J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

Improving the speed of the Lizard implementation

Shakhriddin Rustamov[1] Younho Lee[2*]

ABSTRACT

Along with the recent advances in quantum computers, it is anticipated that cryptographic attacks using them will make it insecure to use existing public key algorithms such as RSA and ECC. Currently, a lot of researches are underway to replace them by devising PQC (Post Quantum Cryptography) schemes. In this paper, we propose a performance enhancement method for Lizard implementation which is one of NIST PQC standardization submission. The proposed method is able to improve the performance by 7 ~ 25% for its algorithms compared to the implementation in the submission through the techniques of various implementation aspects. This study hopes that Lizard will become more competitive as a candidate for PQC standardization.

☞ keyword : Efficient implementation, Lizard, Post-quantum Cryptography, Cryptography, Security

1. Introduction

Post quantum cryptography refers to the cryptographic algorithms to maintain the security against attacks based on quantum computers. In 2017, NIST recruited candidates for standardization of the post quantum cryptography method and 69 candidates were proposed. In 2022, the selection of standard methods will be completed [1]. In this study, we aim to improve the implementation performance of the proposed Lizard [2] method among these candidates. The Lizardmethod is based on the Learning With Errors (LWE) problem [3] and the Learning With Rounding (LWR) problem [4]. We can implement many security services using Lizard, but we try to improve the implementation of the simplest service, KEM (Key Encapsulation Mechanism). KEM means a method by which two participating communicators can securely generate a shared key of a can for secure communication. Normally, if A and B use KEM to generate a shared key, then A sends its public key and public parameters to B, andB generates and passes the ciphertext c passed to A using KEM's Key Encapsulation algorithm do. B also obtains K, the key information to be shared at the same time. Finally, A uses the key decapsulation algorithm to obtain the shared key K using c and its own private key.

We have improved the performance of these KEM implementations. Specifically, we reduced the number of calls to the random number generation function used in key generation. This can be achieved by efficiently using the generated random number bits. Key Encapsulation algorithm and Decapsulation algorithm also improve the execution speed of iterative operations by using registers, and also improve the implementation of pointer operation using multiplication only by addition. As a result of this improvement, we achieved about 25% of key generation, 7% of key encapsulation, and 10% of key decapsulation, compared to the KEM version originally submitted to NIST.

The rest of this paper is organized as follows. Section 2 deals with prior knowledge, and Section 3 deals with related research. In Section 4, we propose an improvement method. In Section 5, we perform performance evaluation. Conclusions are made in Section 6.

1 Division of Industrial and Information Systems Engineering, The Graduate School of Public Policy and Information Technology, Seoul National University of Science and Technology, Seoul, 01811, Korea
2 ITM Division, Seoul National University of Science and Technology, Seoul, 01811, Korea
* Corresponding author: younholee@seoultech.ac.kr
[Received 30 December 2018, Reviewed 23 January 2019(R2 5 March 2019), Accepted 27 May 2019]
☆ This study was supported by the Research Program funded by the Seoul National University of Science and Technology.
☆ A preliminary version of this paper was presented at APIC-IST 2018 and was selected as an outstanding paper.

2. Preliminaries

2.1. Notation

Here is the list of notations which will be used throughout this paper:

(Table 1) Notations

Notation	Description
m	the number of LWE samples, a positive integer, a power of two
n	the dimension of LWE samples, a positive integer
ℓ	a positive integer, the number of secret vectors in case of Lizard primitive, i.e. the number of plaintext slots in case of Lizard primitive
ℓ1	a positive integer, the number of secret vectors in case of IND-CPA KEM schemes
ℓ2	a positive integer, the number of ephemeral secret vectors in IND-CPA KEM schemes
d	a positive integer, the number of plaintext slots in case of IND-CPA PKE, the bit-length of shared secret key in case of IND-CPA KEM
p	the small modulus for rounding, a positive integer, a power of two
q	the large modulus, a positive integer, a power of two
hr	the Hamming weight of an ephemeral secret vector r
α	Lizard only considers the discrete Gaussian $\chi = DG_{\alpha}$ as an error distribution where α is the error rate in (0, 1), so α will substitute the distribution χ.
G, H, H′	three hash functions to achieve the IND-CCA2 security
Z_q	a set {0, 1, ..., q − 1}
Z_p	a set {0, 1, ..., p − 1}
$HWT_m(h)$	the uniform distribution over the subset of {−1, 0, 1}m whose elements contain m − h number of zeros
$B_{m,h}$	the subset of {−1, 0, 1}m of which elements have exactly h number of non-zero components, i.e. the set of all possible vectors chosen from
R	the ring $Z[X]/(X^n+1)$
A^t	the transpose of the matrix A
$ZO_n(\rho)$	the distribution over {−, 0, 1}n where each component x satisfies Pr[x= 1] =Pr[x= −] = ρ/2 and Pr[x= 0] = 1 − ρ
(ρ)	1/2
DGσ	the discrete Gaussian distribution with the parameter σ
⌊⌉	rounding function, ⌊⌉ is the nearest integer to the rational number x, rounding upwards in case of a tie
‖	concatenation operator
‖ · ‖	norm operator
LWE_N	refers to the notation n, in which the number of LWE samples, positive integer and power of two are set
LWE_M	refers to the notation m, in which the number of LWE samples, positive integer and power of two are set
LWE_L	refers to the notation ℓ, in which positive integer and the number of plaintext slots in case of Lizard primitive are set
LWE_L1	refers to the notation ℓ1, in which positive integer and the number of secret vectors in case of IND-CCA2 KEM schemes are set
LWE_L2	refers to the notation ℓ2, in which positive integer and the number of ephemeral secret vectors in IND-CCA2 KEM schemes are set

The Lizard implementation in this paper uses the parameter KEM_CATEGORY1_N536 [1], which provides the same level of security as that provided by 128bit AES. In this case, the values of some of the symbols defined above are defined as shown in Table 2.

(Table 2) KEM_CATEGORY1_N536 parameter setup

Notation	Value
LWE_N (n)	536
LWE_M(m)	1024
LWE_L(ℓ)	256
LWE_L1 (ℓ1)	16
LWE_L2 (ℓ2)	16
HR	140

2.2 Introduction to Lizard KEM

Recently, Hofheinz et al. proposed a method of converting a public key cryptographic algorithm of arbitrary IND-CPA

security, to a KEM that provides IND-CCA2 security by using FO (Fusisaki-Okamoto) transformation [5]. The Lizard KEM is a result of modifying the Lizard cipher algorithm according to the above method, and can be described as Tables 3 to 5 below.

Lizard is composed of the key generation algorithm (Lizard.KeyGen), the key encapsulation algorithm (Lizard.Encap) and the key decapsulation algorithm (Lizard.Decap). the parameters used for Lizard.KeyGen (params) are m, n, l1, l2, l, d, p, q, (2|p|q), hr (<m), l=l1 • l2, and also include 0 < ρ α<1, and the hash functions G:{0,1}*→,1}d, H:{0,1}* →l2m,h, and H':{0,1}*→,1}l. The following Table 3~5 describe the details of the algorithms.

(Table 3) Lizard.KeyGen

Lizard.Keygen	
Input	The set of parameters params
Output	A key pair containing the private key (S, T) ∈ $\{-, 0, 1\}^{n\times \ell 1}\times\{0, 1\}^{\ell 1\times \ell 2}$ and the public key (A‖B)∈$Z_q^{m\times(n+\ell 1)}$.
Operation	
Step 1	Generate a random matrix A←$Z_q^{m\times n}$.
Step 2	Set a secret matrix S: = (s0‖..‖$s_{\ell 1-}$) by sampling each si independently from the distribution $ZO_n(1/2)$.
Step 3	Generate a random matrix T ← $\{0, 1\}^{\ell 1\times \ell 2}$.
Step 4	Generate error matrix E. Sample an integer E ← $DG_{\alpha q}$, and then set E = (E) ∈ $Zq^{m\times \ell 1}$.
Step 5	Calculate B: = −S + E ∈ Zqm× ℓ1. It is the second part of the public key. (A,B) is the public key, (S,T) is the secret (private) key.

(Table 4) Lizard.Encap

Lizard.Encap	
Input	The set of public parameters params, public key pk = (A‖B)∈$Z_q^{m\times n}\times Z_q^{m\times \ell 1}$.
Output	The ciphertext C = (C1, C2, d) ∈ $Z_p^{n\times \ell 2}\times Z_p^{\ell 1\times \ell 2}$ × $\{0, 1\}^{\ell}$ and the shared key K ∈ $\{0, 1\}^{d}$.
Operation	
Step 1	Generate a random matrix M ∈ $\{0,1\}^{\ell 1\times \ell 2}$.
Step 2	Compute the matrix R:= H(M), create vector r_{idx} containing indices of non-zero components of R, and the vector H'(M). The logic behind this is in the description [8].
Step 3	Extract matrices A and B from the public key.
Step 4	Compute C1:=⌊p/q) · A^tR⌉ ∈ $Z_p^{n\times \ell 2}$ and C2:=⌊p/q) · ((q/2) · M + B^tR)⌉ ∈$Z_p^{\ell 1\times \ell 2}$. C1 and C2 are the first and second vectors.
Step 5	Compute K: = G (C1, C2, d, M), and output the pair (C = (C1, C2, d), K). M is the source text. K is the encapsulated text. Calculate shared secret which is a specific hash function of the plain and encrypted text.

(Table 5) Lizard.Decap

Lizard.Decap	
Input:	The set of public parameters params, the public key pk = (A ‖ B) ∈ $Z_q^{m\times n}$ × $Z_q^{m\times \ell 1}$, the secret key sk = (S, T) ∈$\{-1, 0, 1\}^{n\times \ell 1}$ × $\{0, 1\}^{\ell 1\times \ell 2}$, and the ciphertext C = (C1, C2, d) ∈ $Z_p^{n\times \ell 2}$ × $Z_p^{\ell 1\times \ell 2}\times \{0, 1\}^{\ell}$.
Output:	The shared key K ∈ $\{0, 1\}^{d}$.
Operation	
Step 1	Parse the ciphertext C: = (C1, C2, d). C1 and C2 are the first and second vectors in the encrypted text.
Step 2	Compute M′ := ⌊2/p) · (C2+S^tC1)⌉ ∈ $Z_2^{\ell 1\times \ell 2}$.
Step 3	Compute R′ := H (M′), r_{idx} from M‘and d′:= H'(M′).
Step 4	Compute C1′:= ⌊p/q) · A^tR'⌉ ∈ $Z_p^{n\times \ell 2}$ and C2′:= ⌊p/q) · ((q/2) · M′+ B^tR')⌉ ∈ $Z_p^{\ell 1\times \ell 2}$, and set C′:= (C1′, C2′, d′).
Step 5	If C ≠′, then output K:= G (C1, C2, d, T).
Step 6	Else, output the shared key K:= G (C1, C2, d, M′). Calculate the shared secret of the decapsulated text and compare it to the one passed along with the encrypted text.

The algorithms described in Tables 3-5 take a lot of time unless carefully implemented [1]. We aim to implement the Lizard submitted to NIST as an implementation to be improved [1].

3. Related Work

This paper is an improvement of Lizard proposed to NIST. In this paper, we introduce the competitive PQC algorithms proposed to NIST, and finally introduce Lizard.

Bos et al. proposed a KEM (Frodo) that is based LWE problem [6]. They showed that they could implement their methods in OpenSSL to share keys between the participants in networks. They argued that they could use their methods to provide quantum security so they could replace ECDHE. This method is a KEM competing with Lizard. The problem with this method is that it takes a lot of randomness to

generate matrix A and consumes 40% of the time to generate A [6]. They have attempted to use their methods in a variety of environments through further improvements in the pattern of memory accesses.

Cheon et al. proposed a public key cryptographic algorithm based on spLWE (LWE with sparse secret), which is a variant of the LWE problem [7]. They implemented it based onPeikert's IND-CPA-based public key cryptosystem [8] with FO transform [9]. This method has a disadvantage in that a higher-order parameter is required to be used in comparison with the LWE-based method. This method requires about 313 microseconds in the Macbook Pro with 2.6 GHz Intel Core i5 CPU environment to share the message of 256 bits length.

Alkim et al. proposed a method to improve efficiency and safety compared to [6] by improving the method proposed by Peikert et al. [8] [10]. In their method, the large modulus q value could be reduced to q = 12289 < 214 compared to the previous method [6] where q=232. It can also provide 128-bit post quantum safety.

Bernstein et al. [10] mentioned the possibility of side channel attack and proposed a method based on streamlined NTRU prime [11]. The proposed method is improved in performance compared to the previous method, and it is known that it has a small attack surface because it uses a ring with no structure.

[12] is a well-known KEM based on the module LWE problem- and is known to exhibit very efficient performance. They are known to show very good performance in terms of the required ciphertext size and computation efficiency by applying a compression method for small numbers. In [13], unlike the previous method, it is proposed a cost-free method to achieve QROM security without additional hash.

Finally, Lizard is a method based on LWE and Learning with Rounding (LWR) [2]. This method is very efficient because it does not perform Gaussian Sampling in encryption. Lizard also ensures quantum security under the QROM model.

4. Proposed approach

In this section, we discuss various performance enhancement methods proposed in this study. The target implementation to enhance in this paper is the reference implementation of the Lizard submitted to the NIST PQC Standardization Content [1].

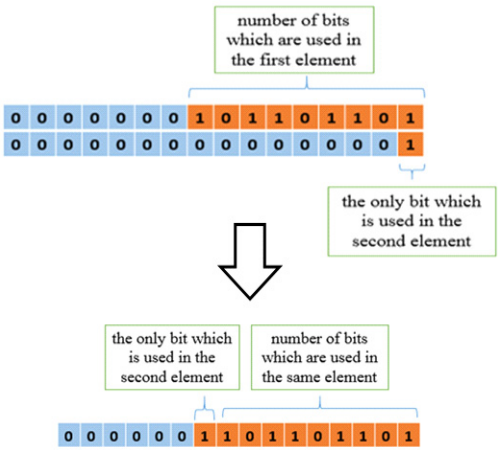
4.1 Improvement of key generation algorithm

We first reduced the number of random bits needed to generate matrix A. In the existing implementation, random bits are generated in byte units. Since the length of each element in matrix A is 11 bits, 5 bits are discarded when 2-byte random numbers are used to generate an element in A. This is done with a larger number of random number generator function calls, which requires a large performance waste. We used these 5 bits to reduce the number of random number generation.

According to the original implementation submitted by NIST [1], 1097728 random bits are required to generate a matrix A in 128bit quantum security setting. However, as a result of the improvement, only 823296 bits are required in the proposed implementation.

The second improvement is areduction in the number of random bits required to generate the secret key sparse vector S. In implementations submitted to the existing standard, a one-byte random number was used to generate S's elements, represented by one of 1,0, -1. We improved this to only use 2 bits, and as a result we could reduce the length of the required random bits to 1/4. As a result of the improvement, unlike the original implementation where 8576 random bits are required in the original implementation, the proposed implememtation requries only 2144 random bits to generate a secret key matrix S.

The third improvement is the reduction of the number of random bits needed to generate matrix E. The elements of this matrix are extracted from the Discrete Gaussian distribution, which requires random bits. Unfortunately, in the existing implementation, 32bit random number was generated and used even though total 10bit random number bit is needed. We have improved it so that only 10bit random number can be used exactly. Fig. 1) explains the improvement. As a result of this improvement the number of random bits generated for the matrix E is reduced from 65536bits to 32768bits.



(Fig. 1) Improvement to generate the matrix E: only one random word is generated and extracts 10bits from the word.

```
for (k = 0; k < LWE_L2; ++k) {
    size_t neg = 0, back_position = HR;
    uint8_t* r_t = r + k * LWE_M;
    uint16_t* r_idxt = r_idx + k * HR;
    for (j = 0; j < LWE_M; ++j) {
        if (r_t[j] == 0x01)
            r_idxt[neg++] = j;
        else if (r_t[j] == 0xff)
            r_idxt[--back_position] = j;
    }
    neg_start[k] = neg;
}
```

(Fig. 2) The initial implementation in the step of generation of r_idx in Key Encapsulation.

4.2 Improvement of key encapsulation and decapsulation algorithms

We propose an efficient method to generate r_idx by repeating the operation of extracting {-1,0,1} from the existing implementation. The concrete contents are shown in figures 2 and 3.

In the existing implementation, the multiplication operation is required when calculating two pointers r_t and r_idxtin the process of creating r_idx as shown in Fig. We have replaced the multiplication operations required for r_t by addition operations during these operations. If this process exists simultaneously in the encapsulation and decapsulation implementations, we modify it and improve the performance.

```
register size_t neg, back_position;
register uint8_t* r_t = r;
register uint16_t* r_idxt;
for (k = 0; k < LWE_L2; ++k) {
    neg = 0;
    back_position = HR;
    r_idxt = r_idx + k * HR;
    for (j = 0; j < LWE_M; ++j) {
        if (r_t[j] == 0x01)
            r_idxt[neg++] = j;
        else if (r_t[j] == 0xff)
            r_idxt[--back_position] = j;
    }
    neg_start[k] = neg;
    r_t += LWE_M;
}
```

(Fig. 3) Step of improved generation of r_idx in Key Encapsulation.

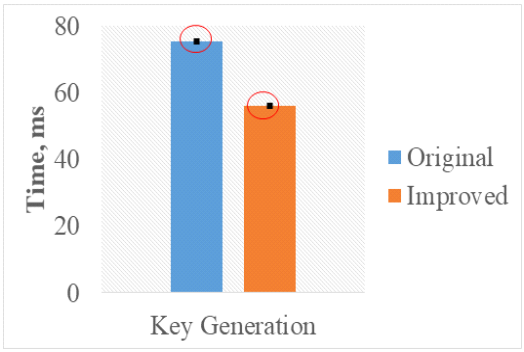
5. Performance evaluation

We performed 100,000 iterations for each improved algorithm implementation and measured the performance with the meanvalue. Performance evaluation was performed with parameters of 128 bit Quantum safety. The execution environment is Intel Core i5-4557 CPU 3.20GHz, 4GB RAM, and Ubuntu 14.04 LTS. As a result, we found that key generation is improved by 25%, Key Encapsulation and Decapsulaton by 7% and 10%, respectively, compared with the existing implementation.

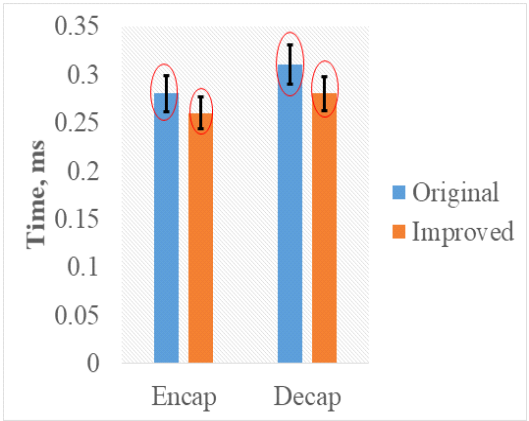
Figures 3 and 4 below show the execution time of each algorithm including the degree of change of execution time. As shown in the figure, the key generation time does not change at every execution, but the Key Encapsulation / Decapsulation method can confirm that there is a difference in execution time according to the generated random number value. The difference of the execution times vary in each iteration, around 0.4~0.55 ms. In order to improve the security of the Lizard, it is necessary to reduce this value to make the scheme resilient against side-channel attacks.

6. Conclusion

In this paper, we analyze the implementation of Lizard proposed in NIST PQC and proposeda method to improve performance. By simply reducing the number of wasted random bits and improving the register operation, we have achieved a performance improvement of 7% ~ 25%. We hope that the results of this paper will be applied to Lizard implementations so that Lizard can be adopted as an international standard algorithm. Also, as a future study, we will try to improve the performance of Ring Lizard based on Ring LWE problem unlike Lizard. This study is also expected to be applied as a key method in various security fields [14-16].



(Fig. 4) Comparison of Key generation time with its time



(Fig. 5) Comparison on Key Encapsulation/ Key Decapsulation times with its time variation

References

[1] NIST PQC Standardization. Available at: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[2] J. Cheon, D. Kim, J. Lee, and Y. Song, "Lizard: Cut Off the Tail! A Practical Post-quantum Public-Key Encryption from LWE and LWR", In Proc International Conference on Security and Cryptography for Networks (SCN) 2018, LNCS vol. 11035, pp. 160-177, 2018. https://doi.org/10.1007/978-3-319-98113-0_9

[3] Oded Regev, "On lattices, learning with errors, random linear codes, and cryptography", In Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, STOC '05, pages 84 - 93, New York, NY, USA, 2005. ACM. https://doi.org/10.1145/1568318.1568324

[4] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, Daniel Wich, "Learning with Rounding, Revisited", In Proc. CRYPTO, LNCS vol. 8042, pp. 57-74, 2013. https://doi.org/10.1007/978-3-642-40041-4_4

[5] Hofheinz, D., Hövelmanns, K., & Kiltz, E, "A modular analysis of the Fujisaki-Okamoto transformation,"In Theory of Cryptography Conference (pp. 341-371) Nov, 2017. Springer. https://doi.org/10.1007/978-3-319-70500-2_12

[6] Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., et al. "Frodo: Take off the Ring!: Practical, Quantum-Secure Key Exchange from LWE", In Proc. 23rd ACM Conference on Computer and Communications Security. https://doi.org/10.1145/2976749.2978425

[7] J Cheon et al, "A Practical Post-Quantum Public-Key Cryptosystem Based on spLWE.", In Proc. International Conference on Information Security and Cryptology (ICISC 2016) - LNCS vol. 10157, pp. 51-74, 2016. https://doi.org/10.1007/978-3-319-53177-9_3

[8] Peikert, C. "Lattice Cryptography for the Internet". In Proc. Post-Quantum Cryptography, LNCS vol. 8772, pp. 197-219. 2014. https://doi.org/10.1007/978-3-319-11659-4_12

[9] Targhi, E., & Unruh, D, "Quantum Security of the Fujisaki-Okamoto Transform", In Proc. Theory of

Cryptography:14th International Conference, 2016, pp. 192-216. Berlin: Springer. https://doi.org/10.1007/978-3-662-53644-5_8

[10] Alkim, E., Ducas, L., Pöppelman, T., & Shwabe, P, "Post-quantum Key Exchange – A New Hope," In, Proc.25th USENIX Security Symposium USENIX Security 16. Austin, TX: USENIX Association, 2016. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim

[11] Bernstein, D., Chuengsatiansup, C., Lange, T., & van Vredendaal, C., "NTRU Prime: Reducing Attack Surface at Low Cost", In Proc. Selected Areas in Cryptography SAC 2017, pp. 235-260, 2017. https://doi.org/10.1007/978-3-319-72565-9_12

[12] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., and Stehlé, D. (2018, April). CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In 3rd IEEE European Symposium on Security and Privacy. London, United Kingdom. https://doi.org/10.1109/eurosp.2018.00032

[13] Jiang, H., Zhang, Z., Chen, L., Wang, H., & Ma, Z., "Post-quantum IND-CCA-secure KEM without additional hash", Cryptology ePrint Archive, https://eprint.iacr.org/2017/1096 .

[14] Seul-Ki Choi, Chung-Huang Yang, Jin Kwak, " System Hardening and Security Monitoring for IoT Devices to Mitigate IoT Security Vulnerabilities and Threats", KSII Transactions on Internet and Information Systems, vol. 12, no. 2, Feb., 2018. https://doi.org/10.3837/tiis.2018.02.022

[15] Mrutyunjanya Sahani, Subhashree Subudhi, Mihir Narayan Mohanty, "Design of Face Recognition based Embedded Home Security System", KSII Transactions on Internet and Information Systems, vol. 10, no. 4, Apr., 2016. https://doi.org/10.3837/tiis.2016.04.016

[16] Admir Midzic, Zikrija Avdagic, Samir Omanovic, "Intrusion Detection System Modeling Based on Learning from Network Traffic Data", KSII Transactions on Internet and Information Systems, vol. 12, no. 11, Nov., 2018. https://doi.org/10.3837/tiis.2018.11.022

◐ 저 자 소 개 ◑

Shakhriddin Rustamov

Shakhridin Rustamov graduated TUIT at 2015. He obtained his M.S. degree from the Department of Industrial and Information Systems, Graduate School of Seoul National University of Science and Technology at 2018. Since 2018, he is working in Mongterang International Inc as an Application Specialist.

Younho Lee

YOUNHO LEE received the B.S., M.S., and Ph.D. degrees in computer science from KAIST, South Korea, in 2000, 2002, and 2006, respectively. He was a Visiting Post-Doctoral Researcher and as a Member of Research Staff with the Georgia Tech Information Security Center from 2007 to 2009. From 2009 to 2013, he was an Assistant Professor with the Department of Information and Communication Engineering, Yeungnam University, South Korea. He is currently an Associate Professor with the Department Industrial and Systems Engineering, Seoul National University of Science and Technology, South Korea. His research interests include network security, applied cryptography, and fintech security.