$

$$CH := \text{CHS.HashFunc}(CHS.PK),$$
$$\sigma_1 := \text{CHS.Collision}(CH, CHS.SK, (\mathbf{u}_0, v_0), (\mathbf{u}_1, v_1), \sigma_0),$$
$$V = CH(\mathbf{u}_0, v_0, \sigma_0) = CH(\mathbf{u}_1, v_1, \sigma_1).$$

  - Using the Chinese remainder theorem to merge the ciphertexts,

$$\mathbf{u} = p_1 q_1 \mathbf{u}_0 + p_0 q_0 \mathbf{u}_1, v = p_1 q_1 v_0 + p_0 q_0 v_1.$$

  - Define the ciphertext c, the key K_1 and the coins b_0, b_1,

$$c := ((\mathbf{u}, v), \sigma_0, \sigma_1, CHS.PK, V),$$

$$K_1 := H(\hat{K}_i, H(\mathbf{u}_i, v_i)),$$
$$b_i := H(\sigma_i, H(\mathbf{u}_i, v_i)) \quad i = 0, 1.$$

– **return** (c, K_0, K_1, b_0, b_1).

We recall that in our decapsulation algorithm, decapsulating with the corresponding prime q is equivalent to decapsulating in the corresponding set of KEM. Therefore, in order to break the Kyber's KEM, we only need to run Decap(sk_0, c), and we will have the same $m', (\hat{K}', r'), (\mathbf{u}', v')$ as in the Kyber's KEM. Since the verification step will pass in both schemes, we will finally get the same K by computing $H(\hat{K}', H(\mathbf{u}', v'))$. By the above discussion, we can conclude that if one can break our deniable KEM scheme, then for any Kyber's KEM scheme, we can extend it to our deniable KEM scheme and then break the Kyber's KEM scheme. □

3.6 Deniability Proof

Let (c, K_0, b_0, sk_0) be the input of $Verify$ for the normal KEM scheme and (c', K_0', b_0', sk_0'), (c', K_1', b_1', sk_1') for the deniable KEM scheme when we want to verify. The tuple (c', K_0', b_0', sk_0') is for the encapsulation of the real key and the tuple (c', K_1', b_1', sk_1') is for the fake. To prove that our KEM scheme has deniability, we need to show that (c, K_0, b_0, sk_0) and (c', K_1', b_1', sk_1') are indistinguishable. Since (c, K, b, sk) are pairwise independent by security property, we need only show the indistinguishability between c and c', K_0 and K_1', b_0 and b_1', and sk_0 and sk_1'.

Lemma 1. *Under the LWE assumption, the secret keys for the normal KEM scheme sk_0 and the deniability scheme sk_1' are indistinguishable.*

Proof. The secret key we use in the KEM scheme contains $\mathbf{s}, z, \mathbf{t}, \rho$ and q, here $\mathbf{s}$ is a random vector, z and ρ are random values, q is a prime generated by the security parameter, so these parameters are indistinguishable. Furthermore, by the definition of the hardness of the Module-LWE assumption, we know that it is hard to distinguish $\mathbf{t}$ from the uniform sample $a_i \leftarrow R_q^k$, therefore it is also hard to distinguish $\mathbf{t}_0$ and $\mathbf{t}_1'$. By the above discussion, we can conclude that the secret key for the normal KEM scheme sk_0 and for the deniable scheme sk_1' are indistinguishable. □

Lemma 2. *Under the security property of the Kyber PKE scheme and the chameleon hash function, the normal ciphertext c and the deniable ciphertext c' are computationally indistinguishable.*

Proof. The ciphertext c consists of parameters $(\mathbf{u}, v), \sigma_0, \sigma_1, CHS.PK$ and V. For $(\mathbf{u}, v)$ and $(\mathbf{u}', v')$, the former is a linear combination of ciphertexts $(\mathbf{u}_0, v_0)$ from CPA.Enc and $(\mathbf{u}_1, v_1)$ chosen randomly from a uniform distribution, while the latter is a linear combination of ciphertexts $(\mathbf{u}_0', v_0')$ and $(\mathbf{u}_1', v_1')$ from CPA.Enc. Since random parameters $(\mathbf{u}_1, v_1)$ are indistinguishable from the ciphertext generated by CPA.Enc, we can regard it as an output of CPA.Enc, now $(\mathbf{u}_0, v_0), (\mathbf{u}_1, v_1), (\mathbf{u}_0', v_0'), (\mathbf{u}_1', v_1')$ can be considered as outputs of CPA.Enc, which means that those parameters are indistinguishable. For the rest parameters, Since $CHS.PK$ is a key generated by the security parameter; σ_0, σ_1 are randomness found by the chameleon hash, V is the output of the hash function with indistinguishable input $(\mathbf{u}, v)$ and randomness σ, by the collision resistance property of the chameleon hash, it is indistinguishable between σ_i and σ_i', between $CHS.PK$ and $CHS.PK'$, and between V and V'. By the above discussion, we can conclude that c and c' are indistinguishable.

□

Lemma 3. *According to the security property of the hash function, the keys for the normal KEM scheme K_0 and for the deniable scheme K_1' are indistinguishable.*

Proof. In the encapsulation algorithm, the pseudo key $\hat{K}_i$ generated by hash G with a random message input m_i, so we may consider the pseudo key $\hat{K}_i$ as randomness. Since the key K is the output of a hash function with indistinguishable input $(\mathbf{u}_i, v_i)$ and randomness $\hat{K}_i$, we can ensure that the keys K_0 and K'_1 are indistinguishable. □

Lemma 4. *According to the security property of the hash function, coins for the normal KEM scheme b_0 and for the deniable KEM scheme b'_1 are indistinguishable.*

Proof. Since the coin b_i is the output of a hash function with indistinguishable input $(\mathbf{u}_i, v_i)$ and randomness σ_i, we can ensure that the coins b_0 and b'_1 are indistinguishable. □

By Lemma 1 to 4 and the fact that the PKE scheme is an IND-CPA secure scheme, we know that our KEM scheme is deniable. According to [7], we also know that the PKE scheme is IND-CPA secure under the module-LWE hardness assumption. By the above discussion, we come to the following conclusion:

Theorem 2. *Our KEM scheme is deniable under the Module-LWE hardness assumption.*

4 Performance Evaluation

In this section, we evaluate the performance of our scheme. The testing environment is a computer with an Intel (R) Core (TM) i7-7700 CPU, and 16 GB memory. The experiment is based on an existing Kyber KEMs code [17] and a chameleon hash code [18]; for deniable encapsulation, we run the experiments with $q = 3329$ twice to simulate the running time of two KEMs and implement our scheme using a chameleon hash once for security reasons. The running times of our algorithms are listed in the TABLE 1. Note that there is no deniable decapsulation algorithm, the Decaps entries of our scheme in Table 1 are the same.

Comparing our KEM scheme to Kyber KEM scheme, the difference is that our scheme has two sets and an additional merge process, and the merge process uses only normal hash function, chameleon hash function, vector addition and multiply a coefficient to a vector. The most time consuming part for the merge process is the key generation step for chameleon hash function, which takes more than 1 millisecond in our experiment, therefore, we estimate that the running time of our algorithm is about twice of the running time of Kyber KEM algorithm adding the running time of key generation progress for chameleon hash. Although the chameleon hash causes the running time of encapsulation to significantly grow, but we also gained the deniability property, and we think this is a reasonable trade off for our scheme. In practice, this could be improved by running the chameleon hash key generation algorithm in advance, since the algorithm only takes a security parameter as input.

5 Discussion

In this work, we use the factoring property to construct a redundant space to embed fake messages. This approach raises an important question: if the quantum computer can break the integer factorization problem, how can we say that our scheme is secure despite we generate two KEM under different prime modulo? In fact, our deniable KEM scheme is still safe even if we reveal all the primes we use, this is because the security of this scheme is based on the secret key, as described in correctness proof, one will not decapsulate key correctly if they don't have the corresponding secret key, so we can assure that our KEM scheme is safe even if the integer factorization problem being solve.

Table 1: Running time of our deniable KEM scheme

		KYBER512	KYBER768	KYBER1024
Original Kyber	KeyGen	0.03264 ms	0.05542 ms	0.08917 ms
	Encaps	0.04121 ms	0.06343 ms	0.09531 ms
	Decaps	0.05187 ms	0.07757 ms	0.11285 ms
Ours (Normal)	KeyGen	0.03335 ms	0.05707 ms	0.09025 ms
	Encaps	2.06551 ms	2.11032 ms	2.17408 ms
	Decaps	0.10396 ms	0.15526 ms	0.22682 ms
Ours (Deniable)	DenKeyGen	0.06598 ms	0.11214 ms	0.17937 ms
	DenEncaps	2.06561 ms	2.11093 ms	2.17505 ms
	Decaps	0.10396 ms	0.15526 ms	0.22682 ms

6 Conclusion

In this work, we constructed a bi-deniable multi-distributional quantum-resist KEM scheme, which is a modify version of Kyber KEM scheme and the security is based on Module-LWE problem. We proved that the real key will not be revealed if the real secret key is kept hidden, so that this scheme can resist attack from a quantum computer.

References

[1] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, "Deniable encryption," in *Advances in Cryptology — CRYPTO '97*, B. S. Kaliski, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 90–104.

[2] M. Ajtai and C. Dwork, "A public-key cryptosystem with worst-case/average-case equivalence," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 284–293. [Online]. Available: https://doi.org/10.1145/258533.258604

[3] O. Regev, "New lattice based cryptographic constructions," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 407–416. [Online]. Available: https://doi.org/10.1145/780542.780603

[4] ——, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, Sep. 2009. [Online]. Available: https://doi.org/10.1145/1568318.1568324

[5] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM*, vol. 60, no. 6, Nov. 2013. [Online]. Available: https://doi.org/10.1145/2535925

[6] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Des. Codes Cryptography*, vol. 75, no. 3, p. 565–599, Jun. 2015. [Online]. Available: https://doi.org/10.1007/s10623-014-9938-4

[7] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "Crystals - kyber: A cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 353–367.

[8] A. O'Neill, C. Peikert, and B. Waters, "Bi-deniable public-key encryption," in *Proceedings of the 31st Annual Conference on Advances in Cryptology*, ser. CRYPTO'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 525–542.

[9] A. Sahai and B. Waters, "How to use indistinguishability obfuscation: deniable encryption, and more," in *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, ser.

STOC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 475–484. [Online]. Available: https://doi.org/10.1145/2591796.2591825

[10] A. De Caro, V. Iovino, and A. O'Neill, "Deniable functional encryption," in *Public-Key Cryptography – PKC 2016*, C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 196–222.

[11] Z. An, H. Tian, C. Chen, and F. Zhang, "Deniable cryptosystems: Simpler constructions and achieving leakage resilience," in *Computer Security – ESORICS 2023: 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25–29, 2023, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 24–44. [Online]. Available: https://doi.org/10.1007/978-3-031-50594-2$_$2

[12] R. Canetti, S. Park, and O. Poburinnaya, "Fully deniable interactive encryption," in *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 807–835. [Online]. Available: https://doi.org/10.1007/978-3-030-56784-2$_$27

[13] S. Agrawal, S. Goldwasser, and S. Mossel, "Deniable fully homomorphic encryption from learning with errors," in *Advances in Cryptology – CRYPTO 2021*, T. Malkin and C. Peikert, Eds. Cham: Springer International Publishing, 2021, pp. 641–670.

[14] A. Coladangelo, S. Goldwasser, and U. Vazirani, "Deniable encryption in a quantum world," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1378–1391. [Online]. Available: https://doi.org/10.1145/3519935.3520019

[15] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the fujisaki-okamoto transformation," Cryptology ePrint Archive, Paper 2017/604, 2017. [Online]. Available: https://eprint.iacr.org/2017/604

[16] H. Krawczyk and T. Rabin, "Chameleon hashing and signatures," Cryptology ePrint Archive, Paper 1998/010, 1998. [Online]. Available: https://eprint.iacr.org/1998/010

[17] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "PQ-Crystals Kyber on Github." 2018. [Online]. Available: https://github.com/pq-crystals/kyber

[18] julwil, "chameleon hash on Github." 2020. [Online]. Available: https://github.com/julwil/chameleon_hash