Proposal for NIST Post-Quantum Cryptography Standard
(EMBLEM and R.EMBLEM)

Contents

List of Figures

List of Tables

1 BACKGROUND

1.1 Notation

We denote the set of natural numbers by $\mathbb{N}$ and the set of integers by $\mathbb{Z}$. For $q \in \mathbb{N}$, we define the set $\mathbb{Z}_q$ as $\mathbb{Z} \cap (-\frac{q}{2}, \frac{q}{2}]$. For a finite set $\mathcal{S}$, we write $a \leftarrow_R \mathcal{S}$ to describe that a is chosen uniformly at random from $\mathcal{S}$ and for a distribution $\mathcal{X}$, we write $a \leftarrow \mathcal{X}$ to denote that a is sampled according to the distribution $\mathcal{X}$. For a matrix M, M^T denotes the transpose of M. For a vector a of length n, we define a_i to be the i-th component of a, and for a matrix M, we define $M[i, j]$ to be the i-th row and j-th column entry of M. The Euclidean norm $||a||$ is defined as $\sqrt{\Sigma_{i=1}^n a_i^2}$. We define the function $[a]_d$ which drops d least significant bits of a. We expand this function to matrices by applying it to each component of the matrix. For positive integers q and n, we define $\mathcal{U}(\mathbb{Z}_q^n)$ by the uniform distribution over $\mathbb{Z}_q^n$.

1.2 Discrete Gaussian Distribution

For a given $s > 0$, the *discrete* Gaussian distribution over a lattice L is defined as

$$\mathcal{GD}_{L,s}(x) = \frac{\rho_s(x)}{\Sigma_{y \in L} \rho_s(y)} \tag{1}$$

for any $x \in L$, where ρ denote the Gaussian function $\rho_s(x) = e^{-\pi \|x\|^2/s^2}$.

Note that the standard deviation of $\mathcal{GD}_{L,s}$ is $\sigma = s/\sqrt{2\pi}$. The Gaussian parameter s is used to describe a discrete Gaussian distribution throughout this paper. We write $\mathcal{GD}_s$ to denote the discrete Gaussian distribution $\mathcal{GD}_{\mathbb{Z},s}$. Moreover, it holds that $\mathcal{GD}_{\mathbb{Z}^n,s} = \mathcal{GD}_s^n$.

1.3 Learning with Errors

We define a standard lattice-based problem on which the security of our proposal is based.

Definition 1.1 (Decision LWE problem). Let $m, n, k, q \in \mathbb{N}$ and $\mathcal{D}_s, \mathcal{D}_e$ be distributions over $\mathbb{Z}_q$. One is given m samples $(A_i, B_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^k$ and asked to distinguish whether there exists $\mathbf{S} \leftarrow \mathcal{D}_s^{n \times k}$ such that the samples are of the form $(\mathbf{A}, \mathbf{AS} + \mathbf{E} \mod q)$ with $\mathbf{A} \leftarrow_R \mathbb{Z}_q^{m \times n}$, $\mathbf{E} \leftarrow \mathcal{D}_e^{m \times k}$ or the samples are chosen uniformly at random from $\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times k}$. We denote the decision LWE problem by $\mathsf{LWE}_{n,m,q,\mathcal{D}_e}$.

The binary-LWE problem (where secret vector $\mathbf{s}$ is from $\{-1, 0, 1\}^n$) has been considered in work by Micciancio and Peikert [MP13]. In [BG14], Bai and Galbraith proved that binary-LWE problem is as hard as the LWE problem as long as increasing the parameter n by a factor of $\log(\log(n))$. Note that the errors are still discrete Gaussians. We apply Bai and Galbraith's embedding method to the case where $\mathbf{s}$ is sampled from $[-B, B]^n$ for any $B < \sigma$. The matrix version of a (decisional) small secret LWE problem, denoted by $\mathsf{smaLWE}_{n,m,q,\mathcal{D}_e}$, is defined as follows.

Definition 1.2 (Decision LWE Problem with small secrets). Let $m, n, k, q \in \mathbb{N}$ and $\mathcal{D}_e$ be a distribution over $\mathbb{Z}_q$. One is given m samples $(A_i, B_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^k$ and asked to distinguish whether there exists $\mathbf{S} \leftarrow [-B, B]^{n \times k}$ such that the samples are of the form $(\mathbf{A}, \mathbf{AS} + \mathbf{A} \mod q)$ with $\mathbf{A} \leftarrow_R \mathbb{Z}_q^{m \times n}$, $\mathbf{A} \leftarrow \mathcal{D}_e^{m \times k}$ or the samples are chosen uniformly at random from $\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times k}$.

Lemma 1.3 (The Number of Binary LWE samples [BG14]). Let q, n, σ and δ be fixed. Let $m' \approx m + n$ be the dimension of the embedded lattice in the new attack described in Section 6 of [BG14]. For a given Hermite factor δ, the optimal value for m' is approximately

$$\sqrt{\frac{n(\log q - \log \sigma)}{\log \delta}}. \tag{2}$$

Albrecht et al. applied the embedding technique of Bai and Galbraih [BG14] to generalized cases, that is, elements of the secret key are randomly sampled from $[a, b]$. As a result, they proposed the following lemma.

Lemma 1.4 (Log root Hermite factor of LWE instances of small secret [APS15]). Let a small secret LWE instance be characterised by n, α, q, let $\mathbf{s}_{(i)} \leftarrow_R \{a, \ldots, b\}$, let $\xi = 2/(b-a)$ and let $\sigma = \frac{\alpha q}{\sqrt{2\pi}}$. Any lattice reduction algorithm achieving *log root-Hermite factor*:

$$\log \delta = \frac{\big(\log(q/\sigma) - \log(2\tau\sqrt{\pi e})\big)^2 \cdot \log(q/\sigma)}{n\big(2\log(q/\sigma) - \log \xi\big)^2} \tag{3}$$

solves LWE by reducing BDD to uSVP for some fixed $\tau \leq 1$ if we have that $(q^m(\xi\sigma)^n)^{1/(m+n)} \cdot \sqrt{\frac{m+n}{2\pi e}} \leq q$ where $m = m' - n = \sqrt{\frac{n(\log q - \log \sigma)}{\log \delta}} - n$.

Lyubashevsky et al. proposed the ring-LWE problem in [LPR10, LPR13], namely, the LWE problem over rings. We define $a \leftarrow \mathcal{X}^n$ as meaning that n coefficients of a polynomial a are chosen independently from $\mathcal{X}$.

Definition 1.5 (Decision Ring-LWE problem). Let $n, q \in \mathbb{N}$ and $\mathcal{D}_s, \mathcal{D}_e$ be distributions over $\mathbb{Z}_q$. For an irreducible polynomial $f(x) \in \mathbb{Z}[x]$ of degree n, let $R_q = \mathbb{Z}_q[x]/f(x)$ be the ring modulo q. One is given $(a, b) \in R_q^2$ and asked to distinguish whether there exists a polynomial $\mathbf{s} \leftarrow \mathcal{D}_s^n$ such that (a, b) is of the form $(a, a \cdot s + e)$ with $a \leftarrow_R R_q$, $e \leftarrow \mathcal{D}_e^n$ or (a, b) is chosen uniformly at random from R_q^2. We denote the decision Ring-LWE problem by $\mathsf{RLWE}_{n,q,\mathcal{D}_e}$.

The Ring-LWE problem with small secrets (where coefficients of secret polynomial $\mathbf{s} \leftarrow [-B, B]^n$) can be defined similar to smaLWE. In [GKPV10], it is shown that for the standard LWE, the secret $\mathbf{s}$ can be sampled from any distribution, as long as its entropy is sufficiently large. Assuming that the analysis results in the standard LWE setting equally hold in the ring-LWE setting, we can set the coefficients of the secret polynomial to be small in the ring-LWE problem. A (decisional) small secret Ring-LWE problem, denoted by $\mathsf{smaRLWE}_{n,q,\mathcal{D}_e}$, is defined as follows.

Definition 1.6 (Decision Ring-LWE Problem with small secrets). Let $n, q \in \mathbb{N}$ and $\mathcal{D}_e$ be a distribution over $\mathbb{Z}_q$. For an irreducible polynomial $f(x) \in \mathbb{Z}[x]$ of degree n, let $R_q = \mathbb{Z}_q[x]/f(x)$ be the ring modulo q. One is given $(a, b) \in R_q^2$ and asked to distinguish whether there exists a polynomial $\mathbf{s} \leftarrow [-B, B]^n$ such that (a, b) is of the form $(a, a \cdot s + e)$ with $a \leftarrow_R R_q$, $e \leftarrow \mathcal{D}_e^n$ or (a, b) is chosen uniformly at random from R_q^2.

1.4 Definitions

Definition 1.7 (Public Key Encryption scheme). A public-key encryption (PKE) scheme consists of the following three PPT algorithms: KeyGen, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$.

- $\mathsf{KeyGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a public/secret key pair (pk, sk).
- $\mathsf{Encrypt}(pk, m)$: The encryption algorithm takes as input a public key pk and a message $m \in \mathcal{M}$. Then it outputs a corresponding ciphertext C.
- $\mathsf{Decrypt}(sk, C)$: The decryption algorithm takes as input a secret key sk and a ciphertext C. It outputs a message m or $\perp$ (which indicates decryption failure).

Correctness. We guarantee the correctness of a PKE scheme if the following condition holds: For all $m \in \mathcal{M}$,

$$\Pr\left[(pk, sk) \leftarrow_R \mathsf{KeyGen}(1^\lambda);\ C \leftarrow_R \mathsf{Encrypt}(pk, m) : \mathsf{Decrypt}(sk, C) = m\right] > 1 - \epsilon(\lambda)$$

where ϵ is a negligible function.

Definition 1.8 (Key Encapsulation Mechanism). A key encapsulation mechanism (KEM) consists of the following three PPT algorithms: KeyGen, Encap, and Decap.

- $\mathsf{KeyGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a public/secret key pair (pk, sk).
- $\mathsf{Encap}(pk)$: The encapsulation algorithm takes as input a public key pk. Then it outputs a ciphertext C and a key $K \in \mathcal{K}$.
- $\mathsf{Decap}(sk, C)$: The decapsulation algorithm takes as input a secret key sk and a ciphertext C. It outputs a key K.

Correctness. We guarantee the correctness of KEM if the following condition holds:

$$\Pr\left[(pk, sk) \leftarrow_R \mathsf{KeyGen}(1^\lambda);\ (C, K) \leftarrow_R \mathsf{Encap}(pk) : \mathsf{Decap}(sk, C) = K\right] > 1 - \epsilon(\lambda)$$

where ϵ is a negligible function.

Definition 1.9 (IND-CPA Security of PKE). Let $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public-key encryption scheme. Let us define the following experiment (parameterized by a bit b) between an adversary $\mathcal{A}$ and a challenger:

Experiment $\textbf{IND-CPA}^b_{\mathsf{PKE},\mathcal{A}}(k)$:

1. *The challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}(params)$ and gives pk to $\mathcal{A}$;*
2. *$\mathcal{A}$ outputs two message (m_0, m_1) of the same length;*
3. *The challenger computes $\mathsf{Encrypt}(pk, m_b)$ and gives it to $\mathcal{A}$;*
4. *$\mathcal{A}$ outputs a bit b'. The challenger returns b' as the output of the game.*

The advantage of $\mathcal{A}$ for breaking the IND-CPA security of a PKE is defined as

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{PKE},\mathcal{A}} = \left| \Pr\left[\textbf{IND-CPA}^1_{\mathsf{PKE},\mathcal{A}}(k) = 1\right] - \Pr\left[\textbf{IND-CPA}^0_{\mathsf{PKE},\mathcal{A}}(k) = 1\right] \right|.$$

We say that PKE *is IND-CPA secure if for any polynomial time adversary $\mathcal{A}$ and any k, we have $\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{PKE},\mathcal{A}} \leq \epsilon(k)$ where ϵ is a negligible function.*

Definition 1.10 (IND-CCA Security of KEM). Let $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism. Let us define the following experiment (parameterized by a bit b) between an adversary $\mathcal{A}$ and a challenger:

Experiment $\textbf{IND-CCA}^b_{\mathsf{KEM},\mathcal{A}}(k)$:

1. *The challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and gives pk to $\mathcal{A}$;*
2. *$\mathcal{A}$ queries to the decapsulation oracle $\mathsf{Decap}(sk, \cdot)$;*
3. *The challenger computes $(C^*, K_0^*) \leftarrow \mathsf{Encap}(pk)$ and $K_1^* \leftarrow_R \mathcal{K}$. Then the challenger gives (C^*, K_b^*) to $\mathcal{A}$. ;*
4. *$\mathcal{A}$ continues to query the decapsulation oracle, but may not query the ciphertext C^*. Finally, $\mathcal{A}$ outputs a bit b'. The challenger returns b' as the output of the game.*

In the (quantum) random oracle model, the challenger additionally runs the random oracles. The quantum accessible random oracles are described in [TU16]. The advantage of $\mathcal{A}$ for breaking the IND-CCA security of KEM is defined as

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{A}} = \left| \Pr\left[\mathbf{IND\text{-}CCA}^1_{\mathsf{KEM},\mathcal{A}}(k) = 1\right] - \Pr\left[\mathbf{IND\text{-}CCA}^0_{\mathsf{KEM},\mathcal{A}}(k) = 1\right] \right|.$$

We say that KEM *is IND-CCA secure if for any polynomial time adversary $\mathcal{A}$ and any k, we have $\mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{A}} \leq \epsilon(k)$ where ϵ is a negligible function.*

2 ALGORITHM SPECIFICATIONS AND SUPPORTING DOCUMENT

In this section, we introduce EMBLEM and R.EMBLEM as standard candidates. The schemes secure against chosen-plaintext attacks, EMBLEM.CPA and R.EMBLEM.CPA, are presented together for the sake of understanding.

2.1 Algorithm Specifications

2.1.1 EMBLEM.CPA (CPA-Secure Public-Key Encryption)

Encoding and decoding function. Let $\mathcal{M} = \{0,1\}^l$ be the message space. We first define the encoding function `encode`, which takes a bit string as an input and outputs it in a matrix form. The decoding function `decode` is an inverse of `encode`. For inputs an l-bit message m, a block size t, and a modulus q, `encode` operates in the following manner:

▷ `encode`(m, t, q)

1. Split the message by t-bit (we assume that t divides l) and generate l/t message blocks;
2. Transform l/t blocks into a $v \times k$ matrix $M = \{m_{(i,j)}\}$. Denote $m_{(i,j)}$ be the message block assigned to the i-th row and j-th column entry of M;
3. Output a $v \times k$ matrix $M = \{M[i,j] \quad \text{where } M[i,j] \leftarrow m_{(i,j)}||1||\overrightarrow{0}$ for $i \in [1, v]$, $j \in [1, k]$.

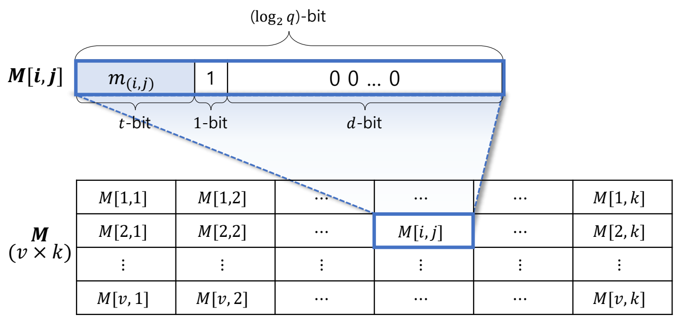


Figure 1: The encoding function `encode`

▷ `decode`(M, t, q)

1. Transform a $v \times k$ matrix into l/t message blocks m_i for $i \in [1, l/t]$;
2. Compute $m'_i = [m_i]_{\log_2(q)-t}$ for $i \in [1, l/t]$;
3. Output l-bit string $m = m'_1|| \cdots ||m'_{l/t}$.

Sampling function. We define the sampling function `Sam` which takes input a random coin r and outputs ephemeral values (R, E_1, E_2) where R is a random string and E_1, E_2 are sampled from Gaussian distribution. The sizes of R, E_1, and E_2 can be extended as desired, and always output the same value for the same input. Since R is a random string, we can generate it directly using the pseudorandom function (PRF) with the random coin r as seed. We can also generate random seeds s_1 and s_2 to generate E_1 and E_2, respectively, as follows: $s_1 \leftarrow \mathrm{PRF}(r||1)$ and $s_2 \leftarrow \mathrm{PRF}(r||2)$. In practice, when implementing Gaussian sampling, a random seed is chosen internally and used to generate a sampling result. We use s_1 and s_2 generated by using PRF, instead of an internally extracted seed, and thus it is reasonable to assume that the distribution of E_1 and E_2 is statistically close to the Gaussian distribution.

When the system is set up, the system parameter *params* is generated as follows: Choose positive integers m, n, k, t, v and the modulus q. Choose a standard deviation $\sigma = s/\sqrt{2\pi}$ for discrete Gaussian distribution $\mathcal{GD}_s$ and a positive integer $B < \sigma$. The parameters are given by $params = (m, n, k, q, t, v, B, \mathcal{GD}_s)$. Note that $l/t = v \times k$. The LWE-based multi-bit encryption scheme EMBLEM.CPA is described as below.

KeyGen(1^λ)**.** Choose a random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{m\times n}$. Choose a secret random matrix $\mathbf{X} \leftarrow [-B, B]^{n\times k}$ and an error matrix $\mathbf{E} \leftarrow \mathcal{GD}_s^{m\times k}$. Compute $\mathbf{B} = \mathbf{AX} + \mathbf{E}$. The key pair (pk, sk) is given by $pk = (\mathbf{A}, \mathbf{B})$ and $sk = (\mathbf{X})$. [1]

Encrypt$(pk, msg \in \mathcal{M})$**.** To generate the ciphertext, proceed with the following steps:

1. $\mathbf{M} \leftarrow \mathtt{encode}(msg, t, q)$;
2. Choose a random coin $r \in \{0,1\}^{256}$;
3. $(\mathbf{R}, \mathbf{E_1}, \mathbf{E_2}) \leftarrow \mathtt{Sam}(r)$ where $\mathbf{R} \in [-B, B]^{m\times v}$ and $(\mathbf{E_1}, \mathbf{E_2}) \in \mathcal{GD}_s^{v\times(n+k)}$;
4. Compute $(\mathbf{C_1}, \mathbf{C_2}) = (\mathbf{R}^T\mathbf{A} + \mathbf{E_1}, \mathbf{R}^T\mathbf{B} + \mathbf{E_2} + \mathbf{M})$;
5. Return the ciphertext $C = (\mathbf{C_1}, \mathbf{C_2}) \in \mathbb{Z}_q^{v\times n} \times \mathbb{Z}_q^{v\times k}$.

Decrypt(sk, C)**.** Parse the ciphertext C as $(\mathbf{C_1}, \mathbf{C_2})$.

1. Compute $\mathbf{M} = \mathbf{C_2} - \mathbf{C_1}\mathbf{X}$;
2. Output $msg \leftarrow \mathtt{Decode}(\mathbf{M}, t, q)$.

Correctness. We show the correctness of the encryption scheme described in Section 2.1.1.

Theorem 2.1 (Correctness). *Let* $\mathbf{E} = \{E^{(i)}\}$ *where* $E^{(i)}$ *is an i-th column of* $\mathbf{E} \in \mathbb{Z}_q^{m\times k}$, $E_2 = \{E_2^{(i)}\}$ *where* $E_2^{(i)}$ *is an i-th entry of column* $E_2 \in \mathbb{Z}_q^k$*, and* $\mathbf{X} = \{X^{(i)}\}$ *where* $X^{(i)}$ *is an i-th column of* $\mathbf{X} \in \mathbb{Z}_q^{n\times k}$*. Let* $\epsilon = \Pr\left[\max_{i\in[1,k]}\left\{|\langle r, E^{(i)}\rangle| + |E_2^{(i)}| + |\langle E_1, X^{(i)}\rangle|\right\} \geq 2^d\right]$ *where* $d = \log_2(q) - (t+1)$*. Then* EMBLEM.CPA *is (1−*ϵ*)-correct.*

Proof. The decryption phase proceeds as follows:

$$\begin{aligned}\mathbf{C_2} - \mathbf{C_1}\mathbf{X} &= (\mathbf{R}^T\mathbf{B} + \mathbf{E_2} + \mathbf{M}) - (\mathbf{R}^T\mathbf{A} + \mathbf{E_1})\mathbf{X} \\ &= \big(\mathbf{R}^T(\mathbf{AX} + \mathbf{E}) + \mathbf{E_2} + \mathbf{M}\big) - (\mathbf{R}^T\mathbf{AX} + \mathbf{E_1}\mathbf{X}) \\ &= (\mathbf{R}^T\mathbf{E} + \mathbf{E_2} - \mathbf{E_1}\mathbf{X}) + \mathbf{M}\end{aligned}$$

[1]Note that the secret matrix $\mathbf{X}$ can be generated using pseudorandom functions, i.e., $\mathbf{X} \leftarrow PRF(seed_{\mathbf{X}})$, since we choose the elements of $\mathbf{X}$ from $[-B, B]$, not from the Gaussian distribution. In other words, the user only needs to store a bit string $seed_{\mathbf{X}}$ instead of the entire matrix of $\mathbf{X}$. Similarly, the matrix $\mathbf{A}$ in the public key can also be derived from the seed by using PRF. In [BCD+16], `AES128-ECB` was used as a PRF with a 256-bit seed.

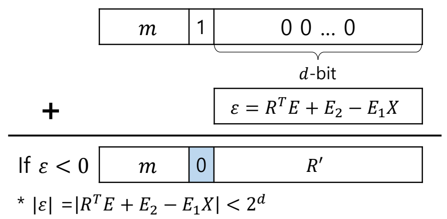

m
1
0 0 … 0
d-bit
+
$\varepsilon = R^T E + E_2 - E_1 X$
If $\varepsilon < 0$
m
0
R'
* $|\varepsilon| = |R^T E + E_2 - E_1 X| < 2^d$

Figure 2: Error propagation in the decryption phase

$\mathbf{M}$ is a $v \times k$ matrix and for $i \in [1, v]$, $j \in [1, k]$, $\mathbf{M}[i, j]$ (i.e., the i-th row and j-th column entry of $\mathbf{M}$) is in the form of $m_{(i,j)}||1||\overrightarrow{0}$ where $\overrightarrow{0}$ is of length d. Therefore, if $|\varepsilon_{(i,j)}|$, the absolute value of (i, j)-th entry in $\mathbf{R^T E} + \mathbf{E_2} - \mathbf{E_1 X}$, is less than 2^d, then it does not affect the message part $m_{(i,j)}$ at all. If $|\varepsilon_{(i,j)}|$ is positive, it is trivial to see that the error is not propagated beyond d least significant bits. Even if $|\varepsilon_{(i,j)}|$ is negative and thus propagated beyond the d-bit, the message part can not be affected by the negative $\varepsilon_{(i,j)}$ because of *error-blocking* bit 1. Fig. 2 illustrates the error propagation in the decryption phase when the error is negative. As a result, when the inequality $|\varepsilon_{(i,j)}| < 2^d$ holds for any $i \in [1, v], j \in [1, k]$, our scheme satisfies correctness. □

In Section 2.1.5, we will set parameters to make the decryption error negligible, i.e., $\epsilon = 2^{-140}$.

Security. We show that the proposed scheme is IND-CPA secure under the hardness assumption of the LWE problem with small secrets.

Theorem 2.2 (Security). *The LWE-based multi-bit encryption* EMBLEM.CPA *is IND-CPA secure if the decision-*smaLWE$_{n,m,q}$ *assumption holds.*

Proof. The proof proceeds by the sequence of games. Note that in Game 0 (1st hybrid game), the public key is the small secret LWE instance and the ciphertext is an encryption of m_0. In Game 5 (6th hybrid game), the public key is the small secret LWE instance and the ciphertext is an encryption of m_1 where $|m_0| = |m_1|$. We show that distributions of Game 0 and Game 5 are computationally indistinguishable for the adversary.

▷ **Game 0.** This is the original game, where the public key and the ciphertext are generated honestly as in Section 2.1.1. In this game, the ciphertext is an encryption of m_0. Note that $M_0 \leftarrow$`encode`(m_0, t, q). The distribution of Game 0, denoted by $\mathcal{D}_0$ is given as follows:

$$\mathcal{D}_0 = \{pk \leftarrow (\mathbf{A}, \mathbf{B} = \mathbf{AX} + \mathbf{E}),\ C \leftarrow (\mathbf{C_1} = \mathbf{R^T A} + \mathbf{E_1}, \mathbf{C_2} = \mathbf{R^T B} + \mathbf{E_2} + \mathbf{M_0})\}$$

▷ **Game 1.** In this game, the public key $\mathbf{B}$ is generated uniformly at random, rather than computed with a secret key $\mathbf{X}$. The rest is the same as in Game 0. The distribution of Game 1, denoted by $\mathcal{D}_1$, is given as follows:

$$\mathcal{D}_1 = \{pk \leftarrow (\mathbf{A}, \boxed{\mathbf{B} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m \times k})}),\ C \leftarrow (\mathbf{C_1} = \mathbf{R^T A} + \mathbf{E_1}, \mathbf{C_2} = \mathbf{R^T B} + \mathbf{E_2} + \mathbf{M_0})\}$$

▷ **Game 2.** In Game 2, the small secret LWE instances contained in the ciphertext change to random matrices. In other words, the ciphertext $C = (\mathbf{C_1}, \mathbf{C_2})= (\mathbf{U_1}, \mathbf{U_2} + \mathbf{M_0})$, where $(\mathbf{U_1}, \mathbf{U_2})$ is generated uniformly at random in $\mathbb{Z}_q^n \times \mathbb{Z}_q^k$.

The rest is the same as in Game 1. In this game, there are no small secret LWE instances. The distribution of Game 2, denoted by $\mathcal{D}_2$, is given as follows:

$$\mathcal{D}_2 = \{pk \leftarrow (\mathbf{A}, \mathbf{B} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m \times k})),\ C \leftarrow (\boxed{\mathbf{C_1} = \mathbf{U_1}, \mathbf{C_2} = \mathbf{U_2} + \mathbf{M_0}})\}$$

▷ **Game 3.** In Game 3, the message m_0 contained in the ciphertext changes to another message m_1. Note that $M_1 \leftarrow$`encode`(m_1, t, q). The rest is the same as in Game 2. The distribution of Game 3, denoted by $\mathcal{D}_3$, is given as follows:

$$\mathcal{D}_3 = \{pk \leftarrow (\mathbf{A}, \mathbf{B} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m\times k})),\ C \leftarrow (\boxed{\mathbf{C_1} = \mathbf{U_1}, \mathbf{C_2} = \mathbf{U_2} + \mathbf{M_1}})\}$$

▷ **Game 4.** In this game, the ciphertext is restored to the small secret LWE instance, and the rest is the same as in Game 3. Note that the ciphertext is an encryption of M_1. The distribution of Game 4, denoted by $\mathcal{D}_4$, is given as follows:

$$\mathcal{D}_4 = \{pk \leftarrow (\mathbf{A}, \mathbf{B} \leftarrow \mathcal{U}(\mathbb{Z}_q^{m\times k})),\ C \leftarrow (\boxed{\mathbf{C_1} = \mathbf{R^T A} + \mathbf{E_1}, \mathbf{C_2} = \mathbf{R^T B} + \mathbf{E_2} + \boxed{\mathbf{M_1}}})\}$$

▷ **Game 5.** In this game, the public key is restored to the small secret LWE instance, and the rest is the same as in Game 4. Game 5 is the same as Game 0, except that the ciphertext is an encryption of M_1. The distribution of Game 5, denoted by $\mathcal{D}_5$, is given as follows:

$$\mathcal{D}_5 = \{pk \leftarrow (\mathbf{A}, \boxed{\mathbf{B} = \mathbf{AX} + \mathbf{E}}),\ C \leftarrow (\mathbf{C_1} = \mathbf{R^T A} + \mathbf{E_1}, \mathbf{C_2} = \mathbf{R^T B} + \mathbf{E_2} + \mathbf{M_1})\}$$

The distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally indistinguishable under the decision-$\mathsf{smaLWE}_{n,m,q}$ assumption. The distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are also computationally indistinguishable under the decision-$\mathsf{smaLWE}_{m,n+k,q}$ assumption, since the ciphertext in Game 1 forms the small secret LWE instances with $(n+k)$ samples in dimension n for a given public key $(\mathbf{A}||\mathbf{B})$. Note that, even if $m > n+k$, one can reduce $\mathsf{smaLWE}_{n,m,q}$ to $\mathsf{smaLWE}_{m,n+k,q}$ by adjusting the size of the matrix during the simulation. In Game 2 and 3, the ciphertexts are computed in a one-time pad manner by adding the message to a random matrix, thus the distributions $\mathcal{D}_2$ and $\mathcal{D}_3$ are statistically indistinguishable. The hybrid game from Game 3 to Game 5 proceeds in the reverse manner from Game 0 to Game 2. If we set the parameters n, m and q of smaLWE to be as hard as LWE, then the security of EMBLEM.CPA can be reduced to the standard LWE problem. □

2.1.2 EMBLEM (CCA-Secure Key Encapsulation Mechanism)

In this section, we propose an IND-CCA secure key encapsulation mechanism (KEM) EMBLEM = (KeyGen, Encap, Decap) in the quantum random oracle model. To construct a CCA secure KEM, we apply the KEM variant of Fujisaki-Okamoto (FO) transformation to our IND-CPA secure encryption scheme EMBLEM.CPA described in Section 2.1.1 [HHK17].

Let $\mathcal{M} = \{0,1\}^{256}$ be the message space of EMBLEM.CPA scheme. The system parameters *params* are given the same as in the EMBLEM.CPA scheme. In the FO transformation, the following three hash functions are used:

- The hash function $G : \{0,1\}^* \to \{0,1\}^{256}$
- The hash function $H : \{0,1\}^* \to \{0,1\}^{256}$
- The hash function $\hat{H} : \{0,1\}^* \to \{0,1\}^{256}$

These hash functions will be modeled as random oracles in the security proof. The CCA-secure KEM EMBLEM is constructed as follows:

KeyGen(1^λ)**.** Same as EMBLEM.CPA.KeyGen.

Encap(pk)**.** To generate the key K and ciphertext C, proceed with the following steps:

1. Select $\delta \leftarrow_R \{0,1\}^{256}$ and compute $r = G(\delta)$;
2. Compute $C_1 \leftarrow$ EMBLEM.CPA.Encrypt$(pk, \delta; r)$ and $C_2 = \hat{H}(\delta)$;
3. Compute $K = H(\delta, C_1, C_2)$;
4. Return the ciphertext $C = (C_1, C_2) \in \mathbb{Z}_q^{v\times(n+k)} \times \{0,1\}^{256}$ and the key $K \in \{0,1\}^{256}$.

Decap(sk, C)**.** Parse the ciphertext C as (C_1, C_2), and proceed with the following steps:

1. Compute $\delta \leftarrow$ EMBLEM.CPA.Decrypt(sk, C_1);

2. Compute $r = G(\delta)$;
3. Compute $e \leftarrow \mathsf{EMBLEM.CPA.Encrypt}(pk, \delta; r)$ and $d = \hat{H}(\delta)$;
   - If $e \neq C_1$ or $d \neq C_2$, output $\perp$;
4. Otherwise, output $K = H(\delta, C_1, C_2)$.

Note that all the ephemeral values selected in EMBLEM.CPA.Encrypt algorithm are determined by the random coin r. The correctness of this scheme is derived from that of the underlying CPA-secure public-key encryption scheme [HHK17].

Theorem 2.3 (Correctness). *If* EMBLEM.CPA *is (1-ϵ)-correct, then* EMBLEM *is (1-ϵ)-correct in the quantum random oracle model.*

EMBLEM is tightly IND-CCA secure in the (classical) random oracle model (by Theorem 2.4), and is non-tightly IND-CCA secure in the quantum random oracle model (by Theorem 2.5). Note that, the hash function $\hat{H}$ is not required to prove IND-CCA security in the (classical) random oracle model.

Theorem 2.4 (Theorem 3.1 and 3.2 in [HHK17]). *Assume EMBLEM.CPA to be δ-correct. For any IND-CCA adversary $\mathcal{B}$ issuing at most q_D decryption queries, at most q_G queries to random oracle G, and at most q_H queries to random oracle H, there exists an IND-CPA adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\text{EMBLEM}}(\mathcal{B}) \leq q_H \cdot \delta + \tfrac{q_H + 2q_G + 1}{2^{256}} + 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{EMBLEM.CPA}}(\mathcal{A})$$

and the running time of $\mathcal{A}$ is about that of $\mathcal{B}$.

Theorem 2.5 (Theorem 4.4 and 4.6 in [HHK17]). *Assume EMBLEM.CPA to be δ-correct. For any IND-CCA quantum adversary $\mathcal{B}$ issuing at most q_D (classical) decryption queries, at most q_G queries to the quantum random oracle G, at most q_H queries to the quantum random oracle H, and at most $q_{\hat{H}}$ queries to the quantum random oracle $\hat{H}$, there exists an IND-CPA quantum adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\text{EMBLEM}}(\mathcal{B}) \leq (2q_{\hat{H}} + q_H) \cdot \sqrt{8 \cdot \delta (q_G + 1)^2 + (1 + 2q_G)\sqrt{\mathsf{Adv}^{\text{IND-CPA}}_{\text{EMBLEM.CPA}}(\mathcal{A})}}$$

and the running time of $\mathcal{A}$ is about that of $\mathcal{B}$.

Note that, by Theorem 4.4 and 4.6 in [HHK17], we can transform a One-Way against Chosen Plaintext Attacks (OW-CPA) secure encryption scheme into an Indistinguishability against Chosen-Ciphertext Attacks (IND-CCA) secure KEM. Since IND-CPA security of encryption scheme with sufficiently large message space implies its OW-CPA security (Lemma 2.3 in [HHK17]), we can say that Theorem 2.4 and 2.5 imply the conversion from IND-CPA secure encryption scheme to IND-CCA secure KEM.

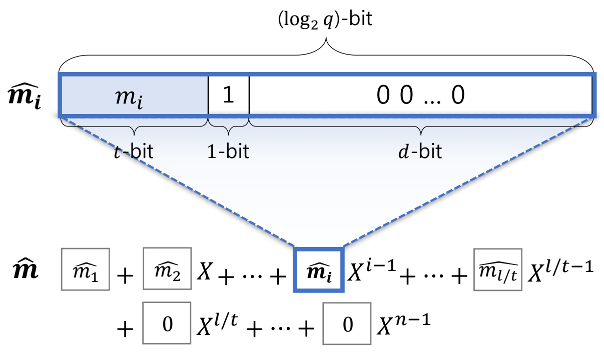
2.1.3 R.EMBLEM.CPA (CPA-Secure Public-Key Encryption over Rings)

In this section, we provide a CPA-secure public-key encryption (PKE) scheme over rings.

Encoding and decoding function over rings. Let $\mathcal{M} = \{0, 1\}^l$ be the message space. We define the encoding function `R.encode`, which takes a bit string as an input and outputs it in a polynomial form, and its inverse function `R.decode`. It operates similarly to the `encode` function in Section 2.1.1, except that the output is in polynomial form.

▷ `R.encode`(m, t, q)

1. Split l-bit message m by t-bit and generate l/t message blocks $\{m_i\}$;
2. Generate $\hat{m_i} = m_i||1||\overrightarrow{0}$, of length $\log_2(q)$, for $i \in [1, l/t]$;
3. Output a polynomial $\hat{m}$ where the coefficient vector is set to $(\hat{m_1}, \dots, \hat{m_{l/t}}, 0, \dots, 0)$.



(log₂ q)-bit
m̂ᵢ
mᵢ
1
0 0 … 0
t-bit
1-bit
d-bit
m̂
m̂₁ + m̂₂ X + ⋯ + m̂ᵢ X^{i−1} + ⋯ + m̂_{l/t} X^{l/t−1}
+ 0 X^{l/t} + ⋯ + 0 X^{n−1}

Figure 3: The encoding function `R.encode`

▷ `R.decode`$(\hat{m}, t, q)$

1. Parse the first l/t coefficients of $\hat{m}$ as $\hat{m}_i$ for $i \in [1, l/t]$;
2. Compute $m_i = [\hat{m}_i]_{\log_2(q)-t}$ for $i \in [1, l/t]$;
3. Output l-bit string $m = m_1|| \cdots ||m_{l/t}$.

For the cyclotomic polynomial $f(x) = x^n + 1 \in \mathbb{Z}[x]$, let $R= \mathbb{Z}[x]/\langle f(x)\rangle$ be the ring of integer polynomials modulo $f(x)$, and $R_q= \mathbb{Z}_q[x]/\langle f(x)\rangle$ be the ring of integer polynomials modulo both $f(x)$ and q. Elements of R_q can be represented by polynomials of degree less than n whose coefficients are from the set $\{0, \dots, q-1\}$.

Truncate function. To increase the efficiency of transmission, we use a function Trunc that truncates the ciphertext [AOP+17]. Let $a \in R_q$ be a polynomial of degree n, represented by $a = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$. For $1 \le l \le n$, we define Trunc as follows:

$$\mathsf{Trunc}(a, l) = a_0 + a_1X + \cdots + a_{l-1}X^{l-1}.$$

When the system is set up, the system parameter *params* is generated as follows: Choose positive integers n, t and a modulus q. Choose a standard deviation $\sigma= s/\sqrt{2\pi}$ for discrete Gaussian distribution $\mathcal{GD}_s$ and a positive integer $B < \sigma$. Let R_q denote the reduction of a ring R modulo q, i.e., $R_q= \mathbb{Z}_q[X]/\langle f(x)\rangle$. The parameters are given by $params = (n,\ t,\ q,\ B,\ R_q,\ \mathcal{GD}_s)$. Our ring-based PKE, R.EMBLEM = (KeyGen, Encrypt, Decrypt), is described as below.

KeyGen(1^λ)**.** Choose a polynomial $a \leftarrow R_q$. Choose a secret polynomial $x \in R_q$ where the coefficient vector is sampled randomly from $[-B, B]^n$. [2] Choose an error polynomial $e \in R_q$ where the coefficient vector is sampled randomly from $\mathcal{GD}_s^n$. Compute $b = a \cdot x + e \in R_q$. A key pair (pk, sk) are given by $pk = (a, b) \in R_q^2$ and $sk = (x) \in R_q$.

Encrypt$(pk, m \in \mathcal{M})$**.** Assume that $l = 256 < n$. To generate the ciphertext, proceed with the following steps:

[2]As in EMBLEM.CPA scheme in Section 2.1.1, the coefficients of the secret polynomial x can be generated using PRF, and thus it is required to store only the seed, the input of the PRF, rather than the entire polynomial (i.e., all the coefficients).

1. $\hat{m} \leftarrow \texttt{R.encode}(m, t, q)$;
2. Choose a random coin $z \in \{0,1\}^{256}$;
3. $(r, e_1, e_2) \leftarrow \texttt{Sam}(z)$ where $r \in [-B, B]^n$ and $(e_1, e_2) \in \mathcal{GD}_s^{n+n}$;
4. Compute $c_1 = r \cdot a + e_1$ and $c_2 \leftarrow \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m}$;
5. Return the ciphertext $c = (c_1, c_2) \in R_q^2$.

Decrypt(sk, c)**.** Parse the ciphertext c as $(c_1, c_2) \in R_q^2$.

1. Compute $d \leftarrow \mathsf{Trunc}(c_1 \cdot x,\ l/t)$ and $\hat{m} = c_2 - d$;
2. Output $m \leftarrow \texttt{R.decode}(\hat{m}, t, q)$.

Theorem 2.6 (Correctness). *Let $r^{(i)}$, $e^{(i)}$, $e_1^{(i)}$, $e_2^{(i)}$, and $x^{(i)}$ be the i-th coefficients of the polynomials r, e, e_1, e_2, and x, respectively. Let $\epsilon = \Pr\left[\max_{i \in [1, l/t]} \{|r^{(i)} \cdot e^{(i)}| + |e_2^{(i)}| + |e_1^{(i)} \cdot x^{(i)}| \ \geq 2^d\right]$ where $d = \log_2(q) - (t+1)$. Then* R.EMBLEM.CPA *is $(1-\epsilon)$-correct.*

Proof. Parse (c_1, c_2) of the ciphertext c as $c_1 = r \cdot a + e_1$, $c_2 \leftarrow \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m}$. In the decryption phase,

$$\begin{aligned} c_2 - d &= \mathsf{Trunc}\big(r \cdot (a \cdot x + e) + e_2,\ l/t\big) + \hat{m} - \mathsf{Trunc}\big((r \cdot a + e_1) \cdot x,\ l/t\big) \\ &= \mathsf{Trunc}\big(r \cdot e + e_2 - e_1 \cdot x,\ l/t\big) + \hat{m} \end{aligned}$$

where polynomials c_2 and d are of degree l/t.

Each coefficient of the polynomial $\hat{m}$, denoted by $\hat{m_i} \in \mathbb{Z}_q$, is encoded as $m_i||1||\overrightarrow{0}$ where $\overrightarrow{0}$ is a vector of length d. If $|\hat{e}_i| < 2^d$, when adding $\hat{e}_i$ to $\hat{m}_i$, $\hat{e}_i$ does not affect m_i. Note that m_i is the most significant t bits of $\hat{m}_i$ where $t = \log_2(q) - (d+1)$. Even if $\hat{e}_i$ is negative, the error does not affect m_i by virtue of the error-blocking bit 1, as in EMBLEM.CPA. In other words, $\hat{e_i} + \hat{m_i} \in \mathbb{Z}_q$ is in the form of $m_i||R$ where R is the bit string of length $\log_2(q) - t$. As a result, R.EMBLEM.CPA successfully decrypts with the probability of $1 - \epsilon$. □

Theorem 2.7 (Security). R.EMBLEM.CPA *is IND-CPA secure if the decision-*$\mathsf{smaRLWE}_{n,q}$ *assumption holds.*

Proof. The proof proceeds with a sequence of games, as in Theorem 2.2. In Game 0 (the first hybrid game), the public key is a small secret ring-LWE instance and the ciphertext is an encryption of m_0. In Game 7 (the last hybrid game), the public key is still a small secret ring-LWE instance, and only the ciphertext changes to the encryption of m_1 such that $|m_0| = |m_1|$. Let denote $\mathcal{D}_i$ as the distribution of Game i. We show that $\mathcal{D}_0$ and $\mathcal{D}_7$ are computationally indistinguishable for the adversary.

▷ **Game 0.** In this game, the public key and the ciphertext are generated honestly as in Section 2.1.3 where the ciphertext is an encryption of m_0. Note that $\hat{m_0} \leftarrow \texttt{R.encode}(m_0, t, q)$. $\mathcal{D}_0$ is given as follows:

$$\mathcal{D}_0 = \{pk \leftarrow (a, b = a \cdot x + e),\ C \leftarrow (c_1 = r \cdot a + e_1, c_2 = \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m_0})\}$$

▷ **Game 1.** In this game, the public key b is generated uniformly at random, rather than computed with a secret key x. The rest is the same as in Game 0. $\mathcal{D}_1$ is given as follows:

$$\mathcal{D}_1 = \{pk \leftarrow (a, \boxed{b \leftarrow \mathcal{U}(R_q)}),\ C \leftarrow (c_1 = r \cdot a + e_1, c_2 = \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m_0})\}$$

▷ **Game 2.** In Game 2, the small secret ring-LWE instance c_1 contained in the ciphertext change to a random polynomial. In other words, the ciphertext $c = (c_1, c_2) = (u_1, \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m_0})$, where u_1 is generated uniformly at random in R_q. The rest is the same as in Game 1. $\mathcal{D}_2$ is given as follows:

$$\mathcal{D}_2 = \{pk \leftarrow (a, b \leftarrow \mathcal{U}(R_q)),\ C \leftarrow (\boxed{c_1 = u_1}, c_2 = \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m_0})\}$$

▷ **Game 3.** In Game 3, the small secret ring-LWE instance c_2 contained in the ciphertext change to a random polynomial. In other words, the ciphertext $c = (c_1, c_2) = (u_1, \mathsf{Trunc}(u_2,\ l/t) + \hat{m_0})$, where u_2 is generated uniformly at random in R_q. The rest is the same as in Game 2. In this game, there are no small secret ring-LWE instances. $\mathcal{D}_3$ is given as follows:

$$\mathcal{D}_3 = \{pk \leftarrow (a, b \leftarrow \mathcal{U}(R_q)),\ C \leftarrow (c_1 = u_1, \boxed{c_2 = \mathsf{Trunc}(u_2,\ l/t)} + \hat{m_0})\}$$

▷ **Game 4.** In Game 4, the message m_0 contained in the ciphertext changes to m_1. Note that $\hat{m_1} \leftarrow \mathtt{R.encode}(m_1, t, q)$. The rest is the same as in Game 2. $\mathcal{D}_4$ is given as follows:

$$\mathcal{D}_4 = \{pk \leftarrow (a, b \leftarrow \mathcal{U}(R_q)),\ C \leftarrow (c_1 = u_1, \boxed{c_2 = \mathsf{Trunc}(u_2,\ l/t) + \boxed{\hat{m_1}}})\}$$

▷ **Game 5.** In Game 5, c_2 in ciphertext is restored to the small secret ring-LWE instance, and the rest is the same as in Game 4. $\mathcal{D}_5$ is given as follows:

$$\mathcal{D}_5 = \{pk \leftarrow (a, b \leftarrow \mathcal{U}(R_q)),\ C \leftarrow (c_1 = u_1, \boxed{c_2 = \mathsf{Trunc}(r \cdot b + e_2,\ l/t)} + \hat{m_1})\}$$

▷ **Game 6.** In this game, c_1 in ciphertext is restored to the small secret ring-LWE instance, and the rest is the same as in Game 5. Note that the ciphertext is an encryption of m_1. $\mathcal{D}_6$ is given as follows:

$$\mathcal{D}_6 = \{pk \leftarrow (a, b \leftarrow \mathcal{U}(R_q)),\ C \leftarrow (\boxed{c_1 = r \cdot a + e_1}, c_2 = \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m_1})\}$$

▷ **Game 7.** In this game, the public key is restored to the small secret ring-LWE instance, and the rest is the same as in Game 6. Game 7 is the same as Game 0, except that the ciphertext is an encryption of m_1. $\mathcal{D}_7$ is given as follows:

$$\mathcal{D}_7 = \{pk \leftarrow (a, \boxed{b = a \cdot x + e}),\ C \leftarrow (c_1 = r \cdot a + e_1, c_2 = \mathsf{Trunc}(r \cdot b + e_2,\ l/t) + \hat{m_1})\}$$

$\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally indistinguishable under the decision-$\mathsf{smaRLWE}_{n,q}$ assumption. $\mathcal{D}_1$ and $\mathcal{D}_2$ are also computationally indistinguishable under the decision-$\mathsf{smaRLWE}_{n,q}$ assumption. $\mathcal{D}_2$ and $\mathcal{D}_3$ are computationally indistinguishable under the decision-$\mathsf{smaRLWE}_{l/t,q}$ assumption. Since $l/t \leq n$, $\mathsf{smaRLWE}_{l/t,q}$ can be reduced to $\mathsf{smaRLWE}_{n,q}$. In Game 3 and 4, the ciphertexts are computed in a one-time pad manner by adding the message to a random polynomial, thus $\mathcal{D}_3$ and $\mathcal{D}_4$ are statistically indistinguishable. The hybrid games from Game 4 to Game 7 proceeds in the reverse manner from Game 0 to Game 3. If we set the parameters n and q of smaRLWE to be as hard as RLWE, then the security of R.EMBLEM.CPA can be reduced to the standard Ring LWE problem. □

2.1.4 R.EMBLEM (CCA-Secure Key Encapsulation Mechanism over Rings)

In this section, we propose a CCA-secure key encapsulation mechanism (KEM) R.EMBLEM in the quantum random oracle model, by applying the KEM variant of Fujisaki-Okamoto (FO) transformation [HHK17] to R.EMBLEM.CPA. Let $\mathcal{M} = \{0,1\}^{256}$ be the message space and the system parameter *params* are given the same as in R.EMBLEM.CPA, except that the hash functions $G, H, \hat{H}$ described in Section 2.1.2 are additionally included. Our ring-based KEM R.EMBLEM = (KeyGen, Encap, Decap) is described as below.

KeyGen(1^λ)**.** Same as R.EMBLEM.CPA.

Encap(pk)**.** Assume that $n > 256$. To generate the key K and the ciphertext c, proceed with the following steps:

1. Select $\delta \leftarrow_R \{0,1\}^{256}$ and compute $z = G(\delta)$;
2. Compute $(c_1, c_2) \leftarrow$R.EMBLEM.CPA.Encrypt$(pk, \delta; z)$, $c_3 = \hat{H}(\delta)$;
3. Return the ciphertext $c = (c_1, c_2, c_3) \in R_q^2 \times \{0,1\}^{256}$ and the key $K = H(\delta, c) \in \{0,1\}^{256}$.

Decap(sk, c). Parse the ciphertext c as $(c_1, c_2, c_3) \in R_q^2 \times \{0,1\}^{256}$, and proceed with the following steps:

1. Compute $\delta \leftarrow$R.EMBLEM.CPA.Decrypt$(sk, (c_1, c_2))$;
2. Compute $z = G(\delta)$;
3. Compute $(d_1, d_2) \leftarrow$R.EMBLEM.CPA.Encrypt$(pk, \delta; z)$ and $d_3 = \hat{H}(\delta)$;
   - If $(d_1, d_2) \neq (c_1, c_2)$ or $d_3 \neq c_3$, output $\perp$;
4. Otherwise, output $K = H(\delta, c)$.

As in EMBLEM, R.EMBLEM.CPA.Encrypt algorithm outputs the same ephemeral values according to the same random coin z. The correctness of R.EMBLEM is derived from R.EMBLEM.CPA [HHK17].

Theorem 2.8 (Correctness). *If* R.EMBLEM.CPA *is* $(1-\epsilon)$*-correct, then* R.EMBLEM *is* $(1-\epsilon)$*-correct in the quantum random oracle model.*

R.EMBLEM is constructed by applying the KEM variant of Fujisaki-Okamoto transformation [HHK17]. Therefore, in common with EMBLEM, R.EMBLEM is IND-CCA secure in both the classical random oracle model (by Theorem 2.9) and the quantum random oracle model (by Theorem 2.10).

Theorem 2.9 (Theorem 3.1 and 3.2 in [HHK17]). *Assume R.EMBLEM.CPA to be δ-correct. For any IND-CCA adversary $\mathcal{B}$ issuing at most q_D decryption queries, at most q_G queries to random oracle G, and at most q_H queries to random oracle H, there exists an IND-CPA adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\text{R.EMBLEM}}(\mathcal{B}) \leq q_H \cdot \delta + \tfrac{q_H + 2q_G + 1}{2^{256}} + 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{R.EMBLEM.CPA}}(\mathcal{A})$$

and the running time of $\mathcal{A}$ is about that of $\mathcal{B}$.

Theorem 2.10 (Theorem 4.4 and 4.6 in [HHK17]). *Assume R.EMBLEM.CPA to be δ-correct. For any IND-CCA quantum adversary $\mathcal{B}$ issuing at most q_D (classical) decryption queries, at most q_G queries to the quantum random oracle G, at most q_H queries to the quantum random oracle H, and at most $q_{\hat{H}}$ queries to the quantum random oracle $\hat{H}$, there exists an IND-CPA quantum adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\text{R.EMBLEM}}(\mathcal{B}) \leq (2q_{\hat{H}} + q_H) \cdot \sqrt{8 \cdot \delta(q_G+1)^2 + (1+2q_G)\sqrt{\mathsf{Adv}^{\text{IND-CPA}}_{\text{R.EMBLEM.CPA}}(\mathcal{A})}}$$

and the running time of $\mathcal{A}$ is about that of $\mathcal{B}$.

As mentioned in Section 2.1.2, since IND-CPA security with sufficiently large message space implies its One-Way against Chosen Plaintext Attacks (OW-CPA) security, by Lemma 2.3 in [HHK17], we can insist that if R.EMBLEM.CPA is an IND-CPA secure PKE, then R.EMBLEM is an IND-CCA secure KEM by Theorem 2.9 and 2.10. □

2.1.5 Parameter Selection

• **Secret distribution $\mathcal{D}_s$.** The distribution $\mathcal{D}_s$ is to sample a value from $[-B, B]$ for $B < \sigma$ uniformly at random. If $S \leftarrow \mathcal{D}_s^{n \times k}$, then $(n \times k)$ entries of the matrix S are chosen uniformly at random from $[-B, B]^{n \times k}$. Using a small B, the size of the error generated in the decryption phase can be reduced. When decrypting, the inner product of two vectors, an error vector sampled from the Gaussian distribution and the secret vector, is computed. Since the secret key is made up of small values chosen from $[-B, B]$, it gives a much smaller result than the inner product of Gaussian vectors. In addition, since the secret key is sampled from $[-B, B]$ uniformly at random, it can be generated using a pseudorandom function (PRF). Therefore, we only need to store the *seed*, used as input to the PRF, as a secret key instead of the entire matrix, which causes the secret key size to decrease remarkably.

• **Error distribution $\mathcal{D}_\mathbf{e}$.** The distribution $\mathcal{D}_e$ is the discrete Gaussian distribution with the Gaussian parameter s, i.e., $\mathcal{GD}_s$. If $E \leftarrow \mathcal{D}_e^{m\times k}$, then $(m \times k)$ entries of the matrix E are chosen from $\mathcal{GD}_s^{m\times k}$. The standard deviation of $\mathcal{GD}_s$ is set to $\sigma = s/\sqrt{2\pi}$. In Section 2.1.5, we set σ to be $s > 2\sqrt{n}$. For efficiency reasons, most LWE-based public-key encryption schemes have used Gaussian parameters much smaller than $2\sqrt{n}$, which fails to achieve worst-case to average-case reduction [Reg09]. We choose $\sigma > \sqrt{2n/\pi}$ so that our constructions have worst-case to average-case security reduction.

• **Message Space.** In our scheme, we set the message space to $\{0,1\}^{256}$, i.e., $l = 256$. To encrypt a 256-bit plaintext, the parameter should be set to satisfy the following:

$$k \times t \times v \geq 256 \text{ for integers } k, t, v \geq 1.$$

For simplicity, we assume that $k \times t \times v = 256$. k is associated with the size of the public key and ciphertext, and v is associated with the size of the ciphertext. t implies the number of bits encrypted per entry of the matrix of ciphertext, which in turn affects the size of q. If one intends to reduce the size of the ciphertext, v should be set to be small, and if one intends to reduce the size of the public key, k and t should be set to be small.

• **Probability of Decryption Failure.** For simplicity, let r be a $m \times 1$ vector (i.e., $v = 1$). Let $\mathbf{E} = \{E^{(i)}\}$ where $E^{(i)}$ is an i-th column of $\mathbf{E} \in \mathbb{Z}_q^{m\times k}$, $E_2 = \{E_2^{(i)}\}$ where $E_2^{(i)}$ is an i-th entry of column $E_2 \in \mathbb{Z}_q^k$, and $\mathbf{X} = \{X^{(i)}\}$ where $X^{(i)}$ is an i-th column of $\mathbf{X} \in \mathbb{Z}_q^{n\times k}$. To ensure the ***correctness*** of $\mathsf{EMBLEM.CPA}$ and EMBLEM, we must set d to satisfy the following equations:

$$\Pr\left[\max_{i\in[1,k]}\left\{|\langle r, E^{(i)}\rangle| + |E_2^{(i)}| + |\langle E_1, X^{(i)}\rangle|\right\} < 2^d\right] = 1 - \epsilon. \tag{4}$$

Once d is determined, for a given t, we can determine the size of q such that $\log_2(q) = t + (1+d)$. We will set d to make the decryption error negligible, namely, $\epsilon = 2^{-140}$. To obtain d satisfying the equation (4), we use the following Lemmas:

Lemma 2.11 (Lemma 2.4 of [Ban95]). *For any real $s > 0$ and $Q > 0$, and any $x \in \mathbb{R}^n$, we have*

$$\Pr\left[\ |\langle x, \mathcal{GD}_{\mathbb{Z}^n,s}\rangle| \geq Q \cdot s||x||\ \right] < 2e^{-\pi\cdot Q^2}. \tag{5}$$

where the standard deviation σ of $\mathcal{GD}_{\mathbb{Z}^n,s}$ is set to $\sigma = s/\sqrt{2\pi}$.

Lemma 2.12 (Lemma 3.3 of [Lyu16]). *For any $r > 0$ and $z \leftarrow \mathcal{GD}_s$ where the standard deviation $\sigma = s/\sqrt{2\pi}$, we have*

$$\Pr\left[|z| > T\sigma\right] < 2e^{-T^2/2}. \tag{6}$$

$|\langle r, E^{(i)}\rangle|$ and $|\langle E_1, X^{(i)}\rangle|$ have approximate values of $Q \cdot \sigma\sqrt{2\pi} \cdot ||r||$ and $Q \cdot \sigma\sqrt{2\pi} \cdot ||X^{(i)}||$, respectively, with the probability of $2e^{-\pi\cdot Q^2}$. Q is set to about 5.5776 so that the probability that $|\langle x, \mathcal{GD}_s\rangle| \geq Q \cdot s||x||$ is at most 2^{-140} (i.e., $2e^{-\pi\cdot 5.5776^2} \approx 2^{-140}$). In addition, $|E_2^{(i)}|$ has approximate value of $T\sigma$ with the probability of $2e^{-T^2/2}$. T is set to about 13.98 so that the probability that $|E_2^{(i)}| > T\sigma$ is at most 2^{-140} (i.e., $2e^{-13.98^2/2} \approx 2^{-140}$). Therefore, for $i \in [1, k]$, $\Pr\left[|\langle r, E^{(i)}\rangle| + |E_2^{(i)}| + |\langle E_1, X^{(i)}\rangle| < Q \cdot \sigma\sqrt{2\pi} \cdot ||r|| + T\sigma + Q \cdot \sigma\sqrt{2\pi} \cdot ||X^{(i)}||\right] = 1 - 2^{-140}$. So we should find d such that $Q \cdot \sigma\sqrt{2\pi} \cdot ||r|| + T\sigma + Q \cdot \sigma\sqrt{2\pi} \cdot ||X^{(i)}|| < 2^d$ where $Q = 5.5776$ and $T = 13.98$. The selected parameter set is given in Table 1.

<u>Case 1.</u> If r and $X^{(i)}$ are sampled from $\{-1,0,1\}^m$ and $\{-1,0,1\}^n$ uniformly at random (i.e., $B=1$), the expected values of $||r||$ and $||X^{(i)}||$ are $\sqrt{\frac{2}{3}m}$ and $\sqrt{\frac{2}{3}n}$, respectively. In case of **I** in Table 1, $(m,n,\sigma)=(1003,770,25)$, and thus we can calculate as follows:

- $\triangleright$ $Q\cdot\sigma\sqrt{2\pi}\cdot||r||=5.5776\times 25\sqrt{2\pi}\times\sqrt{\frac{2}{3}\cdot 1003}\approx 2^{13.14}$
- $\triangleright$ $T\sigma=13.98\times 25\approx 2^{8.45}$
- $\triangleright$ $Q\cdot\sigma\sqrt{2\pi}\cdot||X^{(i)}||=5.5776\times 25\sqrt{2\pi}\times\sqrt{\frac{2}{3}\cdot 770}\approx 2^{12.95}$

Following this, $|\langle r,E^{(i)}\rangle|+|E_2^{(i)}|+|\langle E_1,X^{(i)}\rangle|<2^{13.14}+2^{8.45}+2^{12.95}\approx 2^{14.08}<2^d$. Therefore, $d=15$ is sufficient in this setting.

<u>Case 2.</u> If r and $X^{(i)}$ are sampled from $\{-2,-1,0,1,2\}^m$ and $\{-2,-1,0,1,2\}^n$ uniformly at random (i.e., $B=2$), the expected values of $||r||$ and $||X^{(i)}||$ are $\sqrt{2m}$ and $\sqrt{2n}$, respectively. In case of **II** in Table 1, $(m,n,\sigma)=(832,611,25)$, and thus we can calculate as follows:

- $\triangleright$ $Q\cdot\sigma\sqrt{2\pi}\cdot||r||=5.5776\times 25\sqrt{2\pi}\times\sqrt{2\cdot 832}\approx 2^{13.8}$
- $\triangleright$ $T\sigma=13.98\times 25\approx 2^{8.45}$
- $\triangleright$ $Q\cdot\sigma\sqrt{2\pi}\cdot||X^{(i)}||=5.5776\times 25\sqrt{2\pi}\times\sqrt{2\cdot 611}\approx 2^{13.58}$

Following this, $|\langle r,E^{(i)}\rangle|+|E_2^{(i)}|+|\langle E_1,X^{(i)}\rangle|<2^{13.8}+2^{8.45}+2^{13.58}\approx 2^{14.71}<2^d$. Therefore, $d=15$ is sufficient in this setting.

The same analysis of the probability of decryption failure can be applied to R.EMBLEM.CPA and R.EMBLEM as well.

• **Proposed parameter sets.** The parameters of EMBLEM in Section 2.1.2, aiming at 128-bit security, are given in Table 1. The columns **I** and **II** correspond to cases where each element of the secret key is sampled from [-1,1] and [-2,2], respectively. In each case, n and the root Hermite factor δ are derived from the equation (3), and then, m is calculated using the equation (2). Note that d, the maximum size of the error, should be selected to satisfy the equation (4). In addition, we set the standard deviation σ of the Gaussian distribution to be larger than $\sqrt{2n/\pi}$, which is much higher than other LWE-based schemes [GPV08, LP11, CKLS16, BDK+17], to support worst-case to average-case reduction of the underlying LWE problem.

	I	**II**
	[-1,1]	[-2,2]
m	1003	832
n	770	611
$\log_2(q)$	24	24
σ	25	25
t	8	8
δ	1.003292	1.003945

Table 1: Parameter sets for 128-bit security

In the LWE instance with small secret, the secret key can be sampled from $[-B,B]$ for any $B<\sigma$. We consider only $B=1$ and $B=2$ (column **I** and **II** in Table 1, respectively), because there is no large difference in parameter size when $B\geq 3$. Figure 4 illustrates the tradeoff between security and performance in case of $B=1$ (L.H.S) and $B=2$ (R.H.S). In this figure, uSVP implies the security level against the primal attack via standard embedding, dual embedding, and Bai-Galbraith embedding [AFG13,BG14]. The parameter n represents the dimension and m represents the number of samples. The larger the root Hermit factor δ is, the smaller the size of the parameter

n and m is, and thus the lower the security level against the primal attack via uSVP is. Note that we fix the other parameters $\log_2(q) = 24$, $\sigma = 25$, and $t = 8$. The performance of the algorithm is improved by reducing the size of parameters m and n. As a result, the security and performance of the algorithm are inversely proportional.

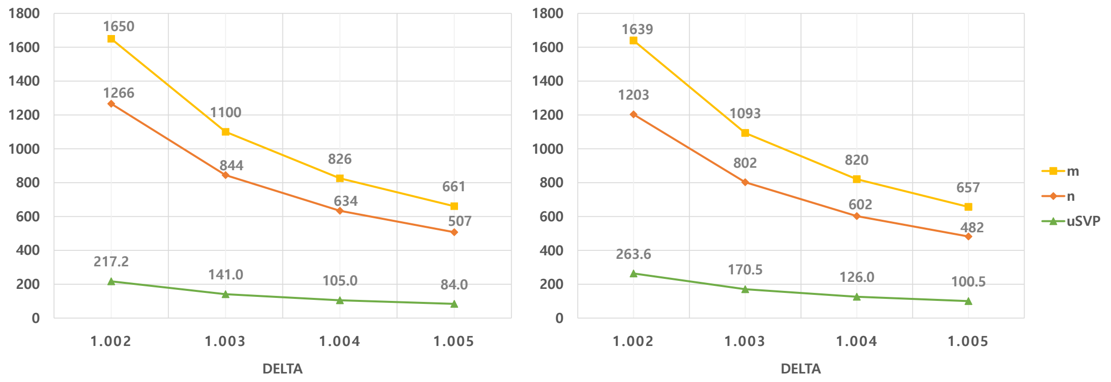


Figure 4: Security/Performance tradeoffs of LWE instances

• **Proposed parameter sets over rings.** The parameters of R.EMBLEM in Section 2.1.4, aiming at 128-bit security, are given in Table 2. The columns (**i,iii**) and (**ii,iv**) correspond to cases where each element of the secret key is sampled from [-1,1] and [-2,2], respectively. In columns **i** and **ii**, the standard deviation σ is set to 25, which is larger than $\sqrt{2n/\pi}$, whereas, in columns **iii** and **iv**, σ is set to 3, which is much smaller than $\sqrt{2n/\pi}$. Since σ is small, the size of the error is small, and consequently the size of q is small. To provide the same 128-bit security despite σ being small, the size of n should be large. As shown in Table 2, n in column **iii** is larger than in column **i**, and n in column **iv** is also larger than in column **ii**.

	i	**ii**	**iii**	**iv**
	[-1,1]	[-2,2]	[-1,1]	[-2,2]
n	463	320	504	437
$\log_2(q)$	16	16	14	14
σ	25	25	3	3
t	1	1	1	1
δ	1.00256	1.00349	1.0027	1.002878

Table 2: Parameter sets for 128-bit security (over Rings)

Figure 5 illustrates the tradeoff between security and performance in case of $\sigma = 25$ (L.H.S) and $\sigma = 3$ (R.H.S). That is, L.H.S of Figure 5 represents the columns **i** and **ii** in Table 2, and R.H.S represents the columns **iii** and **iv**. In this figure, uSVP implies the security level against the primal attack [AFG13,BG14], and uSVP[B=1] and uSVP[B=2] represent the security levels against the primal attack in case of $B = 1$ and $B = 2$, respectively. As in Figure 4, the larger the root Hermit factor δ is, the smaller the size of the parameter n is, and thus the lower the security level against the primal attack via uSVP is. As the size of n decreases, the computational complexity is also reduced, thus improving the performance of the algorithm. For the same σ, n in case of $B = 1$, denoted by n[B=1], should be larger than in case of $B = 2$, denoted by n[B=2], to provide the same security level. And even for the same δ in either L.H.S or R.H.S, n[B=1] is larger than n[B=2], but the security level of n[B=1] is lower than that of n[B=2]. Finally, the smaller σ is, the larger n is to provide the same security level.

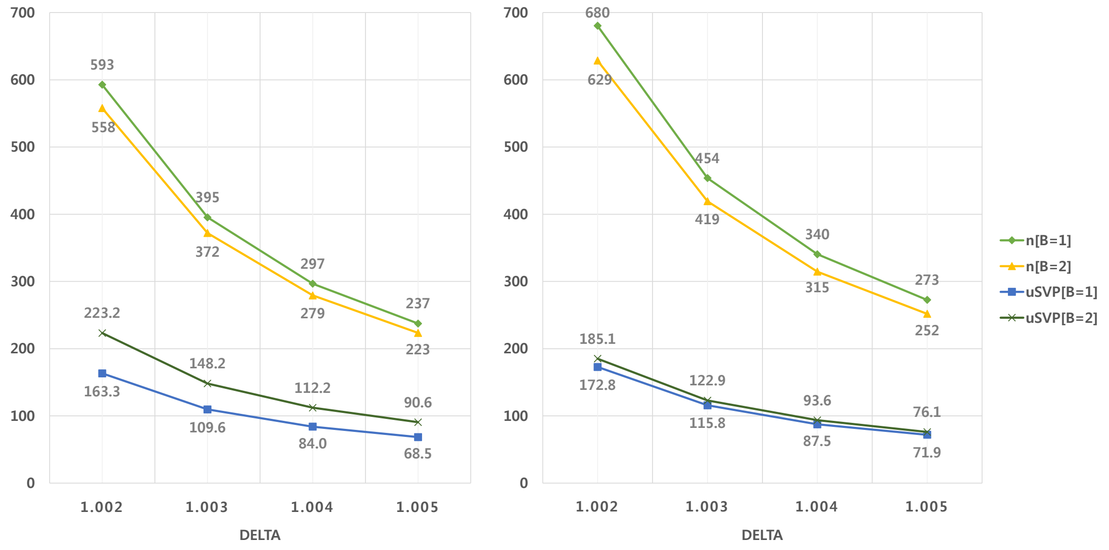


Figure 5: Security/Performance tradeoffs of Ring LWE instances

2.2 Performance Analysis

• **A description of the platform.** Our software implementation was performed on an Intel core i7-6700 (Skylake) running at 3.40GHz, and ran on Linux OS. For hardware implementation, we used a Zynq 7 FPGA platform and Xilinx EDA tool with default option.

• **Speed estimate and memory requirements.** We set the message length to 256-bit. Table 3 and Table 4 represent various parameter sets based on the parameters in column **I** and column **II** of Table 1, respectively. Table 5 presents the number of milliseconds and the size of inputs and outputs (e.g., public key, secret key, and ciphertext) of each algorithm based on the parameters in Table 2.

In Table 3, we have described several parameter sets for EMBLEM in case of $B = 1$. The probability of decryption failure is set to 2^{-140} and k, t, and v are set to satisfy $k \times t \times v = 256$ so that we can encapsulate the key of length 256-bit. From **I.A** to **I.F**, the public key size decreases and the ciphertext size increases. In **I.C**, the sum of the sizes of both public key and ciphertext is the smallest. EMBLEM generates a secret matrix from a 256-bit seed, thus providing a very small secret key size compared to other LWE-based schemes [BCD+16, CKLS16]. Since a secret key is generated from the seed (by using PRF), the secret key size remains the same even if k increases.

We also measured the execution time of each algorithm through software implementation. For more information on software implementation, see *Digital and Optical Media* in our submission package. As k increases from **I.A** to **I.F**, more computation is required to generate the secret key from the *seed*, which increases the execution time of the Key Generation algorithm. Also, as v decreases from **I.A** to **I.F**, the computation required to generate the ciphertext becomes small. Since we applied the KEM variant of Fujisaki-Okamoto transformation to construct EMBLEM, the Encapsulation algorithm operates as a subroutine in Decapsulation algorithm, so the execution time of Encapsulation and Decapsulation algorithms is similar.

(B=1)	**I.A**	**I.B**	**I.C**	**I.D**	**I.E**	**I.F**
m	1003	1003	1003	1003	1003	1003
n	770	770	770	770	770	770
k	1	2	4	8	16	32
$\log_2(q)$	24	24	24	24	24	24
σ	25	25	25	25	25	25
t	8	8	8	8	8	8
v	32	16	8	4	2	1
Public key size (bytes)	3,041	6,050	12,068	24,104	48,176	96,320
Secret key size (bytes)	32	32	32	32	32	32
Ciphertext size (bytes)	74,048	37,088	18,608	9,368	4,748	2,438
KeyGen (ms)	10.608	10.602	12.452	14.523	16.056	20.337
Encap (ms)	30.407	15.101	7.714	4.055	2.157	1.184
Decap (ms)	30.603	15.16	7.637	3.969	2.051	1.158

Table 3: Performance analysis of LWE instance in column **I** of Table 1

In Table 4, we have described several parameter sets for EMBLEM in case of $B = 2$. As in Table 3, the probability of decryption failure is set to 2^{-140} and k, t, and v are set to satisfy $k \times t \times v = 256$ so that we can encapsulate the key of length 256-bit. From **II.A** to **II.F**, the public key size decreases and the ciphertext size increases. In **II.C**, the sum of the sizes of both public key and ciphertext is the smallest. Since the 256-bit *seed* is stored as a secret key and expanded into an $n \times k$ matrix using PRF, the secret key size is constant at 256-bit, independent of n and k.

We also measured the execution time of each algorithm in case of $B = 2$. Since the parameters m and n are smaller than in the case of $B = 1$, the execution time of each algorithm is relatively reduced. From **II.A** to **II.F**, the execution time of the Key Generation algorithm increases and that of the Encapsulation and Decapsulation algorithms decreases.

(B=2)	**II.A**	**II.B**	**II.C**	**II.D**	**II.E**	**II.F**
m	832	832	832	832	832	832
n	611	611	611	611	611	611
k	1	2	4	8	16	32
$\log_2(q)$	24	24	24	24	24	24
σ	25	25	25	25	25	25
t	8	8	8	8	8	8
v	32	16	8	4	2	1
Public key size (bytes)	2,528	5,024	10,016	20,000	39,968	79,904
Secret key size (bytes)	32	32	32	32	32	32
Ciphertext size (bytes)	58,784	29,456	14,792	7,460	3,794	1,961
KeyGen (ms)	6.851	7	8.223	9.698	10.517	12.839
Encap (ms)	23.659	11.924	6.019	3.185	1.633	0.884
Decap (ms)	23.548	11.894	5.997	3.111	1.705	0.849

Table 4: Performance analysis of LWE instance in column **II** of Table 1

In Table 5, we have described the performance of the parameters for R.EMBLEM in Table 2. Basically, the parameters in Table 5 have the probability of decryption failure of 2^{-140}, and the secret key is a 256-bit seed. In columns **iii** and **iv**, σ is set very small to reduce the size of q.

The size of the public key, secret key, and the ciphertext are calculated corresponding to n of each column in Table 5.

	i	**ii**	**iii**	**iv**
n	463	320	504	437
$\log_2(q)$	16	16	14	14
σ	25	25	3	3
t	1	1	1	1
Public key size (bytes)	958	672	914	797
Secret key size (bytes)	32	32	32	32
Ciphertext size (bytes)	1,470	1,184	1,362	1,245

Table 5: Performance analysis of Ring LWE instances in Table 2

However, in practice, in order to apply the Number Theoretic Transform (NTT) operation, n is set to 512, which is a power of two. Even if n becomes larger, applying NTT operation is more efficient in terms of computational complexity. Therefore, the parameters of columns **i** and **ii** become equal, and the same goes for the parameters of columns **iii** and **iv**. In Table 6, NTT.KeyGen, NTT.Encap, and NTT.Decap represent the algorithm execution time when NTT operation is applied. In column **iii, iv**, since q is smaller, the execution time of KeyGen algorithm is reduced to almost half of that in column **i, ii**, and Encap and Decap algorithms are slightly faster than in column **i, ii**.

	i, ii	**iii, iv**
n	512	512
$\log_2(q)$	16	14
σ	25	3
t	1	1
Public key size (bytes)	1,056	928
Secret key size (bytes)	32	32
Ciphertext size (bytes)	1,568	1,376
NTT.KeyGen (ms)	0.138	0.052
NTT.Encap (ms)	1.137	1.001
NTT.Decap (ms)	1.205	1.03

Table 6: Performance analysis of Ring LWE instances applying NTT operation

2.3 Known Answer Test values

Known Answer Test (KAT) values that can be used to determine the correctness of an implementation of the submitted algorithms are provided in a zip file of digital and optical media.

2.4 Security Strength

2.4.1 Security Strength Categories

- **Security strength** In [LP11], for the parameter set $(n, q, s) = (256, 4093, 8.35)$, the estimated runtime/advantage ratio is about 2^{120} seconds, which is compared to the security of AES 128. Since we provide quite larger parameter sets, in Table 1 and Table 2, than $(n, q, s) = (256, 4093, 8.35)$, it is reasonable to assume that, at a minimum, the proposed parameters provide higher security than that of AES 128.

2.4.2 Additional Security Properties

- **Perfect forward secrecy.** Basically, since EMBLEM and R.EMBLEM are key encapsulation mechanisms (KEMs), they do not consider perfect forward secrecy.
- **Resistance to side-channel attacks.** We can make the process of sampling Gaussian errors using cumulative distribution tables (CDT) resilient to memory and timing side-channel attacks, by always scanning all elements and performing comparisons with branchless arithmetic operations [BCD+16]. In addition, by implementing various countermeasure against side-channel analysis [DSVC+15, VG15, Pes16, PPM17], we can make the proposed constructions resistant to side-channel attacks.
- **Resistance to multi-key attacks.** The multi-key setting can be seen as a generalization of the multi-user setting. In [BBM00], Bellare et al. addressed that, if a public-key encryption scheme is polynomially-secure against chosen-plaintext (resp. chosen-ciphertext) attack in the single-user setting, then it is also polynomially-secure against chosen-plaintext (resp. chosen-ciphertext) attack in the multi-user setting. Since EMBLEM and R.EMBLEM are proven to be IND-CCA secure in the single-user setting, we can say that they are IND-CCA secure in the multi-user setting as well.
- **Resistance to misuse.** If coding errors occur or primitives, such as the random number generator, used in the implementation are malfunctioning, vulnerabilities may naturally arise. Assume that two identical random values are generated by the malfunctioned random number generator and two different messages are encrypted with the same random value. If an attacker obtains a message for one ciphertext, the rest can be easily recovered from the other ciphertext.

2.5 Analysis with respect to Known Attacks

• **Estimating the security of LWE instances to known attacks.** Recently, a sage module for estimating the concrete hardness of LWE instances has been studied [APS15, AGL+17]. This module covers the following algorithms: meet-in-the-middle exhaustive search, coded-BKW [GJS15], dual-lattice attack and small/sparse secret variant [Alb17], lattice-reduction + enumeration [LP11], primal attack via uSVP [AFG13, BG14], Arora-Ge algorithm [AG11] using Gröbner bases [ACFP14]. Using this result, we analyze the security of our proposed parameters with `reduction_cost_model = BKZ.sieve`, which refers to BKZ 2.0 estimates. The hardness of the LWE instances (with small secrets) in Table 1 with respect to uSVP, dec, and dual attacks are estimated as follows:

	I	**II**
	[-1,1]	[-2,2]
uSVP	$2^{128.3}$	$2^{128.3}$
dec	$2^{191.4}$	$2^{147.0}$
dual	$2^{137.4}$	$2^{142.5}$

Table 7: Estimated hardness of LWE instances

The abbreviation “SVP” refers to the minimum of standard (primal) embedding, dual embedding, and Bai-Galbraith embedding, “dec” refers to the decoding attack, and “dual” refers to the distinguishing attack.

In column **I** of Table 1, when $m = 1003$ and $n = 770$, the security levels against uSVP, dec, and dual attacks are 128.3, 191.4, and 137.4, respectively. Reducing m and n to 826 and 634 lowers the computational complexity of the algorithm and improves efficiency, but the security levels it provides are reduced to 105.0, 154.0, and 112.9 as well. That is, when q and σ are fixed, using m and n that are smaller than the values in column **I** of Table 1 will not meet the 128-bit security level. The column **II** in Table 1 also provides security of at least $2^{128.3}$. As with column **I**, the smaller the parameter (m, n) is, the lower the security level is. Figure 6 shows the sizes of the parameters n and m according to the root Hermite factor δ and illustrates the corresponding security levels against uSVP, dec, and dual attacks. L.H.S of Figure 6 describes the case where the secret key is sampled from [-1,1], and R.H.S describes the case where the secret key is sampled from [-2,2]. To provide the same security level, R.H.S requires smaller n and m than L.H.S.

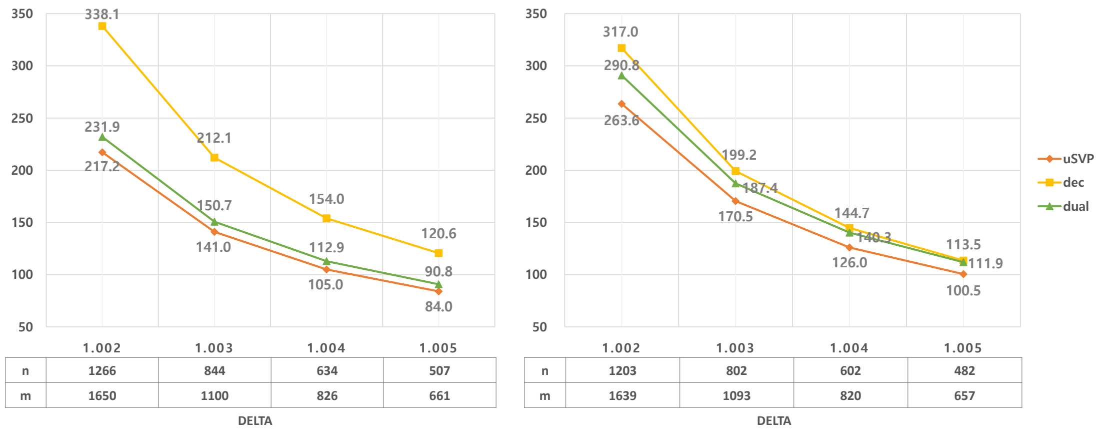


Figure 6: Security of LWE instances against known attacks

• **Estimating the security of Ring LWE instances to known attacks.** The hardness of the Ring LWE instance (with small secrets) in Table 2 with respect to uSVP, dec, and dual attacks are given in Table 8. Note that, in **i** and **ii** of Table 2, the standard deviation $\sigma = 25$ and $\log_2(q) = 16$, whereas in **iii** and **iv**, $\sigma = 3$ and $\log_2(q) = 14$.

	i	**ii**	**iii**	**iv**
	[-1,1]	[-2,2]	[-1,1]	[-2,2]
uSVP	$2^{128.1}$	$2^{128.1}$	$2^{128.3}$	$2^{128.3}$
dec	$2^{231.7}$	$2^{153.9}$	$2^{179.1}$	$2^{152.9}$
dual	$2^{144.1}$	$2^{148.0}$	$2^{142.5}$	$2^{147.7}$

Table 8: Estimated hardness of Ring LWE instances

The four graphs in Figure 7 illustrate the columns **i**, **ii**, **iii**, and **iv** in Table 2 in order from the left. In column **i** of Table 2, when $n = 463$, the security levels against uSVP, dec, and dual attacks are 128.1, 231.7, and 144.1, respectively. Reducing n to 395 lowers the computational complexity of the algorithm and improves efficiency, but the security levels it provides are reduced to 109.6, 194.5, and 124.3 as well. However, when applying the Number Theoretic Transform (NTT) operation, since n should be a power of 2, we set $n = 512$ in both cases. As a result, reducing n from 463 to 395 is meaningless in practice. If n is further reduced to 237, the efficiency can be improved significantly by setting $n = 256$ in NTT operation. However, the security level at this time is as low as 68.5.

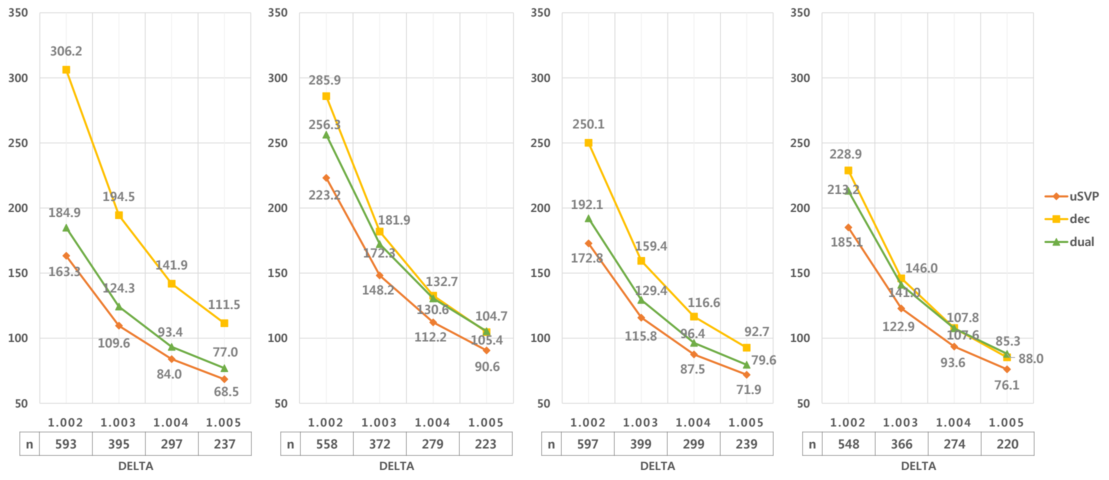


Figure 7: Security of Ring LWE instances against known attacks

2.6 Advantages and Limitations

2.6.1 Advantages

- **Standard (R)LWE assumption.** EMBLEM is a key encapsulation mechanism (KEM) secure against adaptive chosen ciphertext attacks (namely, IND-CCA2 secure), and its construction is based on the small secret LWE problem. R.EMBLEM is a ring version of EMBLEM, also IND-CCA2 secure, and its construction is based on the Ring LWE problem with small secret. Based on the previous results, we set the parameters so that the small secret (Ring) LWE problem becomes as hard as the standard (Ring) LWE problem. Therefore, by rescaling the parameter set, we can be confident that EMBLEM (resp. R.EMBLEM) is secure based on the hardness of *standard* LWE (resp. Ring LWE) problem.

- **Small secret key.** In the LWE instance with small secrets, the secret key is chosen uniformly from $[-B, B]$ for a positive integer $B < \sigma$, rather than from the Gaussian distribution. Because of this nature, it is only necessary to store a 256-bit *seed* to generate the secret key, without having to store the entire matrix. The secret key can be derived from the seed by using pseudorandom functions. As a result, the size of the secret key in our constructions can be greatly reduced.

- **Worst-case to average-case reduction.** In addition, in the parameter sets presented in Table 1 and the columns **i, ii** of Table 2, the standard deviation σ is set to be larger than $\sqrt{2n/\pi}$. That is, the Gaussian parameter is larger than $2\sqrt{n}$, so our constructions have worst-case to average-case security reduction. Also, all of the parameter sets presented provide 128-bit quantum security.

- **New multi-bit encoding method.** Furthermore, we present a new approach to error handling in the decryption phase. The existing LWE-based public key encryption schemes or KEMs eliminate errors generated in the decryption phase through a rounding function. Rounding involves comparison operations, and if multiple bits are encapsulated in each entry of ciphertext, more comparison operations are required. In our approach, we can restore multiple bits simply by parsing the most significant t bits of each entry of a ciphertext, without rounding. By separating the message from the error and inserting the error-blocking bit 1 between them, we prevent errors generated in the decryption phase from being propagated to the message. Using the parameters given in Table 1, an 8-bit message per entry of a ciphertext is encapsulated (i.e., $t = 8$), and the message can be restored by parsing the most significant 8 bits per entry in the decryption phase.

- **Negligible probability of decryption failure.** Finally, we guarantee a very small probability of the decryption failure ($\approx 2^{-140}$). This allows CCA transformation from EMBLEM.CPA (resp. R.EMBLEM.CPA) to EMBLEM (resp. R.EMBLEM). In the announcement by the NIST, the proposed KEM should be semantically secure under adaptive chosen ciphertext attack, namely *IND-CCA2 security*, and it may be assumed that the attacker has access to the decryption oracle approximately 2^{64} times. Therefore, it is important to reduce the correctness error of the proposed scheme so that it can meet the security requirement suggested by NIST.

2.6.2 Limitations

- **Larger size of parameters.** EMBLEM and R.EMBLEM are basically constructed based on the (Ring) LWE problem with small secret. Therefore, the parameter size should be larger than the (Ring) LWE instance in order to provide the same security level. As a result, the size of the public key and the ciphertext becomes somewhat larger. In our new error-blocking approach, we need to set q somewhat larger, because we need to ensure that errors occurring in the decryption phase do not affect the message part. That is, the size of q is equal to the sum of the followings: size of the error in the decryption phase, one bit to prevent error propagation, and the number of bits to be encapsulated in each entry of a ciphertext. As q increases, the size of the public key, secret key, and the ciphertext grows proportionally, and computational complexity also increases.

3 DIGITAL AND OPTICAL MEDIA

All electronic data is provided in the zip file in the submitted package. For more details, refer to the corresponding files. This media has the following structure:

- README : This file includes the list of all files in a zip file of *Digital and Optical Media*.
- Reference_Implementation : This file includes the reference implementation code, which helps to understand how the submitted algorithm is implemented.
- Optimized_Implementation : This file includes the optimized implementation code, which is used to demonstrate the performance of the submitted algorithm.
- KAT : This file includes all of the required test values to determine the correctness of an implementation of the submitted algorithms.
- Supporting_Documentation : This file describes how subroutines of the submitted algorithms are implemented, for public review.

4 INTELLECTUAL PROPERTY STATEMENTS

The following statements will be given to NIST at the first PQC Standardization Conference, if our submission package is "complete and proper" and will be posted for public review.

1. statement by the submitter
2. statement by patent (and patent application) owner(s) (if applicable)
3. statement by reference/optimized implementations' owner(s)

References

[ACFP14] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugre, and Ludovic Perret. Algebraic algorithms for lwe. Cryptology ePrint Archive, Report 2014/1018, 2014. `http://eprint.iacr.org/2014/1018`.

[AFG13] Martin R Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving lwe by reduction to unique-svp. In *International Conference on Information Security and Cryptology*, pages 293–310. Springer, 2013.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *International Colloquium on Automata, Languages, and Programming*, pages 403–415. Springer, 2011.

[AGL$^+$17] Martin Albrecht, Florian Göpfert, Cedric Lefebvre, Rachel Player, Markus Schmidt, and Sam Scott. A sage module for estimating the concrete security of learning with errors instances, 2017.

[Alb17] Martin R Albrecht. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 103–129. Springer, 2017.

[AOP$^+$17] Martin R. Albrecht, Emmanuela Orsini, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. Tightly secure ring-lwe based key encapsulation with short ciphertexts. In *Computer Security - ESORICS 2017, Part I*, pages 29–46. Springer, 2017.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[Ban95] Wojciech Banaszczyk. Inequalities for convex bodies and polar reciprocal lattices in r n. *Discrete & Computational Geometry*, 13(1):217–231, 1995.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 259–274. Springer, 2000.

[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. ACM, 2016.

[BDK$^+$17] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, and Damien Stehlé. Crystals–kyber: a cca-secure module-lattice-based kem. Technical report, Cryptology ePrint Archive, Report 2017/634, 2017. http://eprint. iacr. org/2017/634. 18, 2017.

[BG14] Shi Bai and Steven D Galbraith. Lattice decoding attacks on binary lwe. In *Australasian Conference on Information Security and Privacy*, pages 322–337. Springer, 2014.

[CKLS16] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yong Soo Song. Lizard: Cut off the tail!//practical post-quantum public-key encryption from lwe and lwr. *IACR Cryptology ePrint Archive*, 2016:1126, 2016.

[DSVC$^+$15] François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order dpa with projection pursuits. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 34–50. Springer, 2015.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-bkw: solving lwe using lattice codes. In *Annual Cryptology Conference*, pages 23–42. Springer, 2015.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. 2010.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. Technical report, Cryptology ePrint Archive, Report 2017/604, 2017. http://eprint. iacr. org/2017/604, 2017.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558, pages 319–339. Springer, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43, 2013.

[Lyu16] Vadim Lyubashevsky. Digital signatures based on the hardness of ideal lattice problems in all rings. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II 22*, pages 196–214. Springer, 2016.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of sis and lwe with small parameters. In *Advances in Cryptology–CRYPTO 2013*, pages 21–39. Springer, 2013.

[Pes16] Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *Progress in Cryptology–INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings 17*, pages 153–170. Springer, 2016.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 513–533. Springer, 2017.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[Sta] Xilinx Staff. Gate count capacity metrics for fpgas. *Xilinx Corp., San Jose, CA, Application Note XAPP*, 59.

[TU16] Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the fujisaki-okamoto and oaep transforms. In *Theory of Cryptography Conference*, pages 192–216. Springer, 2016.

[VG15] Praveen Kumar Vadnala and Johann Großschädl. Faster mask conversion with lookup tables. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 207–221. Springer, 2015.

A Hardware Architecture of EMBLEM

A.1 Hardware Architecture of EMBLEM.CPA

In this section, we describes the hardware architecture design of EMBLEM.CPA. Our design is based on the EMBLEM.CPA scheme and it is implemented to the Zynq-7 FPGA. Also we omit control-path because we want to show a simple arithmetic logic circuit in our scheme.

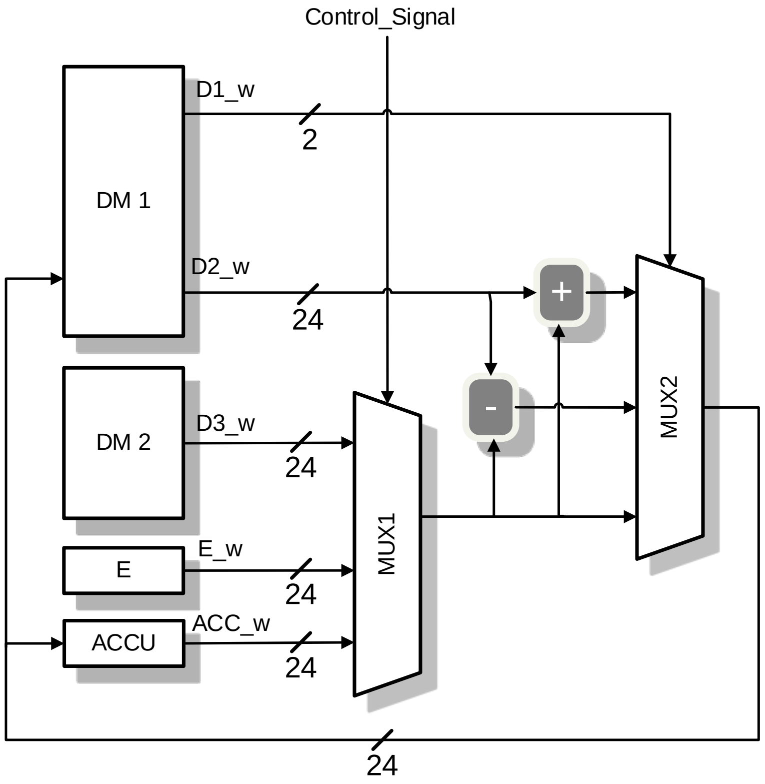


Figure 8: EMBLEM.CPA Data Path

Figure 8 illustrates the high-level architecture of EMBLEM.CPA. An initialization state is required to read Matrices $\mathbf{A}, \mathbf{B}, \mathbf{R}^T$, and $\mathbf{X}$ as well as 24-bit Gaussian sampling error and a matrix $\mathbf{M}$ for key generation, encryption, and decryption. We design the core architecture which can be used as key generation, encryption and decryption. The core architecture consists of two main parts: Arithmetic parts for three algorithms and data memory parts. The arithmetic parts process addition and substraction only for matrix calculations. The data memory parts consist of two detail parts. One is a read/write storage for $\mathbf{A}, \mathbf{B}, \mathbf{R}$, result value, $\mathbf{C_1}$ and the other is read only memory for $\mathbf{M}$ or $\mathbf{C_2}$.

The details of design are as follows. In Figure 8, the sel_1 wire of MUX1 and MUX2 is used to select data of E_w, ACC_w, D1_w, D2_w, D3_w. The Data memory DM2 can store $\mathbf{C_2}$ or $\mathbf{M}$ depending on each algorithm. The parallelized adder and substractor are performed at the same time. Since then, two calculated results and bypass signal can be selected with MUX2 according to 2 bits D1_w whose data will be $\mathbf{R}^T$ or $\mathbf{X}$ which can be decided by each three algorithm. Finally, data output by MUX2 is stored at ACCU register which can be used as the next operand. Once a matrix calculation is completed, ACCU value is selected by MUX1 and forwarded to DM1 data memory. As a result, it is working on pipelining register.

A.2 Hardware Architecture of EMBLEM

EMBLEM hardware architecture only includes hash functions and 256-bit output random number generator, so we will not illustrate the detail of EMBLEM hardware architecture diagram.

A.3 Finite State Machine of EMBLEM

A.3.1 Key Generation

Figure 9 shows our simple finite state machine (FSM) for key generation. The details of FSM are as follows. The first state initializes the core, and is required for ACCU register reset. The Matrices $\mathbf{A}, \mathbf{X}$ as well as 24-bit Gaussian sampling error for key generation should be loaded onto the data memory and register separately at this state. The second state reads data $\mathbf{A}$ and $\mathbf{X}$ from the data memory. The next step is a matrix calculation state. In this state, each column of matrix $\mathbf{A}$ is calculated with $\mathbf{X}$ and $\mathbf{E}$. If C_Count = 770, the memory write state (MEM Write) is entered, and then the R_Count=1003 cycles are rotated.

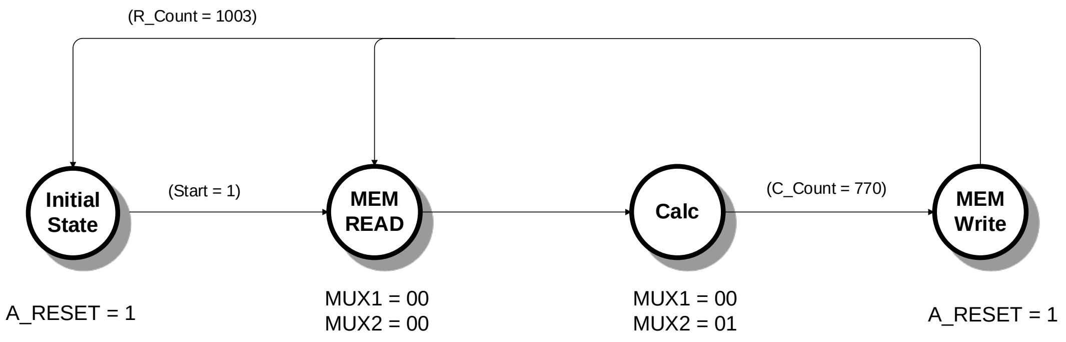


Figure 9: Key Generation Finite State Machine Diagram

A.3.2 Encryption

Figure 10 shows our simple finite state machine for encryption. When the encryption mode started, the first state initializes the core, and it is required for ACCU register reset. The Matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{R}^T$ as well as 24-bit Gaussian sampling error and matrix $\mathbf{M}$ for key generation should be loaded onto data memory and register at this time. The second state reads data $\mathbf{R}^T$, $\mathbf{A}$ or $\mathbf{B}$ from the data memory. The next step is matrix calculation state. In this state, each column of matrix $\mathbf{A}$ or $\mathbf{B}$ is calculated with $\mathbf{R}^T$ and E. If (C_Count = 1003 & C1_flag = 1) | (C2_flag = 1 & C_Count = 1003) | (M_flag = 1 & C_Count = 1), the memory write state is entered, and then the C_Count = 32 cycles are rotated.

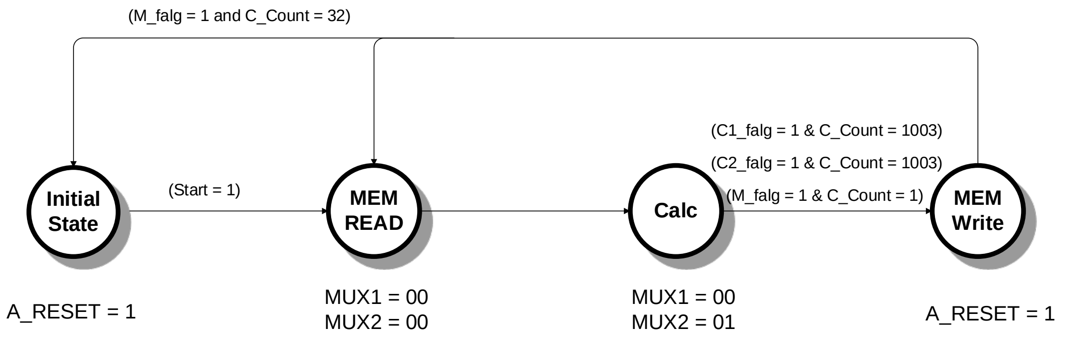

(M_falg = 1 and C_Count = 32)
(Start = 1)
(C1_falg = 1 & C_Count = 1003)
(C2_falg = 1 & C_Count = 1003)
(M_falg = 1 & C_Count = 1)
Initial State
MEM READ
Calc
MEM Write
A_RESET = 1
MUX1 = 00
MUX2 = 00
MUX1 = 00
MUX2 = 01
A_RESET = 1

Figure 10: Encryption Finite State Machine Diagram

A.3.3 Decryption

Figure 11 shows our simple finite state machine for decryption. The details of FSM are as follows. The first state initializes the core, and it is required for ACCU register reset. The matrices $\mathbf{C_1}$, $\mathbf{X}$ as well as $\mathbf{C_2}$ should be loaded onto the data memory at this state. The second state reads data $\mathbf{C_1}$ and $\mathbf{X}$ from the data memory. The next step is matrix calculation state. In this state, each column of the ciphertext $\mathbf{C_1}$ is calculated with $\mathbf{X}$ and $\mathbf{C_2}$. If C_Count = 770, the memory write state is entered, and then the R_Count = 32 cycles are rotated.

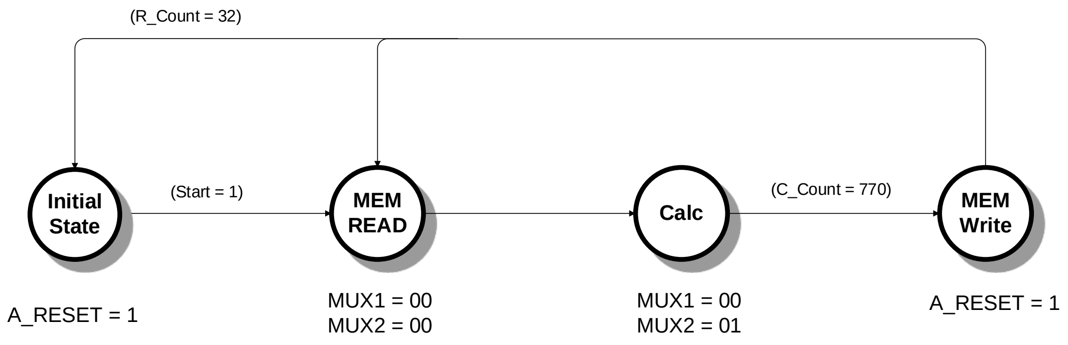


Figure 11: Decryption Finite State Machine Diagram

A.4 Performance

A.4.1 Latency

To evaluate the performance of EMBLEM, we implemented it on an FPGA using Xilinx EDA tool with default option. We use a commercially available 28 nm Zynq-7000 device for those three algorithms. As a result, our design was successfully tested by post implement simulation. Table 9 provides the actual runtime latency of three algorithms. The throughput of the EMBLEM core is measured on the experimental setup and the measured maximum frequency is about 200 MHz. Our EMBLEM design is not optimized for improving its throughput, thus it seems that there is much opportunity to speed-up its performance.

Algorithm	**Operation**	**Device**	**Cycles**	**Latency**
KeyGen	$2 + km(n + 2)$		776.3k	3.8ms
Encrypt	$2 + vn(m + 2) + v(m + 2) + v(k + 2)$	xc7z020	24.7M	123ms
Decrypt	$2 + v(n + 2)$		24.7k	0.1ms

Table 9: Latency of EMBLEM

A.4.2 Memory

To estimate the memory size of our implementation of EMBLEM, we provide memory footprint estimation (d.g., Block-Memory, DRAM) for three algorithms, which are the key generation, encryption and decryption, respectively. As aforementioned, Matrix $\mathbf{A}$, $\mathbf{B}$, $\mathbf{X}$, $\mathbf{R}^T$, and $\mathbf{C_1}$ are stored in the memory DM1 on each algorithm, so the stored matrix data are defined as *Read Data Size* in the table below. The ciphertext $\mathbf{C_2}$ and plaintext $\mathbf{M}$ were implemented using the read only memory DM2. The *Write Data Size* means that the result would be stored back to data memory.

The memory for Gaussian sampling error matrix is not considered in these tables because only 24-bit register is needed in our core design.

Memory	**Read Data Size**	**Write Data Size**	**Total kB**	**Total KiB**
DM1	$\log_2(q)(m \times n)$ $+ \log_2(\{-1,0,1\})(n \times k)$	$\log_2(q)(m \times k)$	2,317.12	2,262.81
DM2	-	-		

Table 10: Memory size of Key Generation in EMBLEM

Memory	**Type**	**Read Data Size**	**Write Data Size**	**Total kB**	**Total KiB**
DM1	C1	$\log_2(q)(m \times n)$ $+ \log_2(\{-1,0,1\})(n \times k)$	$\log_2(q)(v \times n)$	363.82	355.3
	C2	$\log_2(\{-1,0,1\})(m \times k)$ $+ \log_2(q)(v \times k)$$	$\log_2(q)(v \times k)$		
DM2	M	$\log_2(q)(32 \times 1)$	-		

Table 11: Memory size of Encryption in EMBLEM

Memory	**Read Data Size**	**Write Data Size**	**Total kB**	**Total KiB**
DM1	$\log_2(q)(v \times n)$ $+ \log_2(\{-1,0,1\})(n \times k)$	$\log_2(q)(v \times k)$	74.3	72.56
DM2	$\log_2(q)(v \times k)$	-		

Table 12: Memory size of Decryption in EMBLEM

A.4.3 Utilization

The summary of resource utilization is presented in Table 13. The EMBLEM core data path is implemented using an LUT level instantiations mostly with Xilinx primitive libraries. To evaluate the gate count, we use the gate count estimation in [Sta].

Device	**BRAM/DSP48E/FFs/LUTs**	**Gate Count**
xc7z020	0/0/24/48	576

Table 13: EMBLEM Core Estimated Utilization (Data Path only)