LizarMong: Excellent Key Encapsulation Mechanism based on RLWE and RLWR

Chi-Gon Jung[1], JongHyeok Lee[2⋆], Youngjin Ju[3⋆], Yong-Been Kwon[4⋆], Seong-Woo Kim[1], and Yunheung Paek[5]

[1] Department of Engineering Practice, Seoul National University, Seoul, Korea
{cgjung, snwoo}@snu.ac.kr
[2] Department of Financial Information Security, Kookmin University, Seoul, Korea
n_seeu@kookmin.ac.kr
[3] Department of Mathematics, Hanyang University, Seoul, Korea
jyj9327kr@hanyang.ac.kr
[4] Department of IT Convergence, Hansung University, Seoul, Korea
vexyoung@gmail.com
[5] Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea
ypaek@snu.ac.kr

Abstract. The RLWE family algorithms submitted to the NIST post-quantum cryptography standardization process have each merit in terms of security, correctness, performance, and bandwidth. However, there is no splendid algorithm in all respects. Besides, various recent studies have been published that affect security and correctness, such as side-channel attacks and error dependencies. To date, though, no algorithm has fully considered all the aspects. We propose a novel Key Encapsulation Mechanism scheme called LizarMong, which is based on RLizard. LizarMong combines the merit of each algorithm and state-of-the-art studies. As a result, it achieves up to 85% smaller bandwidth and 3.3 times faster performance compared to RLizard. Compared to the NIST's candidate algorithms with a similar security, the bandwidth is about 5-42% smaller, and the performance is about 1.2-4.1 times faster. Also, our scheme resists the known side-channel attacks.

Keywords: Lattice-based Cryptography · Ring-LWE · Ring-LWR.

1 Introduction

Among candidates for the National Institute of Standards and Technology (NIST) post-quantum cryptography standardization process [12], the Ring Learning With Error (RLWE) family[†] is the spotlight in Key Encapsulation Mechanism (KEM) because of its proven hardness, small bandwidth, and good performance.

⋆ These authors contributed equally to this work.

† Ring-LWE (RLWE), Ring-Learning With Rounding (RLWR), Module-LWE (MLWE), Module-LWR (MLWR), Integer-MLWE (I-MLWE)

The design factors of the RLWE family algorithms consist of the underlying problems, the choice of ring, the dimensions of the lattice, the modulus, and the error rate determined by the ratio between the standard deviation of the error distribution and the modulus. These factors have trade-offs and determine security, correctness, performance, and bandwidth. Looking at NIST's candidate algorithms from a trade-off perspective, we can see some notable characteristics.

First, the choice of the underlying problems and the ring determines the innate temperament of the algorithm. The underlying problems are classified as RLWE, Ring Learning With Rounding (RLWR), Module-LWE (MLWE), and MLWR. RLWE has good performance and bandwidth. Also, RLWE has been well-studied for the most prolonged time among the underlying problems of lattice from algebraic structures and is used for other schemes such as homomorphic encryption. Thus, RLWE can claim that it is more conservative security than other underlying problems of the lattice from algebraic structures. MLWE can reduce bandwidth because they are small and flexible in dimension choice, but require more computation if into dimension the same as RLWE. RLWR and MLWR discard some Least Significant Bits (LSBs) instead of error sampling, resulting in better performance and bandwidth compare to RLWE and MLWE. The ring is commonly chosen as the cyclotomic polynomial $X^n + 1$ due to performance and security, where n is a power of two. Exceptionally, Round5 [8] uses the $X^{n+1} - 1$ cyclotomic polynomial. Since $n + 1$ is prime, this polynomial is less constrictive in n choosing. So, it can choose an optimized n for each security level. The required bandwidth can be reduced. However, $X^{n+1} - 1$ is more expensive computation to polynomial modular reduction operation than common polynomial. The algorithm with the smallest bandwidth among NIST's candidates is Round5, which based on RLWR and does not use $X^n + 1$ polynomial. Thus, it is important to choose the underlying problem and the ring.

Another notable characteristic is the modulus size. Modulus size is the main factor that affects the correctness, bandwidth, and performance of RLWE family schemes. Large modulus (2^{12-14}) like Newhope [5] reach security-level by adding relatively large error, but can maintain correctness because of a small error rate. Large modulus, however, increases computation and bandwidth. Thus, they use fast multiplication algorithms (e.g., NTT), public-key compression, and ciphertext compression to solve this problem. Conversely, small modulus (2^{8-12}) like LAC [24] can have relatively low bandwidth and good performance. However, if a large error such as the large modulus is used, the correctness decreases as the error rate increases. Thus, these are using tiny error and secret, and error correction code to improve correctness.

Meanwhile, NIST's standardization process has recently been making impressive results by promoting a study on the RLWE family. In particular, the study that disproves the independent assumptions about the failure of individual bits to calculate the overall failure rate [16], as well as various side-channel attack studies, is meaningful because of affecting most RLWE family algorithms. To date, however, many algorithms do not include most of those studies.

As mentioned above, each algorithm complements the trade-off with brilliant techniques. Unfortunately, however, choosing one excellent algorithm in all aspects (security, performance, bandwidth, and correctness) is a hard decision. Furthermore, considering the recent studies, it is almost infeasible to choose one.

In this paper, we propose a novel key encapsulation mechanism scheme called LizarMong that is excellent in all aspects. It combines each merit of NIST's candidate algorithms with state-of-the-art studies.

Contributions The contributions of this study can be summarized as follow:

- To improve bandwidth and performance, we set small dimensions and modulus, and apply ciphertext and public-key compression.
- We adopt the error correction code called XE5 [8] to compensate for the reduced correctness due to small modulus.
- Resistance to known side-channel attacks, we devise a sparse polynomial multiplication with hiding. Also, we do not use the Cumulative Distribution Table (CDT) technique as the error sampler.
- We estimate the correctness more conservatively by calculating the decryption failure rate considering the dependency of each bit error.

2 Preliminaries

2.1 Notation

The log indicates the logarithm with base 2. For a positive integer q, we use $\mathbb{Z} \bigcap (-q/2, q/2]$ as a representative of $\mathbb{Z}_q$. We denote by R_q the ring $\mathbb{Z}_q[X]/(X^n+1)$. Bold lower-case letters represent polynomials with coefficients in R_q. For a polynomial $\mathbf{a}$, we write $a(i)$ to denote it's the coefficient of order i. Multiplication in R_q is represented by $*$. $\lfloor r \rceil$ is the rounding to the nearest integer to real r, and $\lfloor \mathbf{a} \rceil$ is the rounding to the nearest integer for each coefficient in the polynomial $\mathbf{a}$. $\|x\|$ means the l_2 norm. $x \parallel y$ is the concatenation of x and y. There are two distributions used in this paper, $HWT_n(h)$ and ψ_{cb}. $HWT_n(h)$ is the uniform distribution over the subset of $\{-1, 0, 1\}^n$ whose elements contain $n - h$ number of zeros. ψ_{cb} is the centered binomial distribution with mean zero and standard deviation $\sqrt{cb/2}$. $\texttt{SHAKE256}(m, len)$ is a hash function that receives m and outputs a byte-string of the length len. eccENC and eccDEC are functions for encoding and decoding using the error correction code.

2.2 RLizard

RLizard is the KEM and PKE based on RLWE and RLWR submitted to the NIST standardization process 1round. RLizard uses RLWE for key-generation, considering relatively conservative security, and RLWR for encryption and decryption to improve bandwidth and performance. Another effort to ensure robust security in key-generation is the adoption of CDT from Gaussian distribution

as an error sampler. It is a high precision sampler that does not damage the original RLWE. The ring is chosen as the common form $X^n + 1$. RLizard uses a small modulus and sparse ternary secret, which improved correctness and performance. Moreover, for providing IND-CCA2 PKE and KEM, they use a variant of Fujisaki-Okamoto transformation [18]. As a result, RLizard supports IND-CCA2 PKE and KEM and enjoys fast encryption and decryption, robust security, and high correctness. However, the bandwidth is relatively large.

3 LizarMong

In this section, we detail our KEM scheme called LizarMong. Our goal is to satisfy both security and correctness while making excellent performance and bandwidth. To achieve the goal, LizarMong was designed by adequately combining the design elements of NIST's candidate algorithms and their superior techniques that are used to compensate for the trade-off. Our scheme also considered recent studies such as side-channel attacks and dependency error issues.

3.1 Design Element Selection

Choice of the Ring We use $f(X)=X^n + 1$ in $R_q := \mathbb{Z}_q[X]/(f(X))$, where n is the power of two. It is the common choice used by most of NIST's candidate algorithms and RLizard. The common ring has the advantage in that the polynomial modular reduction operation is straightforward, and there have been no known attacks exploit it [23].

Modulus selection We select $q = 256$, which is small and to the power of two. Intuitively, this choice enjoys a small bandwidth and improved performance. It also provides very efficient modulo operation and memory usage and is suitable for single instruction multiple data (SIMD) implementations such as AVX2 and NEON. Even though the modulus is small, it can not affect the security since we maintain the error rate by selecting proper error distribution [24,27]. The modulus p used for RLWR and the modulus k used for ciphertext compression are also to the power of two. It improve performance by replacing $\lfloor (p/q) \cdot \mathbf{x} \rceil$ with ADD and AND operations [13].

Distribution The RLWE family can sample secret polynomial $\mathbf{s}$ and error polynomial $\mathbf{e}$ using different *Seed* in the same distribution for efficient implementation. Also, this variant has proven to be equivalent to the original RLWE problem [6]. For the above reason, most of NIST's candidate algorithms use the same distribution for error and secret sampling. Recently, however, a fault-attack [29] attempted to analyze by manipulating the *Seed* to make $\mathbf{s}$ and $\mathbf{e}$ the same value. Therefore, we sample $\mathbf{e}$ and $\mathbf{s}$ from each distribution, like the original RLWE, to remove the fault-attack point [29].

- **Error distribution** We use the centered binomial distribution with the standard deviation $1/\sqrt{2}$, i.e. the range of the distribution is {-1, 0, 1}.

Although the original RLWE is defined as a Gaussian distribution, switching to the centered binomial distribution is known to have a negligible impact on security [5]. Also, the best-known attacks against RLWE depend not on the type of the distribution but the standard deviation [5,9].

- **Secret distribution** We use a sparse ternary secret with a Hamming weight, such as RLizard. [13] and [8] proved the hardness of the sparse ternary secret variants LWE and LWR. Multiplication of sparse ternary secret polynomials can be replaced with addition (subtraction) to improve performance [1]. It also maintains correctness by preventing decryption errors from increasing.

Adopt error correction code Our analysis in Section 4.3 shows that 4-5 bits error correction capability is needed. Therefore, we adopted XE5 [8] that is specialized in the RLWE family. Since XE5 avoids table look-up and branch conditions, it resists timing attacks [8]. XE5 has a block size of 490 bits, of which 256 bits is the message, and 234 bits is parity check. Our scheme differs in message length from HILA5 [32] and Round5, which previously used XE5. δ, used in place of PKE messages in the IND-CCA2 KEM, has a significant impact on the security of the scheme. Thus, we match the length of the δ (messages) to the overall security level. The 512-bit δ (messages) in the `Strong` parameter seems to constrain the use of XE5, but we can solve it very simply. Divide the 512-bit δ (messages) in half, encode it with XE5, and concatenate it. Decoding is in reverse order. This process does not affect security and makes our calculation of correctness more conservative in our `Strong` parameters.

Compress Public-key and Ciphertext NIST's candidate algorithms commonly use compression techniques. RLizard can also use these techniques [23], although it does not include in the version submitted to NIST. Public-key compression means sending only the $Seed$ instead of $\mathbf{a}$ in R_q, and the receiver recovers $\mathbf{a}$ using the hash function. This reduces the public-key size from $2n \log q$ to size-of-$Seed$ $+ n \log q$. Ciphertext compression is similar to the RLWR idea of discarding a few LSBs in $\mathbf{c_2}$. IND-CCA2 KEM also can do the same.

3.2 Algorithm Specifications

3.2.1 IND-CPA PKE

Algorithm 1 IND-CPA.KeyGen

Input: The set of public *parameters*
Output: Public key $pk = (Seed_a \parallel \mathbf{b})$, Private Key $sk = (\mathbf{s})$

1: $Seed_a \xleftarrow{\$} \{0,1\}^{256}$
2: $\mathbf{a} \leftarrow \texttt{SHAKE256}(Seed_a, n/8)$
3: $\mathbf{s} \xleftarrow{\$} HWT_n(h_s)$ and $\mathbf{e} \xleftarrow{\$} \psi_{cb}^n$
4: $\mathbf{b} \leftarrow -\mathbf{a} * \mathbf{s} + \mathbf{e}$
5: $pk \leftarrow (Seed_a \parallel \mathbf{b})$ and $sk \leftarrow \mathbf{s}$
6: **return** pk, sk

Algorithm 2 IND-CPA.Encryption

Input: pk, Message $\mathbf{M} \in \{0,1\}^d$
Output: Ciphertext $\mathbf{c} = (\mathbf{c_1} \parallel \mathbf{c_2})$

1: $\mathbf{r} \xleftarrow{\$} HWT_n(h_r)$ and $\mathbf{M'} \leftarrow \text{eccENC}(\mathbf{M})$
2: $Seed_a, \mathbf{b} \leftarrow \text{Parsing}(pk)$
3: $\mathbf{a} \leftarrow \texttt{SHAKE256}(Seed_a, n/8)$
4: $\mathbf{c_1} \leftarrow \lfloor (p/q) \cdot \mathbf{a} * \mathbf{r} \rceil$ and $\mathbf{c_2} \leftarrow \lfloor (k/q) \cdot ((q/2) \cdot \mathbf{M'} + \mathbf{b} * \mathbf{r}) \rceil$
5: $\mathbf{c} \leftarrow (\mathbf{c_1} \parallel \mathbf{c_2})$
6: **return** $\mathbf{c}$

Algorithm 3 IND-CPA.Decryption

Input: sk, Ciphertext $\mathbf{c} = (\mathbf{c_1} \parallel \mathbf{c_2})$
Output: Message $\hat{\mathbf{M}}$

1: $\mathbf{c_1}, \mathbf{c_2} \leftarrow \text{Parsing}(\mathbf{c})$
2: $\hat{\mathbf{M}}' \leftarrow \lfloor (2/p) \cdot ((p/k) \cdot \mathbf{c_2} + \mathbf{c_1} * \mathbf{s}) \rceil$
3: **return** $\hat{\mathbf{M}} \leftarrow \text{eccDEC}(\hat{\mathbf{M}}')$

3.2.2 IND-CCA2 KEM

We design IND-CCA KEM using the transformation technique by Jiang et al. [21]. We use a hash function $H : R_2 \rightarrow HWT_n(h)$, and a hash function $G : \{0,1\}^* \rightarrow \{0,1\}^n$ for Jiang's transformation technique.

Algorithm 4 IND-CCA2-KEM.KeyGen

Input: The set of public *parameters*
Output: Public Key $pk = (Seed_a \parallel \mathbf{b})$, Private Key $sk = (sk_{cpa} \parallel \mathbf{u})$

1: $pk, sk_{cpa} :=$ IND-CPA.KeyGen (Algorithm 1)
2: $\mathbf{u} \xleftarrow{\$} R_2$
3: **return** $pk, sk \leftarrow (sk_{cpa} \parallel \mathbf{u})$

Algorithm 5 IND-CCA2-KEM.Encapsulation

Input: pk
Output: Ciphertext $\mathbf{c} = (\mathbf{c_1} \parallel \mathbf{c_2})$, Shared Key $\mathbf{K}$

1: $\delta \xleftarrow{\$} \{0,1\}^{sd}$
2: $\mathbf{r} \leftarrow H(\delta)$
3: $\delta' \leftarrow \text{eccENC}(\delta)$
4: $\mathbf{c_1} \leftarrow \lfloor (p/q) \cdot \mathbf{a} * \mathbf{r} \rceil$
5: $\mathbf{c_2} \leftarrow \lfloor (k/q) \cdot ((q/2) \cdot \delta' + \mathbf{b} * \mathbf{r}) \rceil$
6: $\mathbf{c} \leftarrow (\mathbf{c_1} \parallel \mathbf{c_2})$
7: $\mathbf{K} \leftarrow G(\mathbf{c}, \delta')$
8: **return** $\mathbf{c}, \mathbf{K}$

Algorithm 6 IND-CCA2-KEM.Decapsulation

Input: pk, sk, Ciphertext $\mathbf{c}$
Output: Shared Key $\mathbf{K}$

1: $\mathbf{c_1}, \mathbf{c_2} \leftarrow \text{Parsing}(\mathbf{c})$
2: $sk_{cpa}, \mathbf{u} \leftarrow \text{Parsing}(sk)$
3: $\hat{\delta}' \leftarrow \lfloor (2/p) \cdot ((p/k) \cdot \mathbf{c_2} + \mathbf{c_1} * sk_{cpa}) \rceil$
4: $\hat{\delta} \leftarrow \text{eccDEC}(\hat{\delta}')$
5: $\hat{\mathbf{r}} \leftarrow H(\hat{\delta})$
6: $\hat{\delta}'' \leftarrow \text{eccENC}(\hat{\delta})$
7: $\hat{\mathbf{c}} \leftarrow \lfloor (p/q) \cdot \mathbf{a} * \hat{\mathbf{r}} \rceil \parallel \lfloor (k/q) \cdot ((q/2) \cdot \hat{\delta}'' + \mathbf{b} * \hat{\mathbf{r}}) \rceil$
8: **if** $\mathbf{c} \neq \hat{\mathbf{c}}$ **then** $\mathbf{K} \leftarrow G(\mathbf{c}, \mathbf{u})$ **else** $\mathbf{K} \leftarrow G(\mathbf{c}, \hat{\delta}'')$
9: **return** $\mathbf{K}$

3.3 Parameter Selection

We construct a `Comfort` version that satisfies category1 security level (128-bit) and a `Strong` version that satisfies category5 security level (256-bit) as required by the NIST standardization process. The assessment of the security level reflected the computational complexity of all known attacks described in Section 4.2. Table 1 shows the detailed parameters of each security level and the bandwidth according to each security level is summarized in Table 2.

n is the dimension of the lattice, q is the modulus of RLWE, p is the modulus of RLWR, k is the modulus used for ciphertext compression, h_s is the Hamming weight of the secret key, h_r is the Hamming weight of the ephemeral secret used to encapsulation. d is the length of the message, and sd is the length of δ used in the IND-CCA2 conversion. cb is a variable used for the centered binomial distribution.

Table 1: The detail parameters for each security level

parameters	n	q	p	k	h_s	h_r	d	sd	cb
`Comfort` (128-bit)	512	256	64	16	128	128	256	256	1
`Strong` (256-bit)	1024	256	64	16	128	128	512	512	1

Table 2: Size of pk, sk, and ciphertext of LizarMong in bytes

Security level	**Ciphertext**	**Public key**	**Secret key**
`Comfort`	640	544	544 (210)*
`Strong`	1280	1056	1088 (290)*

* sk_{cpa} can be encoded by storing only non-zero indexes. Thus, optionally, sk can be compressed with $encoding(sk_{cpa})$, a flag of -1, and $\mathbf{u}$ (for IND-CCA2 KEM).

4 Security Analysis

4.1 Security Proofs of IND-CPA and IND-CCA2

We proved the IND-CPA security of the IND-CPA PKE version of LizarMong under the assumption of the IND-CPA security of RLizard.CPA [13].

Theorem 1. *The IND-CPA PKE version of LizarMong is IND-CPA secure under the hardness assumption of RLWE and RLWR problem for a given parameter, and the assumption that* `SHAKE256` *is a random oracle model.*

Proof. Note that in this proof, we call the IND-CPA PKE version of LizarMong as LizarMong. An encryption of m can be generated from an encryption of zero by the homomorphic property of LizarMong. Hence, it is enough to show that the pair of public information pk and the encryption of zero is computationally indistinguishable from the uniform distribution. Let $LizarMong'$ be an algorithm that is the same as LizarMong except for the ciphertext compression. First, we

Algorithm 7 KeyGen'

Input: The set of public parameters
Output: Public key $pk' = (\mathbf{a}, \mathbf{b})$
1: $pk = (Seed_a, \mathbf{b}) \leftarrow LizarMong.KeyGen$
2: $\mathbf{a} \leftarrow \texttt{SHAKE256}(Seed_a, n)$
3: $pk' \leftarrow (\mathbf{a}, \mathbf{b})$
4: **return** pk'

show that $LizarMong'$ is IND-CPA secure. Define $KeyGen'$ as Algorithm 7. Define distribution D_0, D_1, and D_2 as followings:

$$\begin{aligned} D_0 = \{(pk', C) :& pk' \leftarrow KeyGen'(params), \\ & C = (c_1, c_2) \leftarrow LizarMong.Enc_p k(0)\} \\ D_1 = \{(pk, C) :& pk \leftarrow RLizard.KeyGen(params), \\ & C = (c_1, c_2) \leftarrow RLizard.Enc_p k(ecc(0))\} \\ D_2 = \{(pk, C) :& pk \leftarrow Ring, \\ & C = (c_1, c_2) \leftarrow Ring\} \end{aligned}$$

Since `SHAKE` is a random oracle model, distribution of pk' and pk are computationally indistinguishable. $C_{LizarMong'} \leftarrow LizarMong'.Enc_p k(0)$ for $pk = (Seed_a, b)$ and $C_{RLizard} \leftarrow RLizard.Enc_p k'(ecc(0))$ for $pk' = (\texttt{SHAKE256}(Seed_a, n), b)$ are same by the definition of $RLizard$ and $LizarMong'$ (i.e. $C_{LizarMong} = C_{RLizard}$). Thus, D_0 and D_1 are computationally indistinguishable.

Lemma 1 (See [13]). *RLizard.CPA is IND-CPA secure under the hardness assumption of RLWE and RLWR problem for a given parameter.*

By Lemma 1, D_1 and D_2 are computationally indistinguishable. Therefore, D_0 and D_2 are computationally indistinguishable. In conclusion, $LizarMong'$ IND-CPA secure. Since ciphertext compression does not affect the security of LWE and LWR based public-key cryptography [24], $LizarMong$ is IND-CPA secure. □

By Theorem 1, IND-CPA PKE version of LizarMong is IND-CPA secure PKE. We make IND-CCA2 KEM version of LizarMong by using Jiang et al. transformation [21]. Thus IND-CCA2 KEM version of LizarMong is IND-CCA2 secure.

4.2 Security Analysis against Known Attacks

Our security analysis is based on the pessimistic approach of the BKZ lattice basis reduction algorithm [5]. Also, we use the attack complexity calculation and the online LWE estimator by Albrecht et al. [3,4]. Those are common methods used by most RLWE family algorithms. The BKZ algorithm proceeds by reducing a lattice basis using the SVP oracle repeatedly. There are several discussions about measuring the number of iterations. The core SVP method ignores repeated calls for SVP oracle, which is a pessimistic estimation from the defender point of view. We use the quantum sieve as the SVP oracle, which is also a pessimistic approach [5]. The computational complexity of the BKZ lattice basis reduction algorithm is 2^{cn}, where n is a dimension of lattice, and c is a constant value such that $c = 0.292$ in the classical environment and $c = 0.265$ in the quantum environment.

We considered the attack on the RLWE family studied in [3] and the specific attack on the sparse ternary secret in [2]. The computational complexity for RLWR attacks is the same as RLWE attacks with the same dimensions, same RLWE modulus q, and error rates $p^{-1}\sqrt{\pi}/6$ [13]. Hence, the computational complexities for RLWR attacks are calculated similarly to the RLWE attacks. The online LWE estimator helped the complexity calculation for these attacks. The Python code for calculating computational complexity can be found at https://github.com/LizarMong.

Table 3: Computational complexity of best RLWE and RLWR attacks

Parameters	**Claim Security**	**Attacks**		**Classical**	**Quantum**
`Comfort`	NIST Category 1 (AES 128-bit)	Primal	RLWE	**133**	**121**
			RLWR	144	131
		Dual	RLWE	165	154
			RLWR	180	170
`Strong`	NIST Category 5 (AES 256-bit)	Primal	RLWE	**256**	**236**
			RLWR	269	249
		Dual	RLWE	304	275
			RLWR	328	301

We concluded that the primal attack [2] that uses the BKZ algorithm is the best. Table 3 shows the computational complexities of the best attacks. In summary, LizarMong.`Comfort` satisfies NIST's category1 security level (AES128), and `Strong` satisfies category5 security level (AES256).

In the `Comfort` parameters, our security is overshoots with the requirements of the security level. It is a security margin that we knowingly made. Attacks against RLWE and RLWR have not been enough studied yet, so the security margin prepares for unknown and vital attacks. In `Strong` parameters, on the other hand, it has no security margin. Since the 256-bit security level is very high, the NIST standardization process focuses on up to the 192-bit security level. So, the `Strong` parameter is robust in itself.

4.3 Correctness Analysis

The failure probability calculations of the RLWE family designed so far have been analyzed on the assumption that errors in each bit occur independently. However, D′Anvers et al. proved [16] theoretically and experimentally that the error between each bit does not occur independently. According to D′Anvers et al. [16], even if the probability of error occurrence between each bit is not independent, the calculation based on the independence assumption is valid when the error correction code is not used. However, It is inappropriate when the error correction code is used.

Since LizarMong uses the error correction code, we calculate the probability of failure under the assumption that the error of each bit occurs dependently [16]. Cheon et al. showed that RLizard decryption fails when $|\mathbf{e}*\mathbf{r}+\mathbf{s}*\mathbf{f}| \geq \frac{q}{4}-\frac{q}{2p}$ where $\mathbf{f} = \mathbf{a}*\mathbf{r}-(q/p)\cdot\mathbf{c_1}$ in [14]. Because of ciphertext compression, LizarMong has more errors than RLizard. That is the difference between $\mathbf{c_2} := \lfloor (k/q)\cdot((q/2)\cdot \mathbf{M'}+\mathbf{b}*\mathbf{r}\rceil$ and $\hat{\mathbf{c}}_\mathbf{2} := (p/k)\cdot\mathbf{c_2}$. Hence, decryption failure of LizarMong occurs when $|\mathbf{e}*\mathbf{r}+\mathbf{s}*\mathbf{f}+\mathbf{g}| \geq \frac{q}{4}-\frac{q}{2p}$ where $\mathbf{g} = \mathbf{c_2}-\hat{\mathbf{c}}_\mathbf{2}$. We define $S = (\mathbf{s},\mathbf{e})^T$, $C = (\mathbf{f},\mathbf{r})^T$ to calculate the probability of decryption failure. On the assumption that the error of each bit occurs dependently, the probability of decryption failure is calculated according to the equation (1). Note that $\Pr[Fail]$ is the probability of decryption failure, $\Pr[F_i]$ is the probability that an error occurs in the ith bit, $Binom(k,n,p) = \sum_{i=0}^{k}\binom{n}{i}p^i(1-p)^{n-i}$, $p_b = \Pr[F_0 \mid \|S\|, \|C\|]$, l_m is the length of encoded message, and d is error correcting capability. Since we use XE5 as an error correction code, $d = 5$.

$$\Pr[Fail] \approx \sum_{\|S\|,\|C\|} (1 - Binom(d, l_m, p_b)) \cdot \Pr[\|S\|] \cdot \Pr[\|C\|] \tag{1}$$

We can calculate $p_b = \Pr[F_0 \mid \|S\|, \|C\|]$ as equation (2) by [15].

$$p_b = \sum_{l}\sum_{g_0}(\Pr[|C^TS+\mathbf{g}|_0 > q/4 - q/2p \mid |C^TSs|_0 = l, \mathbf{g}_0]\cdot \Pr[|C^TS|_0 = l \mid \|S\|, \|C\|] \cdot \Pr[\mathbf{g}_0]) \tag{2}$$

The above calculation process is implemented in Python and can be found at https://github.com/LizarMong. According to our calculation, the decryption failure probability of LizarMong is 2^{-179} in the `Comfort` and 2^{-302} in the `Strong`.

4.4 Side-Channel Attacks

The strategy for making LizarMong resistant to known side-channel attacks is as follows. First, we ruled out the operations targeted by the known attacks at the design element selection stage, Section 3.1. Second, for unavoidable vulnerabilities, we added a strategy that internalizes efficient countermeasures.

According to the first strategy, LizarMong resists known cache and timing attacks, as well as some differential and fault attacks. The timing attack of [26] performs the attack by using the time difference depending on whether the modulus is operating or not. This attack does not apply to LizarMong, however, because LizarMong uses all of the moduli to the power of two to replace the modulo operation with ADD and AND operations. Moreover, LizarMong does not use the CDT technique in order to resist the timing attack by Kim et al. [22] and the cache attack of [11], which exploits the CDT technique used by RLizard. The fault attack of [29], which attacks the situation of sampling $\mathbf{s}$ and $\mathbf{e}$ within the same distribution, does not apply to LizarMong, which is designed to sample $\mathbf{s}$ and $\mathbf{e}$ within each distribution. Also, the differential attack of [28] targeting NTT does not apply to LizarMong, which does not use NTT. Despite efforts to minimize attack points at the design element selection stage, some differential attacks and fault attacks are still applicable, as shown in Table 4. Therefore, according to the second strategy, we added countermeasures against the remaining attacks in our scheme.

Differential Attacks [7] and [20] used polynomial multiplication between public and secret keys as the point of attack. Since the polynomial multiplication is necessary for the RLWE family algorithms, it is necessary to design additional countermeasures. Known countermeasures include masking [25,30,31] and hiding schemes [10]. Masking schemes include a general method of construction using random values and a decoder and a unique method of using a homomorphism under the addition of the RLWE family. Hiding schemes include shuffling the order of multiplication operations or adding dummy operations between real operations. In the RLWE family, masking methods such as masked decoders or additively homomorphic masking are relatively expensive. Thus, we devised

Table 4: Known side-channel attacks and countermeasures of LizarMong

Attack methods	**Attacks**	**Attack Points**	**Countermeasures**
Differential Attacks	[7]	Multiplication	hiding scheme
	[20]		
Template Attacks	[10]		
Fault Attacks	[17]	Error sampling	Loop index check

Algorithm 8 Sparse Polynomial Multiplication with Hiding Countermeasure

Input: $\mathbf{a} = \sum_{i=0}^{n-1} a(i) \cdot x^i \in R_q$, $\mathbf{r} = \sum_{i=0}^{n-1} r(i) \cdot x^i \in HWT_n(h)$,
$d = [i_0, \ldots, i_{g-1}, i_g, \ldots, i_{h-1}]$ with $d[k] = i_k$ such that $r(i_k) = 1$ for $k \in [0, g)$ and $r(i_k) = -1$ for $k \in [g, h)$
Output: $\mathbf{v} = \mathbf{a} * \mathbf{r} = \sum_{i=0}^{n-1} v(i) \cdot x^i \in R_q$

1: initialize $\mathbf{v}$ to zero polynomial ▷ size of $\mathbf{v} = 2n$
2: $m \xleftarrow{\$} \{0, 1, \ldots, h-1\}$ ▷ random starting index
3: **for** $i \in \{0, \ldots, h-1\}$, $j \in \{0, \ldots, n-1\}$ **do**
4: **if** $m + i \pmod h < g$ **then**
5: $v(d[m + i \pmod h] + j) = v(d[m + i \pmod h] + j) + a(j)$
6: **else**
7: $v(d[m + i \pmod h] + j) = v(d[m + i \pmod h] + j) - a(j)$
8: **for** $i \in \{0, \ldots, n-1\}$ **do**
9: $v(i) = v(i) - v(n + i)$
10: **return** $\mathbf{v}$

the sparse polynomial multiplication with the hiding scheme, like Algorithm 8. It combines the fast sparse polynomial multiplication algorithm of [1] with the hiding scheme of [10]. This method has fewer overheads than shuffling tasks since that it uses only one random value.

Fault Attacks Known fault attacks targeting the RLWE family exploit the process of generating $\mathbf{s}$ and $\mathbf{e}$. $\mathbf{s}$ and $\mathbf{e}$ are generated by loops after extracting random values. [17] frustrates the loop by injecting a fault and makes $\mathbf{s}$ and $\mathbf{e}$ the initial values of zero. LizarMong is vulnerable to this attack because it generates $\mathbf{s}$ and $\mathbf{e}$ using the above method. Therefore, our scheme resists the fault attack of [17] by applying statistical tests of [19]. The statistical test we use consists of a straightforward operation that compares the expected index with the final index after the loop statement is done, so there is negligible overhead.

5 Evaluation

We evaluate security (computational complexity), correctness (failure probability of decryption), bandwidth (size of ciphertext and public-key), and performance (CPU cycle of encryption, decryption, and key-generation) in comparison with NIST's candidate algorithms and RLizard.

Our comparison is based on the NIST official documents. The result of evaluation is shown in Table 5 and Fig. 1. In Table 5, the three rows for each algorithm correspond to 128, 192, and 256-bit security levels. (i.e., our scheme and NewHope do not support the 192-bit security level.)

Note that the performance evaluation used each optimization code, and the evaluation environment is Intel i7-9700K@3.2GHz CPU, ubuntu 16.04.11, GCC 5.4.0 with option $-$O3, and the value is the average for 1000 iterations. Also, our implementation codes are available to https://github.com/LizarMong.

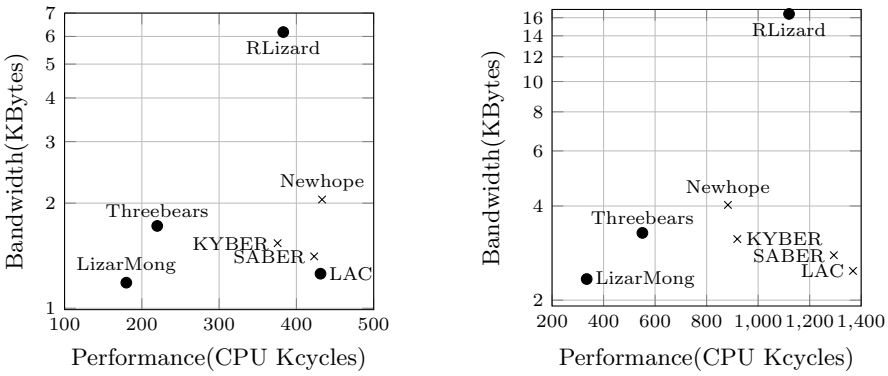

Fig. 1: Comparison of bandwidth and performance based on IND-CCA2 KEM. (left) 128-bit security level (right) 256-bit security level (Note: • are algorithms with security and correctness similar to each security level, and × are not.)

Security The security of some algorithms seems to be slightly lacking to each security level (see Table 5). However, LizarMong.`Comfort` reaches 128-bit security level and LizarMong.`Strong` reaches 256-bit security level. In the `Comfort`, the security overshoots the requirements of the security level. It is a security margin that we knowingly made from a conservative perspective. `Strong` has no security margin because the 256-bit security level is regarded as very highly. Unfortunately, our scheme does not support the 192-bit security level, but `Strong` has a competitive bandwidth and performance compared with the 192-bit security level of other algorithms. Therefore, `Strong` can sufficiently replace the 192-bit security level of other algorithms.

Correctness KYBER, SABER, LAC, and Round5 have smaller failure probability of decryption compared to the mapped security level. However, LizarMong has negligible failure probability such as 2^{-179} in `Comfort` and 2^{-302} in `Strong`. Also, our estimation is accurate than others because we consider dependency.

Bandwidth Bandwidth is one of the significant determinants of algorithm practicality in resource-constrained devices and poor communication environments. In general, because the bandwidth of the RLWE family has a larger bandwidth than the current public-key cryptography such as RSA and ECC, the evaluation of bandwidth is a critical evaluation criterion. LizarMong is the best among the key encapsulation mechanisms supporting IND-CCA2. `Comfort` and `Strong` are smaller about 5% compared with LAC (which is ranked second in bandwidth).

Performance LizarMong has the best performance among NIST's candidate algorithms and RLizard. `Comfort` and `Strong` faster 1.25 times and 1.65 times than ThreeBears that is ranked second in performance. The crucial point is that the recorded performance of LizarMong includes all countermeasures of the known side-channel attacks in Section 4.4.

Table 5: Comparison KEM with NIST candidate algorithms and RLizard

Algorithms	**Security** (log)	**Correctness** (log)	**Bandwidth** (Bytes)	**Performance** (K cycles)	
				Enc+Dec	**KeyGen**
LizarMong	133	−179	1,184	137.5	42.4
	256	−302	2,336	272.7	61.8
RLizard	147	−188	6,176	217.8	165.3
	195	−246	8,240	416.9	232.7
	318	−306	16,448	737.3	382.7
NewHope	112	−213	2,048	329.6	103.6
	257	−216	4,032	673.5	209.2
KYBER	111	−178	1,536	278.2	97.5
	181	−164	2,272	463.6	174.3
	254	−174	3,136	656.0	263.1
SABER	125	−120	1,408	316.9	106.1
	203	−136	2,080	587.6	213.6
	283	−165	2,784	934.8	359.2
LAC	147	−116	1,256	341.2	90.0
	286	−143	2,244	840.1	235.6
	320	−122	2,480	1,101.6	266.6
Round5 (IND-CPA)	128	−88	994	384.4	114.6
	193	−117	1,639	857.2	311.3
	256	−64	2,035	1,794.9	643.4
Threebears	154	−156	1,721	167.8	52.1
	235	−206	2,501	271.4	91.9
	314	−256	3,281	402.5	148.2

6 Conclusion

Our scheme, called LizarMong, is the best of the RLWE family of key encapsulation algorithms to date. Our scheme achieves security levels 1 (128-bit) and 5 (256-bit), and compared with NIST's candidate algorithms, the bandwidth is about 5-42% smaller, and the performance is about 1.2-4.1 times faster. Also, it resists known side-channel attacks.

We need to recall the goal of the NIST post-quantum cryptography standardization process. The purpose of this process is to design cryptography that is compatible with current networks and protocols. Thus, there is a need for algorithms that are excellent in all respects, such as RSA and ECC. The RLWE family algorithms submitted to the NIST post-quantum cryptography standardization process have each merit in terms of security, correctness, performance, and bandwidth. Thus, choosing one optimal algorithm satisfying all aspects is challenging. Besides, various recent studies have been published that affect security and correctness, such as side-channel attacks and error dependencies. These

studies are meaningful because it affects most RLWE family algorithms. To date, however, many algorithms do not include most of those studies.

We consider to break down the barriers between candidate algorithms, merging unique strengths, and quickly reflecting the state-of-the-art studies, for excellent algorithm in all respects. LizarMong, based on the RLizard that was submitted to NIST 1round, muses each of the merit of NIST's candidate algorithms. Specifically, we have inspired by a small modulus of LAC, the error correction code of Round5, and the centered binomial distribution of NewHope. We also included recent studies such as side-channel attacks and error dependencies.

In conclusion, our scheme is an excellent key encapsulation mechanism that combines each merit of NISTs candidate algorithms with state-of-the-art studies.

Acknowledgements.

We would like to thank anonymous reviews of ICISC 2019 and Jung Hee Cheon for their helpful comments and suggestions. This work was supported as part of the Military Crypto Research Center (UD170109ED) funded by Defense the Acquisition Program Administration (DAPA) and the Agency for Defense Development (ADD).

References

1. Akleylek, S., Alkım, E., Tok, Z.Y.: Sparse polynomial multiplication for lattice-based cryptography with small complexity. The Journal of Supercomputing **72**(2), 438–450 (2016)
2. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving usvp and applications to lwe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 297–322. Springer (2017)
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
4. Albrecht, M.: A sage module for estimating the concrete security of learning with errors instances (2017)
5. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchangea new hope. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 327–343 (2016)
6. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Annual International Cryptology Conference. pp. 595–618. Springer (2009)
7. Aysu, A., Tobah, Y., Tiwari, M., Gerstlauer, A., Orshansky, M.: Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 81–88. IEEE (2018)
8. Baan, H., Bhattacharya, S., Fluhrer, S.R., Garcia-Morchon, O., Laarhoven, T., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Zhang, Z.: Round5: Compact and fast post-quantum public-key encryption. IACR Cryptology ePrint Archive **2019**, 90 (2019)

9. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
10. Bos, J.W., Friedberger, S., Martinoli, M., Oswald, E., Stam, M.: Assessing the feasibility of single trace power analysis of frodo. In: International Conference on Selected Areas in Cryptography. pp. 216–234. Springer (2018)
11. Bruinderink, L.G., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload–a cache attack on the bliss lattice-based signature scheme. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 323–345. Springer (2016)
12. Chen, L., Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. US Department of Commerce, National Institute of Standards and Technology (2016)
13. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: Cut off the tail! practical post-quantum public-key encryption from lwe and lwr. Cryptology ePrint Archive, Report 2016/1126 (2016), https://eprint.iacr.org/2016/1126
14. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard public key encryption. Tech. rep., Technical report, National Institute of Standards and Technology, 2017 (2018)
15. D′Anvers, J.P., Vercauteren, F., Verbauwhede, I.: On the impact of decryption failures on the security of lwe/lwr based schemes. Cryptology ePrint Archive, Report 2018/1089 (2018), https://eprint.iacr.org/2018/1089
16. DAnvers, J.P., Vercauteren, F., Verbauwhede, I.: The impact of error dependencies on ring/mod-lwe/lwr based schemes. In: International Conference on Post-Quantum Cryptography. pp. 103–115. Springer (2019)
17. Espitau, T., Fouque, P.A., Gerard, B., Tibouchi, M.: Loop-abort faults on lattice-based signature schemes and key exchange protocols. IEEE Transactions on Computers **67**(11), 1535–1549 (2018)
18. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Theory of Cryptography Conference. pp. 341–371. Springer (2017)
19. Howe, J., Khalid, A., Martinoli, M., Regazzoni, F., Oswald, E.: Fault attack countermeasures for error samplers in lattice-based cryptography. In: 2019 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5. IEEE (2019)
20. Huang, W.L., Chen, J.P., Yang, B.Y.: Correlation power analysis on ntru prime and related countermeasures. IACR Cryptology ePrint Archive **2019**, 100 (2019)
21. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Annual International Cryptology Conference. pp. 96–125. Springer (2018)
22. Kim, S., Hong, S.: Single trace analysis on constant time cdt sampler and its countermeasure. Applied Sciences **8**(10), 1809 (2018)
23. Lee, J., Kim, D., Lee, H., Lee, Y., Cheon, J.H.: Rlizard: Post-quantum key encapsulation mechanism for iot devices. IEEE Access **7**, 2080–2091 (2018)
24. Lu, X., Liu, Y., Zhang, Z., Jia, D., Xue, H., He, J., Li, B., Wang, K., Liu, Z., Yang, H.: Lac: Practical ring-lwe based public-key encryption with byte-level modulus. IACR Cryptology ePrint Archive **2018**, 1009 (2018)
25. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 142–174 (2018)

26. Park, A., Han, D.G.: Chosen ciphertext simple power analysis on software 8-bit implementation of ring-lwe encryption. In: 2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST). pp. 1–6. IEEE (2016)
27. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of ring-lwe for any ring and modulus. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 461–473. ACM (2017)
28. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 513–533. Springer (2017)
29. Ravi, P., Roy, D.B., Bhasin, S., Chattopadhyay, A., Mukhopadhyay, D.: Number not used once-practical fault attack on pqm4 implementations of nist candidates. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 232–250. Springer (2019)
30. Reparaz, O., de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Additively homomorphic ring-lwe masking. In: Post-Quantum Cryptography. pp. 233–244. Springer (2016)
31. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A masked ring-lwe implementation. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 683–702. Springer (2015)
32. Saarinen, M.J.O.: Hila5: On reliability, reconciliation, and error correction for ring-lwe encryption. Cryptology ePrint Archive, Report 2017/424 (2017), https://eprint.iacr.org/2017/424