Ph.D. DISSERTATION

Homomorphic Computation in Reed-Muller Codes and Improvements of *Modified pqsigRM*

Reed-Muller 부호의 동형 연산과 *Modified pqsigRM*의 개선

BY

CHO JINKYU

AUGUST 2023

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Homomorphic Computation in Reed-Muller Codes and Improvements of *Modified pqsigRM*

Reed-Muller 부호의 동형 연산과 *Modified pqsigRM*의 개선

BY

CHO JINKYU

AUGUST 2023

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Homomorphic Computation in Reed-Muller Codes and Improvements of *Modified pqsigRM*

Reed-Muller 부호의 동형 연산과 *Modified pqsigRM*의 개선

지도교수 노 종 선

이 논문을 공학박사 학위논문으로 제출함

2023년 8월

서울대학교 대학원

전기 정보 공학부

조 진 규

조진규의 공학박사 학위 논문을 인준함

2023년 8월

위 원 장: _____
부위원장: _____
위 원: _____
위 원: _____
위 원: _____

Abstract

In this dissertation, two main contributions are given as; i) homomorphic computation in Reed-Muller (RM) codes and ii) improving *Modified pqsigRM* with the key size and bit-security.

First, a method of homomorphic computation in RM codes is proposed. With the ongoing developments in artificial intelligence (AI), big data, and cloud services, fully homomorphic encryption (FHE) is being considered as a solution for preserving privacy and security in machine learning systems. Currently, the most of existing FHE schemes are constructed using lattice-based cryptography. In state-of-the-art algorithms, a huge amount of computational resources are required for homomorphic multiplications and the corresponding bootstrapping that is necessary to refresh the ciphertext for a larger number of operations. Therefore, it is necessary to discover a new innovative approach for FHE that can reduce computational complexity for practical applications. Diverse research works, which are not limited to lattice-based cryptography are also needed. The code-based cryptography can be a new solution for this.

In this dissertation, I propose a code-based homomorphic operation scheme in RM codes. It is known that the linear codes are closed under the addition, however, achieving multiplicative homomorphic operations with linear codes has been impossible until now. I strive to solve this problem by proposing a fully homomorphic code scheme that can support both addition and multiplication simultaneously using the RM codes. This can be considered as a preceding step for constructing code-based FHE schemes. I restrict this to the computation of the first order of RM codes. As the order of RM codes increases after multiplication, a bootstrapping technique is required to reduce the order of intermediate RM codes to accomplish a large number of operations. I propose a bootstrapping technique to preserve the order of RM codes after the addition or multiplication by proposing three consecutive linear transformations that create a one-

to-one relationship between computations on messages and those on the corresponding codewords in RM codes. Furthermore, I propose some trials of making homomorphic encryption in code-based cryptosystems.

Second, a method of improving the key size and bit-security of *Modified pqsigRM* is proposed. The importance of post-quantum cryptography (PQC), which is secure against quantum algorithms, is growing larger and larger. *pqsigRM* is a code-based PQC digital signature scheme that was presented in round 1 of the national institute of standards and technology (NIST)'s PQC standardization process. NIST is a standardization organization in the United States. This scheme was revised as the *Modified pqsigRM* by removing all known vulnerabilities going through the debates in the NIST PQC standardization process. It has the advantages of an efficient decoding process and small signature sizes. Small signature sizes are very useful in digital signature schemes because signatures should be sent in every signing process. However, it has a problem with large public key sizes.

In this dissertation, I propose a method called *Improved Modified pqsigRM* to reduce the public key size and improve the exact bit-security of *Modified pqsigRM*. I change the public key into the systematic form, improve its parameters, and fine-tune the bit-security for each parameter. Thus, I can reduce these to 0.20, 0.40, and 0.23 times public key sizes compared with the *Modified pqsigRM* parameters for 80, 128, and 256 bit-security levels, respectively. Also, I obtain a larger exact bit-security for these parameters than *Modified pqsigRM*. Compared with the NIST PQC finalist algorithms *Crystals-Dilithium, Falcon,* and *Sphincs+*, the public key sizes are still large, but the signature sizes are the smallest among all for every security level. For 128 bits of classical security, the signature size of the proposed signature scheme is 520 bytes, which corresponds to 0.21 times that of *Crystals-Dilithium*. Moreover, I calculate the verification cycles compared with the NIST PQC finalist algorithms. The number of average verification cycles is 172,669, which corresponds to 0.53 times

that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms.

Furthermore, I propose an enhanced version of these, called *Enhanced pqsigRM*, considering the attacks using the information set decoding and finding the minimum-weight codewords. What is different from *Modified pqsigRM* is that it uses the systematic public key, minimizes the secret key, simplifies the usage of the hash function, and improves the security issues. There is a trade-off that the parameters get worse. However, compared with the original *Modified pqsigRM*, the public key size reduces to 2.0 MB, which is 0.5 times the previous one, and the secret key size reduces to 22,512 bytes, which is 0.0015 times the previous one. Also, it still has the advantages of a small signature size and fast verification cycles. These are very important features in digital signature schemes. The signature size is 1,032 bytes, which corresponds to 0.42 times that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms. The number of average verification cycles is 242,901, which corresponds to about 0.74 times that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms.

keywords: Fully homomorphic encryption (FHE), homomorphic computation, post-quantum cryptography (PQC), error-correcting codes (ECCs), Reed-Muller (RM) codes, digital signature scheme, code-based cryptosystem.

student number: 2017-26340

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Error-correcting codes (ECCs) are being used in diverse application areas. These have been developed for wireless communication systems in noisy channels and digital storage systems [1, 2]. Also, these are widely being used in other areas such as distributed computing systems or public-key encryption schemes. In the cryptography area, there is cryptography using ECCs, called code-based cryptography. The security of code-based cryptography is based on the fact that the decoding problem of a random linear code is an NP-complete problem [3]. It cannot be solved in a polynomial time. Especially, code-based cryptography is one of the candidates for post-quantum cryptography (PQC) that can resist attacks using operations over quantum computers.

Recently, machine learning has become popular in many areas and a few applications that employ this technology require the privacy of input data to be secured. For the security of machine learning, differential privacy and fully homomorphic encryption (FHE) are considered as candidates. In the case of differential privacy, the information on the individuals in the dataset is not disclosed even though the entire dataset is available. However, when it comes to FHE, both multiplication and addition can be performed for encrypted messages. Thus, confidential messages can be

securely manipulated on the untrusted cloud server. In FHE, the encryption schemes can support both addition and multiplication without any limitations on the number of operations.

Since Gentry proposed the first generation of FHE schemes in 2009 [4], there has been extensive research on homomorphic encryption schemes based on lattice-based hard problems [5–10]. The most promising recent research works on lattice-based homomorphic encryption schemes are the homomorphic encryption for the arithmetic of approximate numbers scheme, called the Cheon-Kim-Kim-Song (CKKS) scheme [6] and the fast FHE over the torus (TFHE) scheme [7]. Despite substantial progress since Gentry's first FHE scheme, the computational complexity of FHE schemes is still too high to be used in privacy-preserving machine learning systems. For example, it takes almost 30 seconds for one bootstrapping operation while using the CKKS library. As more than 10^5 bootstrappings are necessary to sort hundreds of data packets, several days are needed for the homomorphic sorting operation [10]. Therefore, a new innovative approach to achieve a more efficient FHE scheme is needed to be discovered. The research works on FHE are too concentrated on lattice-based cryptography. For diversity, FHE schemes that are not based on lattice theory are needed such as code-based cryptography. To make code-based homomorphic encryption possible, a scheme that can decode errors after homomorphic operations is needed. Also, the key sizes should be reduced to use the homomorphic operations feasibly.

Besides, there are numerous studies on the application of ECCs in quantum computing [11–14]. Researchers have also succeeded in mapping the ECCs of classical computers into new quantum codes called 'stabilizer codes.' Several trials of using algebraically defined codes, such as Hamming codes, Bose-Chaudhuri-Hocquenghem (BCH) codes, RM codes, and Golay codes have been accomplished. Moreover, methods for using sparse-graph codes such as low-density parity-check (LDPC) codes are also being studied. The goal of all these studies is to achieve fault-tolerant quantum computation. In the future, I expect homomorphic computation to be an important

component in quantum computers, and therefore my contribution to creating a homomorphic computation method for RM codes may turn out to be useful in RM-code-based quantum error correction.

Along with the extremely fast-growing data and computation sizes, the size of distributed computing systems has also grown increasingly larger with time. During computing, a certain amount of unpredictable system noise or straggler nodes that cause delays cannot be avoided. To reduce these problems, coded computation, which is a method of using coding theoretic techniques in distributed computing systems, is frequently used. In this regard, the first known study was conducted on the computing matrix multiplication problem using erasure codes and minimum distance separable codes [15]. Later, more studies with diverse approaches to increasing the speed of coded computations also appeared [16–18]. System components, such as sensors, are required to work efficiently even in noisy conditions, such as high temperatures. To ensure this, I need a coding technique using high error tolerance codes, such as Reed-Muller (RM) codes, because they can correct random erasures and errors with high probability [19]. Thus, the study of homomorphic computation on codes can expand the usage of codes also in distributed computing systems.

In this dissertation, I propose a code-based homomorphic computation scheme using RM codes that can support simultaneous addition and multiplication operations. Addition can be performed freely in linear codes due to the nature of the linearity. However, no such method of multiplication exists for codewords, that is, it is not possible to obtain a valid codeword just by multiplying two valid codewords. Therefore, I propose a linear transformation that can map the multiplication result of any pair of valid codewords to a valid codeword. As the order of the codewords monotonically increases during the multiplication in the RM codes, there is a certain limitation on the number of multiplications. To resolve this problem, I propose a bootstrapping method using a linear transformation to reduce the order of the second-order RM codes to the first-order after the codeword multiplication.

Nowadays, the performance of quantum computers is growing very fast. When quantum computers become well commercialized, the cryptosystems we are using now, such as RSA or elliptic curve cryptography, are proven to be broken. The integer factorization problem and discrete logarithm problem for a large integer have been foundations for these cryptosystems for a long time. However, using Shor's algorithm [20], these problems can be solved with quantum computers in a polynomial time. Therefore, PQC, which is strong even against quantum computers, is getting the spotlight. The US's standardizing organization NIST is leading a PQC standardization process. After revising or withdrawing lots of algorithms, the fourth round of this process is in progress.

There are several types of post-quantum cryptosystems. First, lattice-based cryptography is the most promising because it has very good performance and a small parameter size. Also, code-based cryptography is the second most popular type of PQC. It is based on the coding theory and the security is based on the syndrome decoding problem (SDP) which is an NP-complete problem [1, 2]. For example, the oldest and best-known code-based cryptosystem is McEliece public key encryption scheme [3]. Code-based cryptosystems have the advantage of a strong security problem, which is not broken for over 40 years. However, these have the disadvantage of large key sizes compared with other cryptosystems.

There also exist several digital signature schemes with code-based cryptosystems. Courtois-Finiasz-Sendrier (CFS) signature scheme is the oldest and most popular code-based signature scheme that was proposed by Courtois, Finiasz, and Sendrier in 2001 [21]. *Wave* [22] and *pqsigRM* [23] are the code-based signature schemes that are deviated from the CFS scheme. *Wave* adopted generalized ternary $(U, U + V)$-codes to make efficient decoding and well-proven security. On the other hand, *pqsigRM* uses RM codes which are also $(U, U + V)$-codes. The original version of *pqsigRM* was presented in the NIST PQC standardization process round 1 as a code-based signature scheme. Later it was improved to *Modified pqsigRM* by Lee *et al.,* in 2019 [24]. It

has an efficient decoding algorithm and small signature sizes. However, there exists a drawback of large public key sizes. In the digital signature scheme, having a small signature size is a great advantage because a signature should be sent in every signing process. On the other hand, a public key is sent just once. Also, there is a rank metric code-based signature scheme called *Durandal* [25]. It has very small key sizes as lattice-based cryptosystems. However, the security of this scheme is not fully proven yet.

In this dissertation, I propose a method called *Improved Modified pqsigRM* to reduce the public key size of *Modified pqsigRM*. I use a systematic public key and reset the parameters considering the size of the public key and the security level. I find several improved values for these. Thus, I can reduce these to 0.20, 0.40, and 0.23 times public key sizes compared to the *Modified pqsigRM* parameters for 80, 128, and 256 security levels, respectively. Also, I obtain a larger exact bit-security for these parameters than *Modified pqsigRM*. Compared with NIST PQC finalist algorithms, the public key size is still large, but the signature size is the smallest among all. Moreover, I calculate the verification cycles compared with the NIST PQC finalist algorithms. For 128 bits of classical security, the signature size of the proposed signature scheme is 520 bytes, which corresponds to 0.21 times that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms. The number of average verification cycles is 172,669, which corresponds to 0.53 times that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms.

Furthermore, I propose an enhanced version of these, called *Enhanced pqsigRM*, considering the attacks using the information set decoding and finding the minimum-weight codewords. It enhances the security issues of the previous work, but the parameters get worse. However, compared with the original *Modified pqsigRM*, the public key size reduces to 2.0 MB, which is 0.5 times the previous one, and the secret key size reduces to 22,512 bytes, which is 0.0015 times the previous one. Also, it has the advantages of a small signature size and fast verification cycles. These are very important

features in digital signature schemes. The signature size is 1,032 bytes, which corresponds to 0.42 times that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms. The number of average verification cycles is 242,901, which corresponds to about 0.74 times that of *Crystals-Dilithium*, and it is the second smallest among the NIST PQC finalist algorithms.

1.2 Overview of Dissertation

This dissertation is organized as follows.

In Chapter 2, I introduce preliminaries that are needed in this dissertation. First, I claim the basic notations of this dissertation and describe the fundamental knowledge of the original RM codes. I also explain the features of PQC and code-based cryptography. Moreover, I present the fundamental definition of FHE. Then, I explain the basic concepts of the digital signature scheme.

In Chapter 3, I present a method of homomorphic computation in RM codes. I define the addition and multiplication operations on the message and codeword domains, respectively, and also propose the main scheme used in my bootstrapping technique for RM codes. Then I introduce some examples to help understand the idea. Furthermore, I propose some trials of making homomorphic encryption in code-based cryptosystems.

In Chapter 4, first I explain the details of *Modified pqsigRM*. Then, I present *Improved Modified pqsigRM*, which improves the key size of *Modified pqsigRM*, by making a public key in a systematic form and choosing new parameters for the keys. I show how to make public keys in a systematic form. Then I present which parameter sets to choose, the expected strength of them, and the design rationale. Then the improved key sizes by choosing new parameters are introduced. Moreover, the bit-security is also shown to be improved. The comparison with other PQC schemes is explained, too. The public key and signature sizes comparing these with the proposed scheme are introduced. Also, the verification cycles compared with other NIST PQC finalist

schemes are specified. Then the enhanced version of this algorithm called *Enhanced pqsigRM* is introduced. I enhance some specifications of the algorithm considering the review of the NIST PQC organizer team. The parameter set and design rationale of this is presented. Also, the comparison with other PQC schemes is explained. The public key, signature size, and verification cycles are compared. Moreover, memory usage is also presented.

In Chapter 5, I summarize the main points of this dissertation and mention some issues which can be future works.

Chapter 2

Preliminaries

In this chapter, I introduce some preliminaries to help understand the dissertation. First, I explain the basic notations of this dissertation. Then, I introduce the fundamental knowledge of RM codes. I also present the features of post-quantum cryptography and code-based cryptography. Furthermore, I show the concept of fully homomorphic encryption. Lastly, I explain about the basic concept of the digital signature schemes and CFS signature scheme.

2.1 Notation

I express a row vector with a small letter alphabet in boldface as '$\boldsymbol{a}$' and a matrix with capital letter alphabet in boldface as '$\boldsymbol{A}$.' A concatenation of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is notated as '$(\boldsymbol{a}|\boldsymbol{b})$.' An integer is notated as an alphabet without boldface as 'a.' A polynomial of x is notated as '$a(x)$.' Also, I express the algorithm name with the italic font as '*pqsigRM*.'

2.2 RM Codes

In this subsection, I briefly introduce the fundamental notions and properties of RM codes. Reed [26] and Muller [27] first suggested RM codes in 1954. An RM code, RM(r, m) is defined with integers r and m, where r is the order of the code and $n = 2^m$ is the code length. The dimension of RM(r, m) is $k_r = \sum_{i=0}^{r} \binom{m}{i}$ and the minimum distance is $d_{min} = 2^{m-r}$. Further, I can express RM(r, m) with the r-th order linear combinations of Boolean functions $\boldsymbol{v_0} = 1, \boldsymbol{v_1}, \cdots, \boldsymbol{v_m}$. Thus, the generator matrix $\boldsymbol{G_r}$ of the r-th order RM code, RM(r, m) can be expressed as

$$\boldsymbol{G_r} = \begin{pmatrix} \boldsymbol{v_0} \\ \boldsymbol{v_1} \\ \vdots \\ \boldsymbol{v_m} \\ \boldsymbol{v_1 v_2} \\ \boldsymbol{v_1 v_3} \\ \vdots \\ \boldsymbol{v_{m-1} v_m} \\ \vdots \\ \boldsymbol{v_1 \cdots v_r} \\ \boldsymbol{v_1 \cdots v_{r-1} v_{r+1}} \\ \vdots \\ \boldsymbol{v_{m-r+1} \cdots v_m} \end{pmatrix}, \tag{2.1}$$

where, $\boldsymbol{v_i v_j}$ denotes the component-wise multiplication of $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ [1, 2, 26, 27]. I abuse notations of the real generator matrix and the matrix of Boolean functions. The columns of the generator matrix are evaluated by each Boolean function of each row with every m-tuple binary vector from $(1, 1, \cdots, 1)$ to $(0, 0, \cdots, 0)$ in the reverse lexicographical order.

The generator matrix $\boldsymbol{G}$ of the first-order RM codes is constructed from $\{\boldsymbol{v_0}, \boldsymbol{v_1}, \cdots, \boldsymbol{v_m}\}$ in (2.1), and the generator matrix of the second-order RM codes is created from $\{\boldsymbol{v_0}, \boldsymbol{v_1}, \cdots, \boldsymbol{v_m}, \boldsymbol{v_1 v_2}, \boldsymbol{v_1 v_3}, \cdots, \boldsymbol{v_{m-1} v_m}\}$. The first-order RM codes have a dimension of $k = m+1$ and can be represented with linear combinations of $\boldsymbol{v_0}, \cdots, \boldsymbol{v_m}$.

For example, the generator matrix of RM(4,4) can be expressed as

$$\boldsymbol{G}_{RM(4,4)} = \begin{pmatrix}
1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1 \\
1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0 \\
1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0 \\
1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0 \\
1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0 \\
1\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0 \\
1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,1\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0 \\
1\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0
\end{pmatrix},$$

which is a 16×16 square matrix.

The generator matrices of RM(1,4), RM(2,4), and RM(3,4) are included in $\boldsymbol{G}_{RM(4,4)}$. From the first row to the fifth row make $\boldsymbol{G}_{RM(1,4)}$, which is the first-order RM code. From the first row to the eleventh row make $\boldsymbol{G}_{RM(2,4)}$, which is the second-order RM code. From the first row to the fifteenth row make $\boldsymbol{G}_{RM(3,4)}$, which is the third-order RM code. At last, $\boldsymbol{G}_{RM(4,4)}$ is the square code, and it is the fourth-order RM code. For

RM codes with the same value of m, the codes with larger r contain the codes with smaller r.

The message can be expressed with a polynomial of degree m, $a(x)$, or with a $(m+1)$-tuple vector $\boldsymbol{a}$ as

$$a(x) = \sum_{i=0}^{m} a_i x^i$$
$$\boldsymbol{a} = (a_0, a_1, \cdots, a_m).$$

And a codeword of the first-order RM code, RM(1,m), can be expressed with a polynomial of degree $n-1$, $c(x)$, or with an n-tuple vector $\boldsymbol{c}$ by multiplying the $k \times n$ generator matrix $\boldsymbol{G}$ in (2.1) to message as

$$c(x) = \sum_{i=0}^{n-1} c_i x^i$$
$$\boldsymbol{c} = (c_0, c_1, \cdots, c_{n-1}) = \boldsymbol{a}\boldsymbol{G} = \sum_{i=0}^{m} a_i \boldsymbol{v_i}, \tag{2.2}$$

where I abuse the vector and polynomial notations.

RM codes also have a recursive structure, where the generator matrix of RM(r, m) is given as

$$\boldsymbol{G}_{RM(r,m)} = \begin{pmatrix} \boldsymbol{G}_{RM(r,m-1)} & \boldsymbol{G}_{RM(r,m-1)} \\ 0 & \boldsymbol{G}_{RM(r-1,m-1)} \end{pmatrix}.$$

This structure makes RM codes to be $(U, U+V)$-codes. $(U, U+V)$-codes are codes that have $(U, U+V)$ structure, where U and V are codes with a half-length of the whole codes. There also exists an efficient decoding algorithm for RM codes using the characteristic of the recursive structure [28].

RM codes are widely being used such as for public key encryption schemes [29], homomorphic computations [30], secret sharing [31], and private information retrieval [32].

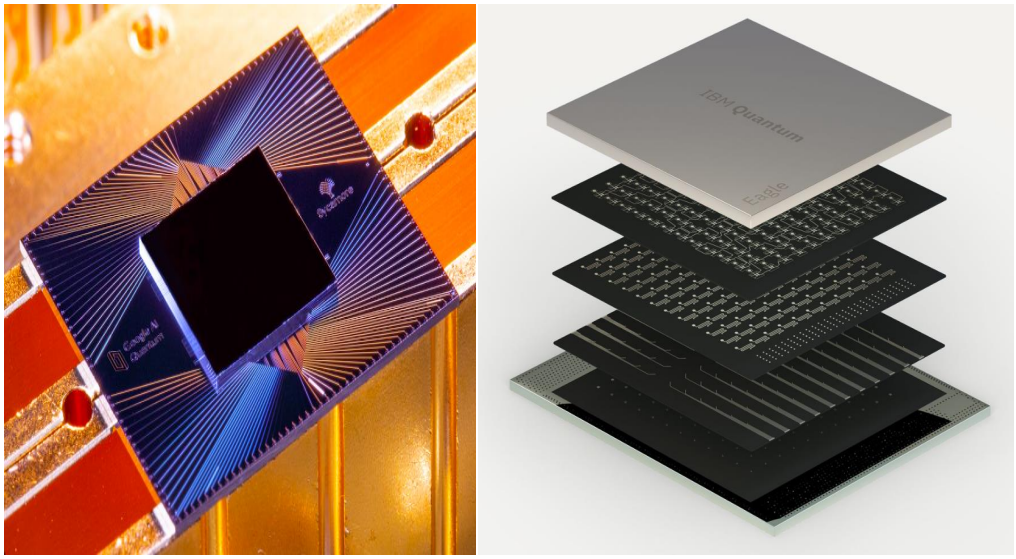

Figure 2.1: Google's Sycamore (left) and IBM's Eagle (right).

2.3 Post-Quantum Cryptography

The performance of quantum computers is growing very fast. Global IT companies such as Google and IBM are working on it. The researchers of Google announced that they had achieved quantum supremacy in 2019. Quantum supremacy or quantum advantage is a research objective that a quantum computer solves a problem, while classical computers cannot in a feasible time [33, 34]. They made a quantum computer called Sycamore, which has 53 qubits. It could solve a problem in 3 minutes, while IBM's supercomputer Summit is known to solve it in 10,000 years. On the other hand, the researchers of IBM made a 127-qubit processor called Eagle in 2021 and also made a 433-qubit processor called Osprey in 2022. They are working on the 1,121-qubit processor called Condor for 2023. Also, they unveiled their quantum hardware plans as in Figure 2.2 and are achieving these step by step. They said that they will achieve a 4,000-qubit quantum computer by 2025. Likewise, Google and IBM are having competitions on quantum computers continuously.

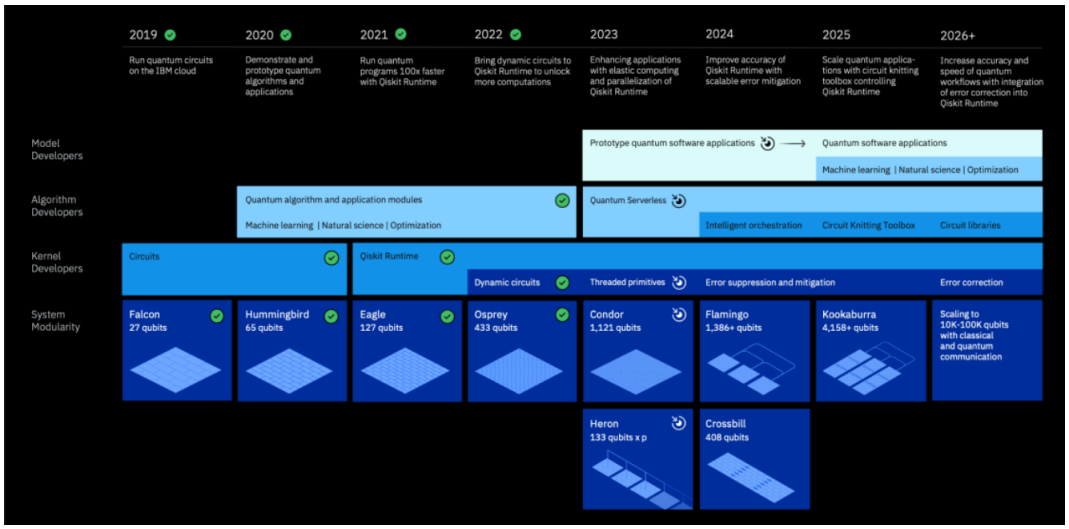

Figure 2.2: IBM's roadmap for quantum computer research.

Shor proposed a quantum computer algorithm, which can break the classical cryptosystems in 1994 [20]. This can compute the factorization problem of large integers or discrete logarithm problems very fast. Thus, the current cryptosystems such as RSA or elliptic curve cryptography, which are based on these problems, are proven to be broken by quantum computers. It is known that about 4000 qubits will be needed to break RSA cryptosystems and it is not that far to happen.

Thus, post-quantum cryptography (PQC), which is secure against quantum algorithms, is getting the spotlight. There exists lattice-based, code-based, multivariate, hash-based cryptography, and so on for PQC.

2.4 Code-Based Cryptography

The US's standardizing organization NIST has been leading a PQC standardization process since January 2017. As in Table 2.1, they started with 64 algorithms for the first round, and 8 algorithms are left now, after the fourth round. For the fourth round, the numbers in the bracket implement the numbers of candidate algorithms, which

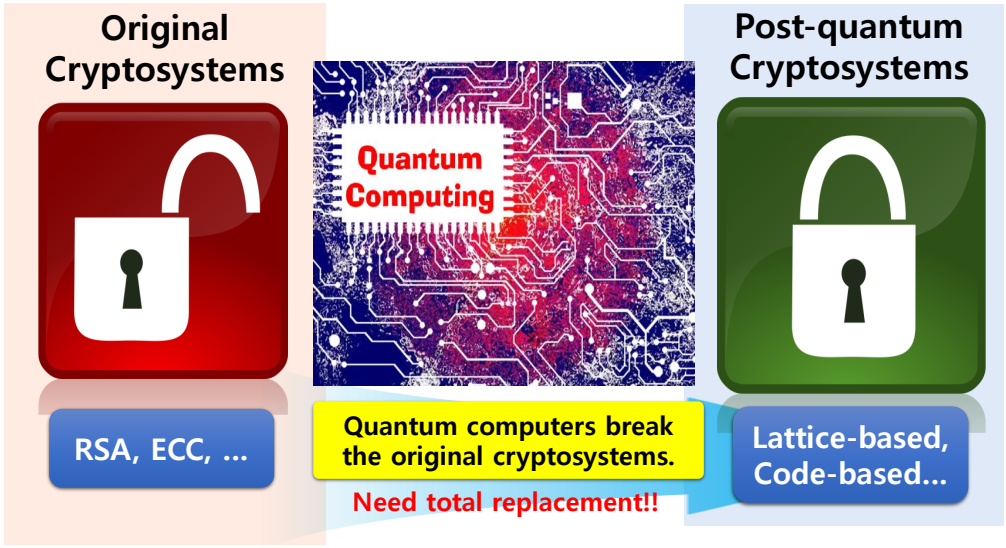

Figure 2.3: Need for post-quantum cryptography.

will be considered more by NIST. On the other hand, the numbers without brackets in the fourth round are the numbers of finalist algorithms. For now, *Crystals-Kyber* [35], *Crystals-Dilithium* [36], and *Falcon* [37] for lattice-based cryptosystems, *Classic McEliece* [38], *BIKE* [39], and *HQC* [40] for code-based cryptosystems, *Sphincs+* [41] for hash-based cryptosystem, and *SIKE* [42] for isogeny-based cryptosystem, are left as in Figure 2.5.

Table 2.1 shows that the lattice-based cryptosystem takes the largest part with 26 schemes in round 1 and 3 schemes left after round 4. The lattice-based cryptosystem is the most promising among other cryptosystems because it has good performance and small signature sizes. These are based on the lattice theory.

For example, there are ring learning with errors (RLWE) key exchange [43] and Goldreich–Goldwasser–Halevi (GGH) encryption schemes [44] for traditional systems. Moreover, there are *Crystals-Kyber* [35], *Crystals-Dilithium* [36] and *Falcon* [37] for NIST PQC finalists. *Crystals-Dilithium* and *Falcon* are signature schemes, which are based on the learning with error (LWE) problem and NTRU-based schemes,

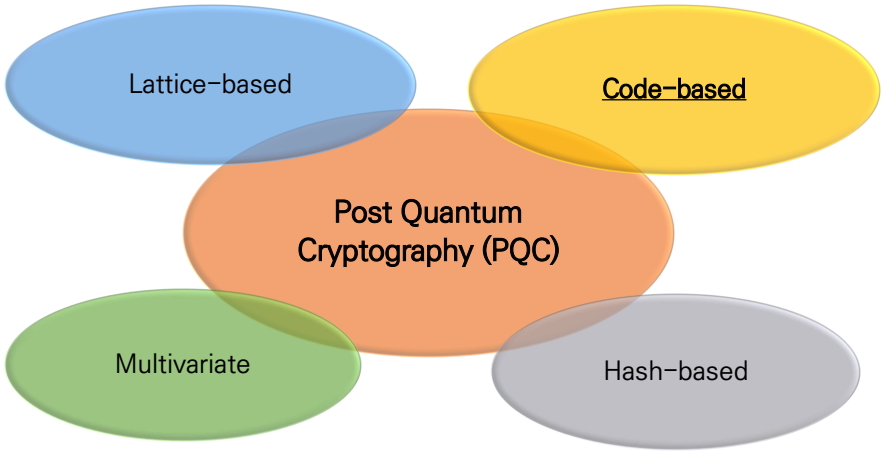

Figure 2.4: Various kinds of post-quantum cryptography algorithms.

respectively. *Crystals-Kyber* is a key encapsulation mechanism (KEM), which has a similar base with *Crystals-Dilithium*. These three schemes have the advantages of fast computing speed and small key sizes.

Also, I can figure out that the code-based cryptosystem takes the second largest part with 19 schemes in round 1 and 3 schemes to be considered in round 4. It has a strong point with the traditional security problem, which has not been broken over 40 years. Also, it can be easier to make hardware implementations because it uses binary operations, while the lattice-based cryptosystem uses modulo operations. However, it has weak points in large key sizes.

Code-based cryptography is a cryptography with ECCs and it is based on the syndrome decoding problem, which is an NP-complete problem. ECCs are being used in diverse applications such as wireless communication systems in noisy channels, digital storage systems, distributed computing, and cryptography.

Problem 1. ***Syndrome decoding problem***

A syndrome decoding problem is a problem that finds a vector $\boldsymbol{e}$ *from a syndrome* $\boldsymbol{s} = \boldsymbol{e}\boldsymbol{H}^{\boldsymbol{T}}$. $\boldsymbol{H}$ *is a parity check matrix of a random code and the Hamming weight of*

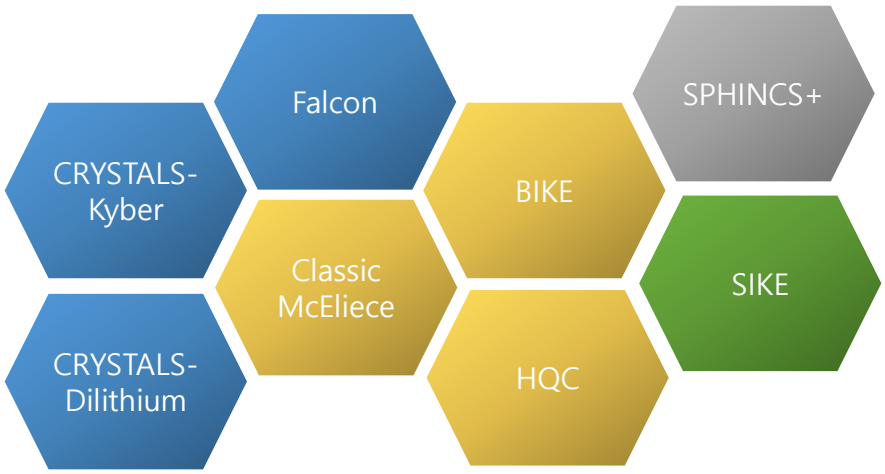

Figure 2.5: NIST PQC round 4 algorithms.

$\boldsymbol{e}$ *must be less than or equal to the error correcting capability* t.

For example, there are McEliece public-key encryption [3] and CFS digital signature scheme [21] for traditional code-based cryptosystems. Moreover, there are *Classic McEliece* [38], *BIKE* [39], and *HQC* [40] for NIST PQC round 4 candidates. For digital signature schemes, NIST is calling for additional schemes because it is concentrated on lattice-based algorithms. NIST mentioned that the schemes should be diversified and the digital signature schemes with short signature sizes and fast verification cycles are needed. I am on this process with a code-based post-quantum signature scheme, called *Enhanced pqsigRM*, and this will be explained in this dissertation. It has the advantages of short signature and fast verification.

The first code-based cryptosystem was invented by McEliece in 1978 [3]. It is a simple public key encryption scheme with matrix computation. The binary Goppa codes are used for this cryptosystem and there has been no valid attack on this for over 40 years. In 1986, Niederreiter proposed a dual version of the McEliece cryptosystem, and the parity check matrix is used instead of the generator matrix of the codes [45]. Also, there were several attempts with other codes such as generalized Reed-Solomon

Table 2.1: The number of algorithms for NIST PQC standardization rounds 1 and 4

	Signatures		KEM/Encryption		Overall	
	1R	4R	1R	4R	1R	4R
Lattice-based	5	2	21	1	26	3
Code-based	2	-	17	(3)	19	(3)
Multi-variate	7	-	2	-	9	-
Hash/symmetric based	3	1	-	-	3	1
Other	2	-	5	(1)	7	(1)
Total	19	3	45	1(+4)	64	4(+4)

(GRS) codes [46], RM codes [29, 47], quasi cyclic-low density parity check (QC-LDPC) codes [48], and polar codes [49].

McElice public key encryption scheme is specified as Algorithm 1. For key generation, I construct $\boldsymbol{G_{pub}}$ by multiplying $\boldsymbol{S}, \boldsymbol{G}$, and $\boldsymbol{P}$. $\boldsymbol{G}$ is the generator matrix of binary Goppa codes, and $\boldsymbol{S}, \boldsymbol{P}$ are random invertible matrices. $\boldsymbol{G_{pub}}$ becomes the public key and $\boldsymbol{S}, \boldsymbol{G}, \boldsymbol{P}$ become secret keys. For encryption, I compute the ciphertext $\boldsymbol{c}$ from the message $\boldsymbol{m}$ by making a codeword added by an error $\boldsymbol{e}$. The Hamming weight of $\boldsymbol{e}$ becomes t. For decryption, I compute $\boldsymbol{c'}$ from multiplying the inverse of $\boldsymbol{P}$ to $\boldsymbol{c}$. Then, I decode $\boldsymbol{c'}$ to $\boldsymbol{m'}$ using the decoding algorithm of binary Goppa codes. At last, I return $\hat{\boldsymbol{m}}$ by multiplying the inverse of $\boldsymbol{S}$ to $\boldsymbol{m'}$. If $\hat{\boldsymbol{m}}$ is the same as the original message $\boldsymbol{m}$, the decryption is successful.

Classic McEliece [38] is one of the NIST PQC round 4 algorithms. It is an updated version of the McEliece public-key encryption scheme for 128 and 256 bit-security levels, which are required security levels of the NIST PQC team. It has the advantage of the oldest proven security, but the disadvantage of large public key sizes. *BIKE* and *HQC* are also NIST PQC key encapsulation mechanisms with QC-MDPC codes and quasi-cyclic codes, respectively. These have smaller key sizes than *Classic McEliece*

Algorithm 1 McEliece public key encryption scheme [3]

Key Generation :

$\boldsymbol{G}$: $k \times n$ generator matrix of binary Goppa codes

t: Error-correcting capability of the binary Goppa codes

$\boldsymbol{S} \xleftarrow{\$} F_2^{k \times k}, \boldsymbol{P} \xleftarrow{\$} F_2^{n \times n}$

$\boldsymbol{G_{pub}} \leftarrow \boldsymbol{SGP}$

Public key: $\boldsymbol{G_{pub}}$

Secret key: $\boldsymbol{S}, \boldsymbol{G}, \boldsymbol{P}$

Encryption :

Compute ciphertext from message $\boldsymbol{m}$:

$\boldsymbol{c} \leftarrow \boldsymbol{mG_{pub}} + \boldsymbol{e}$

$\boldsymbol{e}$: Error vector with Hamming weight t

Decryption :

Compute $\boldsymbol{c'} \leftarrow \boldsymbol{cP^{-1}}$

Decode $\boldsymbol{c'}$ to $\boldsymbol{m'}$ using the decoding algorithm of Goppa codes

Return $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{m'S^{-1}}$

but have a little problem with security.

2.5 Fully Homomorphic Encryption

Briefly speaking, homomorphic encryption is encryption that can compute something in an untrusted cloud without leaking the important information as in Figure 2.6. It plays a crucial part in ensuring privacy because it does not expose the original data in computing environments such as machine learning or cloud services [4]. In other words, an encryption scheme is said to be "homomorphic" with respect to an operation $\Diamond$ on plaintext space P if it satisfies

$$\begin{aligned} &Decrypt(Encrypt(m_1) * Encrypt(m_2)) \\ &\qquad = Decrypt(Encrypt(m_1 \Diamond m_2)) \\ &\qquad = m_1 \Diamond m_2 \end{aligned}$$

for an operation $*$ on ciphertext space C. The operation $\Diamond$ is usually an addition or multiplication.

An encryption scheme is called "somewhat homomorphic" if it satisfies only a limited number of operations because of its inability to perform decryption after a certain number of operations. If a scheme can perform an infinite number of homomorphic operations for addition and multiplication, it is called "fully homomorphic" [5]. Homomorphic encryption is needed in many places such as the Internet of Things (IoT), cloud computing, internet network, and so on as in Figure 2.7. It is getting more and more spotlights.

2.6 Digital Signature

A digital signature scheme consists of three parts. First, public keys and secret keys are generated in the key generation process. Then, a signature is signed using the secret

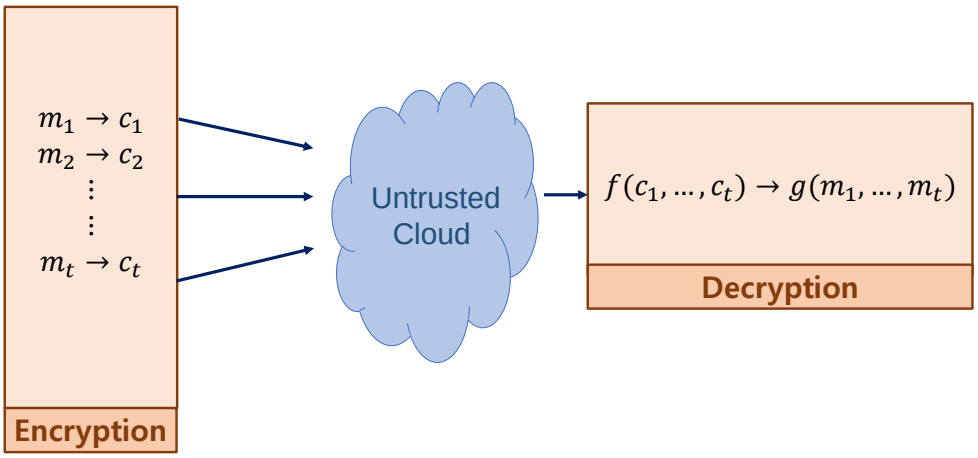

Figure 2.6: The process of homomorphic encryption.

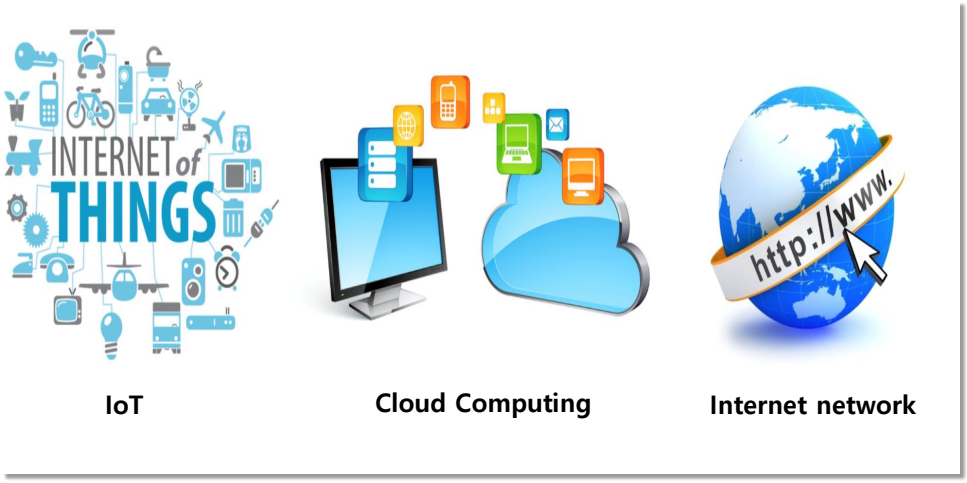

Figure 2.7: Need for homomorphic encryption.

keys in the signing process. At last, the signature is verified using the public keys in the verification process. If an eavesdropper brings a wrong signature, it must be figured out in the verification process.

Courtois, Finiasz, and Sendrier proposed the first code-based digital signature scheme in 2001 [21]. The scheme is called the CFS scheme and it is a modified version of the Niederreiter cryptosystem. As in Algorithm 2, it consists of key generation, signing, and verification processes.

In the key generation, $\boldsymbol{H}$ is taken as a parity check matrix of (n, k) Goppa codes. The error correction capability is $\frac{n-k}{\log n}$. Then random invertible matrices $\boldsymbol{S}$ and $\boldsymbol{Q}$ are constructed. $\boldsymbol{H'}$, which is the multiplication of $\boldsymbol{S}$, $\boldsymbol{H}$, and $\boldsymbol{Q}$ becomes public key. And $\boldsymbol{S}$, $\boldsymbol{H}$, and $\boldsymbol{Q}$ become secret keys. In signing, a message m is signed to a syndrome $\boldsymbol{s}$ by hashing. Then $\boldsymbol{s'}$ is computed from $\boldsymbol{s}$ by multiplying the inverse of $\boldsymbol{S}$. This is repeated until a decodable syndrome $\boldsymbol{s'}$ is found and this is counted with a counter integer i. Then, an error $\boldsymbol{e'}$, which satisfies $\boldsymbol{s' = He'^T}$ is found. Using the decoding algorithm of the Goppa codes, $\boldsymbol{e'}$ can be found and this is an NP-complete problem when the decoding algorithm is unknown. $\boldsymbol{e}$ is computed from $\boldsymbol{e'}$ by multiplying the inverse of $\boldsymbol{Q}$ and the signature becomes $(\boldsymbol{m}, \boldsymbol{e}, i)$. In verification, whether the Hamming weight of $\boldsymbol{e}$ is smaller than or equal to t and whether $\boldsymbol{H'e}^T = h(h(\boldsymbol{m})|i)$ are checked. If these are true, the signing is successful.

To find a valid signature, I must perform $t!$ trials on average. Thus, I should set t with a small value and this means high-rate Goppa codes should be used. However, the high-rate Goppa codes are known to be distinguished from random matrices [50]. Thus, CFS digital signature scheme using Goppa codes is not secure. In *pqsigRM*, a modification of RM codes is used instead of Goppa codes and this does not have such problems.

Wave [22] and *pqsigRM* [23] are based on the CFS scheme. *Wave* adopted generalized ternary $(U, U+V)$-codes to make efficient decoding and well-proven security. On the other hand, *pqsigRM* uses the modification of RM codes which are also $(U, U+V)$-

codes. It has an efficient decoding algorithm and small signature sizes. Having a small signature size is a great advantage in the digital signature scheme because a signature should be sent in every signing process. However, there exists a drawback of large public key sizes. I improved this drawback in this dissertation. I will explain more about this in Chapter 4.

Algorithm 2 CFS signature scheme [21]

Key Generation:

$\boldsymbol{H}$: The parity check matrix of (n, k) Goppa codes

The error correction capability t is $\frac{n-k}{\log n}$

$\boldsymbol{S}$ and $\boldsymbol{Q}$: An $(n-k) \times (n-k)$ scrambler matrix and $n \times n$ permutation matrix, respectively

Public key: $\boldsymbol{H'} \leftarrow \boldsymbol{SHQ}$

Secret key: $\boldsymbol{H}, \boldsymbol{S},$ and $\boldsymbol{Q}$

Signing:

$\boldsymbol{m}$ is a message to be signed

$i \leftarrow 1$

Do

$i \leftarrow\ i + 1$

Find syndrome $\boldsymbol{s} \leftarrow h(h(\boldsymbol{m})|i)$

Compute $\boldsymbol{s'} \leftarrow \boldsymbol{S}^{-1}\boldsymbol{s}$

Until a decodable syndrome $\boldsymbol{s'}$ is found

Find an error vector satisfying $\boldsymbol{H}\boldsymbol{e'}^T \leftarrow \boldsymbol{s'}$

Compute $\boldsymbol{e}^T \leftarrow \boldsymbol{Q}^{-1}\boldsymbol{e'}^T$, and then the signature is $(\boldsymbol{m}, \boldsymbol{e}, i)$

Verification:

Check $\mathrm{wt}(\boldsymbol{e}) \leq t$ and $\boldsymbol{H'}\boldsymbol{e}^T = h(h(\boldsymbol{m})|i)$

If True, then return ACCEPT; else, return REJECT

Chapter 3

Homomorphic Computation in Reed-Muller Codes

In this chapter, I present a method of homomorphic computation in RM codes. I define the addition and multiplication operations on the message and codeword domains, respectively. Also, I propose the main scheme used in the proposed bootstrapping technique for RM codes. Then I introduce some examples to help understand the process of the proposed idea.

3.1 Addition and Multiplication in RM Codes

Homomorphic operations are executed both on the message and codeword domains. While the addition is performed identically in both domains, the multiplication of the codewords must be defined as a new codeword of a message that is defined as a polynomial multiplication with modulo $x^k - 1$.

In the proposed scheme, I only consider the first-order RM codes for homomorphic operations because they have the maximum Hamming distance 2^{m-1} and can be efficiently used for related homomorphic computations. As described in Subsection 2.2, the RM codes can be described as polynomials. Therefore, for the homomorphic addition of two codewords, I perform a component-wise addition $\oplus$ between the coefficients of the same order of the polynomial terms. In the case of homomorphic

multiplication, the codewords can be multiplied by performing some multiplication $\odot$, which corresponds to the codeword of multiplication of two message polynomials of the order m, $a(x)$ and $a'(x)$, as $a(x)a'(x) \bmod (x^{m+1} - 1)$, as given in Table 3.1.

In the case of the addition, it is evident that the computation on the message and codeword domains are directly related because the RM codes are linear. However, while multiplying two corresponding codewords, $c(x)$ and $c'(x)$, or $\boldsymbol{c}$ and $\boldsymbol{c}'$, I have two fundamental problems. The first problem is that just multiplying $\boldsymbol{c}$ and $\boldsymbol{c}'$ component-wisely does not completely match the codeword corresponding to the multiplied message. Therefore, I need to apply the linear transformation to correctly match the message and codeword domains. The second problem is that the multiplied codeword is the second-order RM code instead of the first-order. To fix the relationship between the two domains and reduce the order of the codeword, I need a linear transformation of the multiplied codewords. This linear transformation is described in the next subsection.

Table 3.1: Addition and multiplication with messages and codewords

Computation	Messages	Codewords
Addition	$a(x) + a'(x) \bmod (x^{m+1} - 1)$	$\boldsymbol{c} \oplus \boldsymbol{c}'$
Multiplication	$a(x)a'(x) \bmod (x^{m+1} - 1)$	$\boldsymbol{c} \odot \boldsymbol{c}'$

3.2 Bootstrapping Technique in RM Codes

The two problems encountered during the homomorphic multiplications of RM codes can be resolved by using a bootstrapping technique. This bootstrapping technique is operated on the plaintext domain. The first-order RM codes are represented with linear combination of $\{\boldsymbol{v_0}, \boldsymbol{v_1}, \cdots, \boldsymbol{v_m}\}$. The multiplied codewords are a part of the second-order RM codes, which are RM (2,m). Therefore, I need to reduce the order of the RM

codes for further operations.

Let $\boldsymbol{c}$ and $\boldsymbol{c}'$ be two distinct codewords of the first-order RM codes for two messages $a(x)$ and $a'(x)$. The multiplication of $a(x)$ and $a'(x)$ is expressed as

$$a(x)a'(x) \bmod (x^{m+1}-1) = \sum_{l=0}^{m}(\sum_{i+j=l \bmod (m+1)} a_i a'_j)x^l. \tag{3.1}$$

Thus, the corresponding codeword of $a(x)a'(x)$ mod $(x^{m+1}-1)$ is given as

$$\sum_{l=0}^{m}(\sum_{i+j=l \bmod (m+1)} a_i a'_j)\boldsymbol{v_l}. \tag{3.2}$$

However, the direct multiplication of the two corresponding codewords is given as

$$(\sum_{i=0}^{m} a_i \boldsymbol{v_i})(\sum_{j=0}^{m} a'_j \boldsymbol{v_j}). \tag{3.3}$$

Therefore, (3.2) and (3.3) are not the same even though they represent the same message. Notably, (3.3) is the second-order RM code. Therefore, (3.3) should be modified to fit (3.2). This process is called bootstrapping in this chapter.

The bootstrapping process comprises three steps as follows.

1. Step 1: Represent the coefficients $a_i a'_j + a_j a'_i$ of $\boldsymbol{v_i v_j}$ in (3.3) as the components of the codewords $\boldsymbol{c} = (c_0, c_1, \cdots, c_{n-1})$ and $\boldsymbol{c'} = (c'_0, c'_1, \cdots, c'_{n-1})$, whose transformation is denoted by $(n+m) \times (k_2+m)$ matrix $\boldsymbol{V}$.

2. Step 2: Derive the coefficients $\sum_{i+j=l \bmod (m+1)} a_i a'_j$ of x^l in (3.1) by using coef($\boldsymbol{v_i v_j}$), whose transformation is denoted by $(k_2+m) \times k$ matrix $\boldsymbol{X}$.

3. Step 3: Find the codeword $\boldsymbol{c_{new}}$ of the message $a(x)a'(x)$ mod $(x^{m+1}-1)$ in RM(1,m) by using the generator matrix $\boldsymbol{G}$.

The proposed bootstrapping procedure for homomorphic multiplication in RM (1, m) code is depicted in Figure 3.1, where notations of polynomials and vectors are

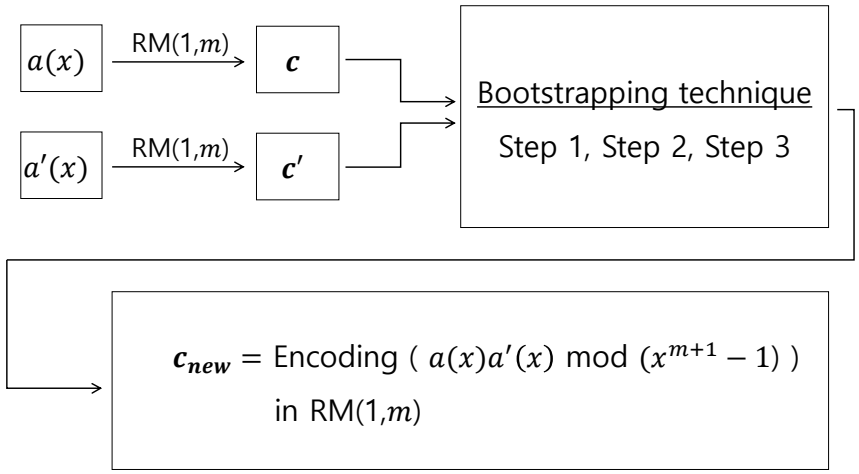

Figure 3.1: Bootstrapping process in the first-order RM codes.

abused. The bootstrapping means resetting or re-booting the order of the RM codes, here. I can do this as many times as necessary to finish the arbitrary homomorphic computations. Notably, the above three steps can be combined into an $(n+m)\times n$ linear transformation $\boldsymbol{VXG}$. To perform Steps 1 and 2, I need the following theorem and corollary.

Theorem 1. *In the first-order RM code, RM(1,m), I have*

$$c_{n-1-\sum_{i=1}^{m}\alpha_i 2^{i-1}} = a_0 + \sum_{i=1}^{m}\alpha_i a_i,$$

where the transpose of $(\alpha_0, \alpha_1, \cdots, \alpha_m)$ *denotes the* p*-th column of* $\boldsymbol{G}$ *with* $p = n - 1 - \sum_{i=1}^{m}\alpha_i 2^{i-1}$.

Proof: From (2.2), I have

$$\begin{aligned} c_p &= (a_0, a_1, \cdots, a_m)(p\text{-th column of } G) \\ &= \sum_{i=0}^{m} a_i g_{ip}, \end{aligned}$$

where g_{ip} denotes the (i,p) element of $\boldsymbol{G}$. Clearly, the first row of $\boldsymbol{G}$ is all-one vector,

that is, α_0 = 1, and thus every c_p includes a_0. Then, for the remaining rows of $\boldsymbol{G}$, I should add a_i if the $(i+1)$-th component of p-th column of $\boldsymbol{G}$ is '1'.

It can be observed that the p-th column of the generator matrix of RM(1,m) except the first row, is the one's complement of the binary representation of p. Let $(1, \alpha_1, \alpha_2, \cdots, \alpha_m)^T$ be the p-th column of $\boldsymbol{G}$. Then, I have $p = 2^m - 1 - \sum_{i=1}^{m} \alpha_i 2^{i-1}$. Thus, the theorem is proved.

□

From Theorem 1, it is straightforward to obtain the following corollary.

Corollary 1. *In the first-order RM code, RM(1,m), I have*

$$c_0 + c_{2^{i-1}} = a_i, \ i = 1, 2, \cdots, m.$$

□

Using the above theorem and corollary, the three steps of bootstrapping are explained in detail as follows.

Step 1: Coefficient mapping from $\boldsymbol{c}$ and $\boldsymbol{c}'$ to the coef($\boldsymbol{v_i v_j}$)

Here, I will represent the coefficients of $\boldsymbol{v_i v_j}$ in (3.3) by using the components of the codewords, $\boldsymbol{c} = (c_0, c_1, \cdots, c_{n-1})$ and $\boldsymbol{c}' = (c'_0, c'_1, \cdots, c'_{n-1})$ as follows. Thus, it is denoted as

$$\text{coef}(\boldsymbol{v_i v_j}) = f_{ij}(c_0, c_1, \cdots, c_{n-1}, c'_0, c'_1, \cdots, c'_{n-1}).$$

The function f_{ij} can be determined by considering the following four cases with variations of i and j.

Case 1-1) $i \neq j$ and $i, j \neq 0$:

From (3.3), the coefficient of $\boldsymbol{v_i v_j}$ becomes $a_i a'_j + a_j a'_i$. I can express this as

$$\begin{aligned}\text{coef}(\boldsymbol{v_i v_j}) &= a_i a'_j + a_j a'_i \\ &= (a_0 + a_i + a_j)(a'_0 + a'_i + a'_j) \\ &\quad + (a_0 + a_i)(a'_0 + a'_i) + (a_0 + a_j)(a'_0 + a'_j) \\ &\quad + a_0 a'_0\end{aligned}$$

and from Theorem 1, I have

$$\begin{aligned}\text{coef}(\boldsymbol{v_i v_j}) = c_{n-1-2^{i-1}-2^{j-1}} c'_{n-1-2^{i-1}-2^{j-1}} \\ + c_{n-1-2^{i-1}} c'_{n-1-2^{i-1}} \\ + c_{n-1-2^{j-1}} c'_{n-1-2^{j-1}} \\ + c_{n-1} c'_{n-1}.\end{aligned}$$

Case 1-2) $i \neq 0$, $j = 0$:

From (3.3), the coefficient of $\boldsymbol{v_i v_0} = \boldsymbol{v_i}$ becomes $a_i a'_0 + a_0 a'_i$. I can express this as

$$\begin{aligned}\text{coef}(\boldsymbol{v_i}) &= a_i a'_0 + a_0 a'_i \\ &= (a_0 + a_i)(a'_0 + a'_i) \\ &\quad + a_0 a'_0 \\ &\quad + a_i a'_i\end{aligned}$$

and from Theorem 1 and Corollary 1, I have

$$\begin{aligned}\text{coef}(\boldsymbol{v_i}) = c_{n-1-2^{i-1}} c'_{n-1-2^{i-1}} \\ + c_{n-1} c'_{n-1} \\ + (c_0 + c_{2^{i-1}})(c'_0 + c'_{2^{i-1}}).\end{aligned}$$

This case is very similar to Case 1-1), but I cannot use Theorem 1 when $j = 0$. Thus, I separate this from Case 1-1) and use Corollary 1.

Case 2-1) $i = j \neq 0$*:*

From (3.3), I can express the coefficient as

$$\text{coef}(\boldsymbol{v_i}^2) = a_i a'_i$$

and from Corollary 1, I have

$$(c_0 + c_{2^{i-1}})(c'_0 + c'_{2^{i-1}}).$$

It should be noted that the Boolean function $\boldsymbol{v_i}^2$ is equal to $\boldsymbol{v_i}$. However, I have separated these two cases because coef($\boldsymbol{v_i}$) corresponds to coef(x^i) and coef($\boldsymbol{v_i}^2$) corresponds to coef($x^{2i \bmod (m+1)}$).

Case 2-2) $i = j = 0$*:*

From (3.3), I can express the coefficient as

$$\text{coef}(\boldsymbol{v_0}) = a_0 a'_0$$

and from Theorem 1, I have

$$c_{n-1} c'_{n-1}.$$

By merging the above four cases, I can make a binary $(n+m) \times (k_2+m)$ matrix $\boldsymbol{V}$, which is a linear transformation from

$$(c_0 c'_0, c_1 c'_1, \cdots, c_{n-1} c'_{n-1}, (c_0 + c_{2^0})(c'_0 + c'_{2^0}),$$
$$(c_0 + c_{2^1})(c'_0 + c'_{2^1}), \cdots, (c_0 + c_{2^{m-1}})(c'_0 + c'_{2^{m-1}}))$$

to

$$(\text{coef}(\boldsymbol{v_0}), \text{coef}(\boldsymbol{v_1}), \cdots \text{coef}(\boldsymbol{v_m}),$$
$$\text{coef}(\boldsymbol{v_1 v_2}), \text{coef}(\boldsymbol{v_1 v_3}), \cdots \text{coef}(\boldsymbol{v_{m-1} v_m}),$$
$$\text{coef}(\boldsymbol{v_1}^2), \text{coef}(\boldsymbol{v_2}^2), \cdots, \text{coef}(\boldsymbol{v_m}^2)).$$

Step 2: Message mapping

I will derive the coefficient $\sum_{i+j=l \bmod (m+1)} a_i a'_j$ of x^l in (3.1) with a function g_l as

$$\begin{aligned} \text{coef}(x^l) &= \sum_{i+j=l \bmod (m+1)} a_i a'_j \\ &= g_l(\text{coef}(\boldsymbol{v_i v_j})). \end{aligned} \tag{3.4}$$

I can determine g_l as given in the following theorem.

Theorem 2. *In the first-order RM code, RM(1,m), coef(x^l) in (3.1) is given as*

$$\textit{coef}(x^l) = \sum_{i+j=l \ \textit{mod}(m+1)} \textit{coef}(\boldsymbol{v_i v_j}).$$

Proof: It can be easily observed that (3.3) can be rewritten as

$$\sum_{l=0}^{m} \Big(\sum_{i+j=l \bmod (m+1)} a_i a'_j \boldsymbol{v_i v_j} \Big). \tag{3.5}$$

Focusing on the coefficients of $\boldsymbol{v_i v_j}$ in (3.5), when $i + j = l \bmod (m + 1)$, the sum of coef($\boldsymbol{v_i v_j}$) is the sum of $a_i a'_j$. And this result equals the coefficients of x^l also from (3.4). Thus, the theorem is proved. □

Now, by using Theorem 2, I can perform a linear transformation from

$$\begin{aligned} &(\text{coef}(\boldsymbol{v_0}), \text{coef}(\boldsymbol{v_1}), \cdots, \text{coef}(\boldsymbol{v_m}), \\ &\quad \text{coef}(\boldsymbol{v_1 v_2}), \text{coef}(\boldsymbol{v_1 v_3}), \cdots, \text{coef}(\boldsymbol{v_{m-1} v_m}), \\ &\quad \text{coef}(\boldsymbol{v_1}^2), \text{coef}(\boldsymbol{v_2}^2), \cdots, \text{coef}(\boldsymbol{v_m}^2)) \end{aligned}$$

to

$$(\text{coef}(x^0), \text{coef}(x^1), \cdots, \text{coef}(x^m))$$

denoted by $(k_2 + m) \times k$ matrix $\boldsymbol{X}$, where $k_2 = \binom{m}{0} + \binom{m}{1} + \binom{m}{2}$.

After merging Steps 1 and 2, I obtain

$$\text{coef}(x^l) = g_l(f_{ij}(c_0, c_1, \cdots, c_{n-1}, c'_0, c'_1, \cdots, c'_{n-1})).$$

Step 3: Re-encoding of RM(1,m)

Here, the $k \times n$ generator matrix $\boldsymbol{G}$ is multiplied to obtain a new codeword of the first-order RM code that corresponds to the resulting codeword obtained for the multiplication of the two messages, $a(x)a'(x)$ mod $x^{m+1} - 1$.

Combining Steps 1–3

In the case of addition, the codeword of the message $a(x) + a'(x)$ is $\boldsymbol{c} \oplus \boldsymbol{c}'$. However, for multiplication, it is divided into three steps called bootstrapping.

Let

$$\boldsymbol{z} = (c_0 c'_0, c_1 c'_1, \cdots, c_{n-1} c'_{n-1}, (c_0 + c_{2^0})(c'_0 + c'_{2^0}),$$
$$(c_0 + c_{2^1})(c'_0 + c'_{2^1}), \cdots, (c_0 + c_{2^{m-1}})(c'_0 + c'_{2^{m-1}})).$$

Then, the new codeword $\boldsymbol{c_{new}}$ in RM(1,m), corresponding to $a(x)a'(x)$ mod $x^{m+1} - 1$, is given as

$$\boldsymbol{c_{new}} = \boldsymbol{z} \cdot \boldsymbol{V} \cdot \boldsymbol{X} \cdot \boldsymbol{G}.$$

Finally, I can determine the codeword $\boldsymbol{c_{new}}$ in RM(1,m), corresponding to the message $a(x)a'(x)$ mod $x^{m+1} - 1$, by using $\boldsymbol{c}$, $\boldsymbol{c}'$, and the $(n + m) \times n$ matrix, $\boldsymbol{T} = \boldsymbol{V} \cdot \boldsymbol{X} \cdot \boldsymbol{G}$.

Note: In fact, there exist a few all-zero rows in $\boldsymbol{V}$ for some row indices. This is because I do not use every case listed in Theorem 1 in Step 1. Only

$$c_{n-1-2^{i-1}-2^{j-1}} c'_{n-1-2^{i-1}-2^{j-1}}, c_{n-1-2^{i-1}} c'_{n-1-2^{i-1}}, c_{n-1-2^{j-1}} c'_{n-1-2^{j-1}},$$

and $c_{n-1}c'_{n-1}$ are used for Cases 1-1), 1-2), 2-1), and 2-2). Thus, for the rest of c_i and c'_i, I have all-zero rows in $\boldsymbol{V}$ and I add $(c_0 + c_{2^0})(c'_0 + c'_{2^0}), (c_0 + c_{2^1})(c'_0 + c'_{2^1}), \cdots,$ and $(c_0 + c_{2^{m-1}})(c'_0 + c'_{2^{m-1}})$ to the last m elements in $\boldsymbol{z}$.

3.3 Example in RM(1,4)

To help understand the homomorphic computation of the first-order RM codes, RM(1,m), I present an example of an RM(1,4) code. First, I determine the matrix as

$$\boldsymbol{V} = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&1&0&1&0&1&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&1&1&0&0&0&0&0\\
0&1&0&0&0&1&1&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}.$$

This matrix is obtained by merging Cases 1-1), 1-2), 2-1), and 2-2), whose size is 20×15. In this example, the 0, 1, 2, 4, and eighth rows are '0's.

Then, in Step 2, I can obtain the relations between coef(x^l) and coef$(\boldsymbol{v_i v_j})$ as

$$\begin{aligned}
\text{coef}(x^0) &= \text{coef}(\boldsymbol{v_0}^2) + \text{coef}(\boldsymbol{v_1 v_4}) + \text{coef}(\boldsymbol{v_2 v_3}) \\
\text{coef}(x^1) &= \text{coef}(\boldsymbol{v_0 v_1}) + \text{coef}(\boldsymbol{v_2 v_4}) + \text{coef}(\boldsymbol{v_3}^2) \\
\text{coef}(x^2) &= \text{coef}(\boldsymbol{v_0 v_2}) + \text{coef}(\boldsymbol{v_1}^2) + \text{coef}(\boldsymbol{v_3 v_4}) \\
\text{coef}(x^3) &= \text{coef}(\boldsymbol{v_0 v_3}) + \text{coef}(\boldsymbol{v_1 v_2}) + \text{coef}(\boldsymbol{v_4}^2) \\
\text{coef}(x^4) &= \text{coef}(\boldsymbol{v_0 v_4}) + \text{coef}(\boldsymbol{v_1 v_3}) + \text{coef}(\boldsymbol{v_2}^2).
\end{aligned}$$

From Theorem 2, the corresponding matrix $\boldsymbol{X}$ is given as

$$\boldsymbol{X} = \begin{pmatrix}
1\,0\,0\,0\,0 \\
0\,1\,0\,0\,0 \\
0\,0\,1\,0\,0 \\
0\,0\,0\,1\,0 \\
0\,0\,0\,0\,1 \\
0\,0\,0\,1\,0 \\
0\,0\,0\,0\,1 \\
1\,0\,0\,0\,0 \\
1\,0\,0\,0\,0 \\
0\,1\,0\,0\,0 \\
0\,0\,1\,0\,0 \\
0\,0\,1\,0\,0 \\
0\,0\,0\,0\,1 \\
0\,1\,0\,0\,0 \\
0\,0\,0\,1\,0
\end{pmatrix},$$

where the matrix size is 15×5. For Step 3, I obtain the 5×16 generator matrix $\boldsymbol{G}$

from (1) as

$$\boldsymbol{G} = \begin{pmatrix} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\ 1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 \\ 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\ 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \end{pmatrix}.$$

Then, by multiplying these three matrices, I obtain a matrix $\boldsymbol{T}$ with the size of 20×16 as

$$\boldsymbol{T} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&1&1&0&0&1&1&0&1&0&0&1&1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&0&0&1&1&0&0&0&0&1&1 \\ 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&1&1&0&1&0&0&1&0&1&1&0&1&0&0&1 \\ 0&1&0&1&1&0&1&0&1&0&1&0&0&1&0&1 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0 \\ 0&0&1&1&0&0&1&1&1&1&0&0&1&1&0&0 \\ 0&1&0&1&1&0&1&0&0&1&0&1&1&0&1&0 \\ 0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0 \end{pmatrix}.$$

Now, I obtain a new codeword $\boldsymbol{c_{new}}$ of the first-order RM code by multiplying $\boldsymbol{T}$ on the right of the vector $\boldsymbol{z}$. Thus, $\boldsymbol{c_{new}}$ corresponds to the codeword of the message multiplication polynomial $a(x)a'(x)$ mod $x^5 - 1$ in RM(1,4), as given in Table 3.2, where I have presented three examples.

Table 3.2: Examples with RM(1,4) codes

	Example 1)	Example 2)	Example 3)
$\boldsymbol{a}$	(01000)	(00010)	(00011)
$\boldsymbol{a'}$	(00010)	(00001)	(10101)
$\boldsymbol{c}$	(1010101010101010)	(1111000011110000)	(0000111111110000)
$\boldsymbol{c'}$	(1111000011110000)	(1111111100000000)	(1100110000110011)
$\boldsymbol{a_{pm}}$	(00001)	(00100)	(11101)
$\boldsymbol{z}$	(1010000010100000 0000)	(1111000000000000 0000)	(0000110000110000 0001)
$\boldsymbol{a_{new}}$	(00001)	(00100)	(11101)
$\boldsymbol{c_{new}}$	(1111111100000000)	(1100110011001100)	(1010010110100101)

For message vectors $\boldsymbol{a}$ and $\boldsymbol{a'}$, I have $\boldsymbol{c}$ and $\boldsymbol{c'}$. Then, I determine a vector $\boldsymbol{z}$ and perform Steps 1 and 2, which are the matrices $\boldsymbol{V}$ and $\boldsymbol{X}$. Thus, I obtain a new message vector "$\boldsymbol{a_{new}}$" and this corresponds to the polynomial multiplication of two messages, $a(x)a'(x)$ mod $x^5 - 1$, also denoted as "$\boldsymbol{a_{pm}}$." Finally, I obtain a new codeword $\boldsymbol{c_{new}}$ of the first-order RM code by multiplying the generator matrix $\boldsymbol{G}$.

3.4 Other Trials of Making Homomorphic Encryption in Code-Based Cryptosystems

3.4.1 Homomorphic Computation in Reed-Muller Codes with Error

Considering the homomorphic computation method in RM codes of the previous sections, if there exist errors on codewords, I can think of the following three trials to solve this problem.

First, I can gain z' and c'_{new} directly from $c+e$ and $c'+e'$ as Figure 3.2. If c'_{new} can be decoded to a_{new}, the homomorphism is possible. However, c'_{new} can be decoded to a message, which is not a_{new}.

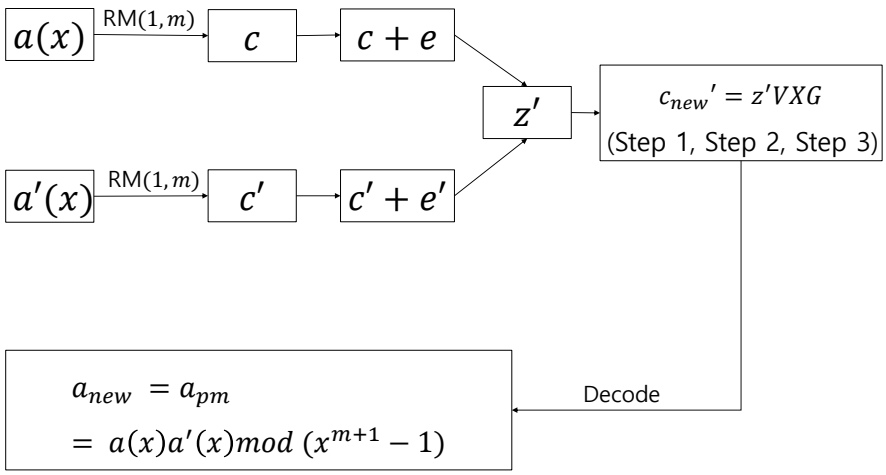

Figure 3.2: Trial on the bootstrapping process in the first-order RM codes with errors.

Secondly, I can consider a method of decoding the multiplication of $c + e$ and $c' + e'$ and matching this to z of Section 3.2. Then from z, I can make the same process as Figure 3.1. The multiplication of $c + e$ and $c' + e'$ makes cc' and $ce' + c'e + ee'$. cc' is the original codeword multiplication and $ce' + c'e + ee'$ is the error parts. When $ce' + c'e + ee'$ has a smaller Hamming weight than the error correcting capability, it is decodable with RM$(2, m)$-decoding. The error correcting capability t is $\left\lfloor \frac{2^{m-2}-1}{2} \right\rfloor$. It

is possible to gain the former part of z from cc', but it is hard to gain the latter part of z.

At last, I can decode $c + e$ and $c' + e'$, respectively by using RM$(1, m)$-decoding. The error correcting capability t is $\left\lfloor \frac{2^{m-1}-1}{2} \right\rfloor$. Then I can apply the same bootstrapping technique as Figure 3.1. This is the only way to make homomorphism possible with errors.

On the other hand, for the addition process, the addition of codewords is another codeword. However, homomorphism is only possible until the added errors are in the error-correcting range.

In the process of data transfer, errors can occur at any time. Thus, countermeasures to errors and knowing the possible correction range are important. For further work, error-tolerant homomorphic computation is needed not only for the first-order RM codes but also for higher-order RM codes.

3.4.2 Hybrid McEliece Cryptosystem Using Constant Weight Constant Distance Codes

Generally, in McEliece public key encryption, the decoding algorithm of the codes, scrambling matrix, and permutation matrix are set to be secret keys. The randomized generator matrix obtained from these matrices becomes the public key and the ciphertext is obtained from the addition of the codeword and error. Here, the codeword is a multiplication of the message and randomized generator matrix. The error is a random vector with Hamming weight of t.

The hybrid McEliece cryptosystem is a modification of the original McEliece cryptosystem. For difference, the systematic generator matrix is used and the error is constructed from the message by using a function $\phi_{n,t}$ [54]. The algorithm of the hybrid McEliece cryptosystem is depicted as Algorithm 3. I start with two integers m and t to set the length and dimension of binary Goppa codes. The length is 2^m and the dimension is $n - mt$. In a key generation, I set the public key as a $k \times (n - k)$ binary

matrix R. $(I|R)$ is a generator matrix of binary Goppa codes $\boldsymbol{G}$. Ψ_G, which is the decoder of G, becomes a secret key. In encryption, I encrypt a plaintext (m_1, m_2) to a ciphertext y. $\phi_{n,t}$ is a mapping from a length l-vector to a length n-vector, which has a Hamming weight of t. In decryption, I decode y to a codeword x using the decoder Ψ_G. Then I match this to the plaintext (m_1, m_2). m_1 is the same as x_1 and m_2 is equal to $\phi_{n,t}^{-1}(y - x)$.

Algorithm 3 Hybrid McEliece public key encryption scheme [54]

System parameters :

Two integers m and t : Let the length $n = 2^m$ and the dimension $k = n - mt$

Key Generation :

Public key : A $k \times (n-k)$ binary matrix R such that $(I|R)$ is a generator matrix of binary Goppa codes $\boldsymbol{G}$

Secret key : Decoder Ψ_G

Encryption : $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^n$

$(m_1, m_2) \mapsto y = (m_1|m_2R) + \phi_{n,t}(m_2)$

where $\phi_{n,t}$ is a mapping from $\{0,1\}^l$ to $\{0,1\}^n$ with Hamming weight t.

Decryption : $\{0,1\}^n \to \{0,1\}^n \to \{0,1\}^k \times \{0,1\}^l$

$y \mapsto \Psi_G(y) = x = (x_1, x_1R) \mapsto (m_1, m_2) = (x_1, \phi_{n,t}^{-1}(y - x))$

where x is a codeword obtained from y using Ψ_G.

The point is that if I use constant weight constant distance codes as $\phi_{n,t}$, the scheme becomes an additively homomorphic encryption scheme. It is because the error made after the addition of ciphertexts is correctable.

For example, using (7,4)-Hamming codes, the additive homomorphism is possible. The codewords with Hamming weight 4 in (7,4)-Hamming codes are given as

$$c_1 = 1001011$$

$$c_2 = 0010111$$

$$c_3 = 0101110$$

$$c_4 = 1011100$$

$$c_5 = 0111001$$

$$c_6 = 1110010$$

$$c_7 = 1100101.$$

Note that all of these seven codewords have Hamming weight of 4 and also every two of these have a distance of 4. Also, they have a special characteristic that a codeword is a cyclic shift of another codeword. These codes are called cyclic difference sets (CDS). Four examples of CDSs are shown in Table 3.3. v implements the number of whole possible indices, k is the number of codewords for one index, and λ means the number of the same indices of two codewords. Let CDS be the set of indices, n be the length, w be the weight, and d be the distance of the sets. These have relationship of $n = v, w = k$, and $d = 2(k - \lambda)$. I can see that these are all constant weight constant distance codes. The first example is the (7,4)-Hamming code with weight 4 as I explained above.

Table 3.3: The examples of CDSs

(v, k, λ)	CDS	n	w	d
(7,4,2)	{0,3,5,6}	7	4	4
(21,5,1)	{3,6,7,12,14}	21	5	8
(31,6,1)	{6,8,9,14,26,30}	31	6	10
(40,13,4)	{1,2,3,5,6,9,14,15,18,20,25,27,35}	40	13	18

Similarly, balanced incomplete block designs (BIBD) can be used. $(t-v, b, r, k, \lambda)$ BIBD is composed of v elements, b blocks, number of blocks for one point r, number

of points in a block k, and number of blocks for t (usually $t = 2$) points λ [55]. The notations are confused with the parameters of CDSs. Considering constant weight constant distance codes, it satisfies $n = b, w = r$, and $d = 2(r-\lambda)$. Three examples of BIBDs are shown in Table 3.4. Also, Table 3.5 is the specification of the first example, 2-(6,10,5,3,2) BIBD.

Table 3.4: The examples of BIBDs

v	b	r	k	λ	n	w	d
6	10	5	3	2	10	5	6
51	425	25	3	1	425	25	48
105	546	26	5	1	546	26	50

Table 3.5: 2-(6,10,5,3,2) BIBD

1	1	1	1	1					
1	1				1	1	1		
1		1			1			1	1
	1		1			1		1	1
		1		1		1	1	1	
			1	1	1		1		1

Thus, it is possible to make constant weight constant distance codes using CDSs and BIBDs. That means in a hybrid McEliece cryptosystem, if I use these CDSs and BIBDs as $\phi_{n,t}$, I can correct errors even after the addition. However, it is hard to match the desired n, w, and d values exactly. Also, it is hard to achieve homomorphism for twice or more addition. These are further works to be solved.

3.4.3 Additive Homomorphism on Niederreiter Cryptosystem with Fixed Errors

Niederreiter public key encryption scheme is a dual version of the McEliece public key encryption scheme. The NIST PQC round 4 algorithm *Classic McEliece* is based on this scheme. It is specified as Algorithm 4. For key generation, I construct $\boldsymbol{H_{pub}}$ by multiplying $\boldsymbol{S}$, $\boldsymbol{H}$, and $\boldsymbol{P}$. $\boldsymbol{H}$ is the parity check matrix of binary Goppa codes, which is made from the generator matrix $\boldsymbol{G}$. $\boldsymbol{S}$ and $\boldsymbol{P}$ are random invertible matrices. The public key is $\boldsymbol{H_{pub}}$ and the secret keys are $\boldsymbol{S}$, $\boldsymbol{H}$, and $\boldsymbol{P}$. For encryption, I compute the ciphertext $\boldsymbol{c}$ from the message $\boldsymbol{e}$ by multiplying $\boldsymbol{H_{pub}}$ on the left side. For decryption, I compute $\boldsymbol{c'}$ from multiplying the inverse of $\boldsymbol{S}$ to $\boldsymbol{c}$. Then, I decode $\boldsymbol{c'}$ to $\boldsymbol{e'}$ using the decoding algorithm of binary Goppa codes. At last, I return $\hat{\boldsymbol{e}}$ by multiplying the inverse of $\boldsymbol{P}$ to $\boldsymbol{e'}$. If $\hat{\boldsymbol{e}}$ is the same as the original message $\boldsymbol{e}$, the decryption is successful.

If the fixed errors are used, homomorphic addition in the Niederreiter scheme is partially possible. First, I use a constant weight constant distance code as

$$\boldsymbol{G'} = \begin{pmatrix} 1110010 \\ 1100101 \\ 0010111 \\ 0111001 \\ 1001011 \\ 1011100 \\ 0101110 \end{pmatrix}. \tag{3.6}$$

$\boldsymbol{G'}$ is obtained from the codewords with Hamming weight 4 in (7,4)-Hamming codes. Then, using scrambling matrices $\boldsymbol{S_{ij}}$, I obtain a matrix $\boldsymbol{D}$ as

Algorithm 4 Niederreiter public key encryption scheme [45]

Key Generation :

$\boldsymbol{G}$: $k \times n$ generator matrix of binary Goppa codes

t: Error-correcting capability of the binary Goppa codes

$\boldsymbol{H}$: $(n-k) \times n$ parity check matrix from $\boldsymbol{G}$

$\boldsymbol{S} \xleftarrow{\$} F_2^{(n-k)\times(n-k)}, \boldsymbol{P} \xleftarrow{\$} F_2^{n\times n}$

$\boldsymbol{H_{pub}} \leftarrow \boldsymbol{SHP}$

Public key: $\boldsymbol{H_{pub}}$

Secret key: $\boldsymbol{S}, \boldsymbol{H}, \boldsymbol{P}$

Encryption :

Compute ciphertext from message $\boldsymbol{e}$:

$\boldsymbol{c} \leftarrow \boldsymbol{H_{pub}e}$

Decryption :

Compute $\boldsymbol{c'} \leftarrow \boldsymbol{S^{-1}c}$

Decode $\boldsymbol{c'}$ to $\boldsymbol{e'}$ using the decoding algorithm of the Goppa codes

Return $\hat{\boldsymbol{e}} \leftarrow \boldsymbol{P^{-1}e'}$

$$\boldsymbol{D} = \begin{pmatrix} \boldsymbol{S_{11}G'} \cdots \boldsymbol{S_{1a}G'} \\ \ddots \\ \boldsymbol{S_{a1}G'} \cdots \boldsymbol{S_{aa}G'} \end{pmatrix}, \tag{3.7}$$

where i and j are integers from 1 to a. At last, I obtain a matrix $\boldsymbol{E}$ as

$$\boldsymbol{E} = \left(\boldsymbol{D}|\boldsymbol{L}\right)\boldsymbol{P'}, \tag{3.8}$$

by concatenating a matrix $\boldsymbol{L}$ and multiplying a permutation matrix $\boldsymbol{P'}$. Every row in $\boldsymbol{L}$ has Hamming weight of b.

If the errors are fixed with the rows of $\boldsymbol{E}$, the addition is possible at least once. The Hamming weights of errors before addition are constant as $4a + b$ and these become $4a + 2b$ after adding once. Thus, the decryption is possible until the Hamming weights of errors are smaller than the error correcting capability t.

There are five parameters for *Classic McEliece*. By using the proposed method, I can set the parameters as in Table 3.6 for one-time addition. I set the values of a, b, w, and s to make n, k, w', and s' after one-time addition. n, k, w', and s' are the original parameters of *Classic McEliece*. n is the length of the codes and k is the dimension. w' is the maximum Hamming weight of each row after addition, $\hat{w}$ is the maximum Hamming weight without using $\boldsymbol{E}$, and w is the maximum Hamming weight before addition. s' is the security level after addition and s is the security level before addition. s is obtained from the value of cw, which is an approximate value of the security level in *Classic McEliece* [38]. c is approximate value of $-\log_2(1 - k/n)$. The values of s are set among 128, 192, and 256-bit security and these have smaller values than cw's. As a result, the addition is impossible for (1), however, it is possible for (2)–(5) by reducing the security levels as in Table 3.6.

Table 3.6: Parameter change of *Classic McEliece* after one-time addition

	n	k	w'	s'	a	b	$\hat{w}$	w	cw	s
(1)	3488	2720	64	128	9	14	32	50	109.16	x
(2)	4608	3360	96	192	13	22	48	74	139.45	128
(3)	6688	5024	128	256	18	28	64	100	200.69	192
(4)	6960	5413	119	256	15	29	59	89	193.10	192
(5)	8192	6528	128	256	18	28	64	100	229.96	192

Chapter 4

Improving Key Size and Bit-Security of *Modified pqsigRM*

In this chapter, first I explain the details of *Modified pqsigRM*. Then, I present *Improved Modified pqsigRM*, which improves the key size of *Modified pqsigRM*, by making a public key into a systematic form and choosing new parameters for the keys. I show how to make the public key in a systematic form. I also present the new parameter sets, the expected strength of them, and the design rationale. Then, the improved key sizes after choosing new parameters are introduced. Moreover, the bit-security is also shown to be improved. The comparison with other PQC schemes is explained, too. The public key and signature sizes comparing these with the proposed scheme are introduced. Also, the verification cycles compared with other NIST PQC finalist schemes are specified. Then the enhanced version of this algorithm called *Enhanced pqsigRM*, is introduced. I enhance some specifications of the algorithm considering the review of the NIST PQC organizer team. I use the systematic public key, minimize the secret key, simplify the hash function, and solve the security issues.

4.1 *Modified pqsigRM*

pqsigRM is an RM code-based digital signature scheme, which was presented in the first round of the NIST PQC conference. There are several versions of *pqsigRM*. The original *pqsigRM* used column puncturing and insertion techniques for key generation [23]. However, there is an attack finding the puncturing locations with the hull. The most recent version is *Modified pqsigRM* [24], which uses partial permutation, row appending, and replacement. There has been no valid attack on it yet.

The process of constructing the generator matrix of modified RM codes is comprised of four steps as follows [24].

Step 1: Making partially permuted RM codes

It is possible to express RM codes by applying the recursive structure. In *Modified pqsigRM*, I first partially permute the recursive generator matrix as in Figure 4.1. Then, I make the same permutation σ_1's on $\boldsymbol{G}_{RM(r,m-2)}$'s and the other permutation σ_2 on $\boldsymbol{G}_{RM(r-2,m-2)}$, respectively. σ_1 and σ_2 are independent permutations, which randomly permute p columns out of $n/4$ columns. This structure allows us to protect the codes from attack using hull.

Step 2: Replacing $\boldsymbol{G}_{RM(r,r)}$'s with random codes

When I make the recursive structure of RM codes repeatedly, I obtain $\boldsymbol{G}_{RM(r,m)}$'s on the first 2^r rows. I replace these with 2^{m-r} number of the same $(2^r, k_{rep})$-codes as in Figure 4.2. Additionally, the dual code of these $(2^r, k_{rep})$ random codes should include at least one non-zero codeword with an odd Hamming weight. This step makes the code to be robust from the attack using its dual code.

Step 3: Appending independent random codewords

In this step, I append k_{app} number of independent random codewords which have at least one non-zero codeword with an odd Hamming weight as in Figure 4.2. By doing this, I can make the codes not to be distinguished.

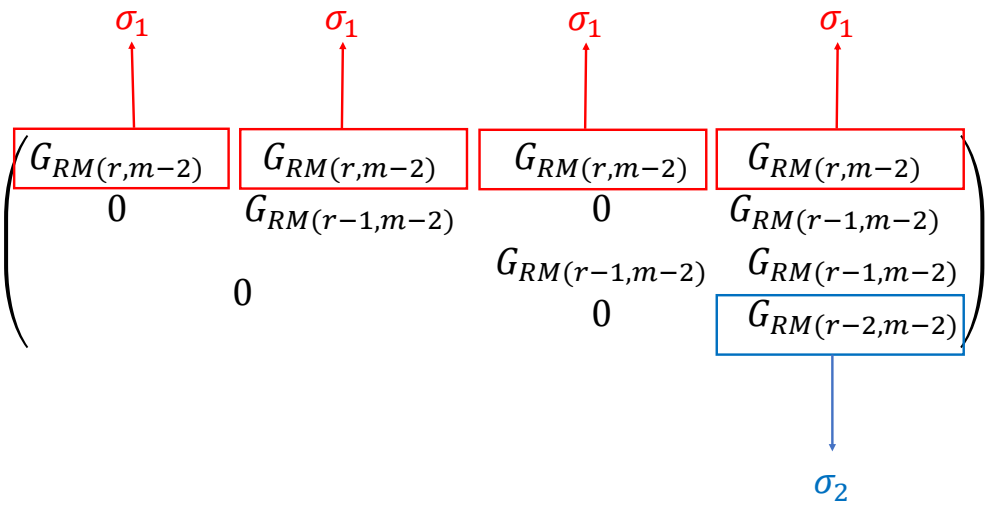

Figure 4.1: The structure of partially permuted RM codes.

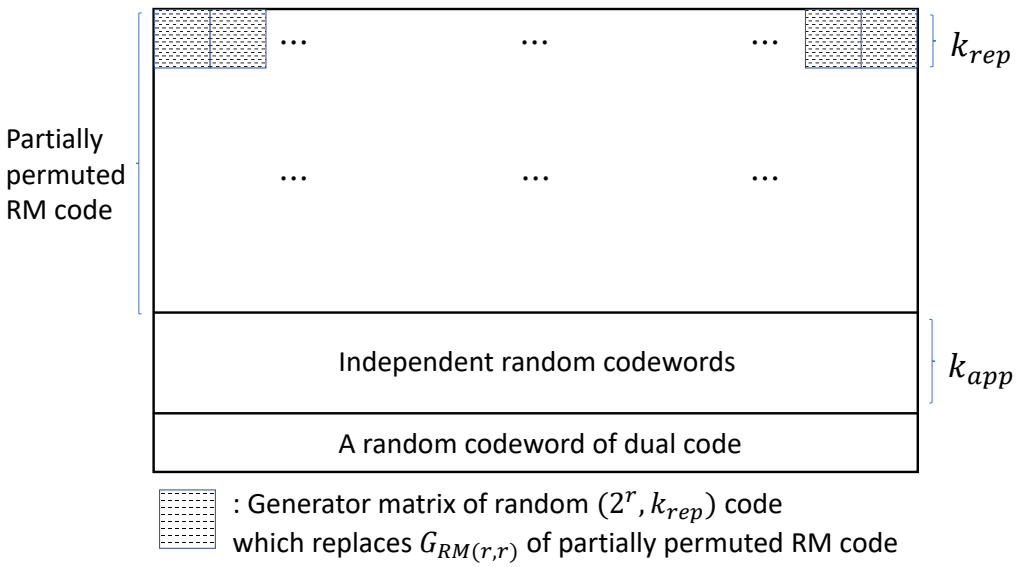

Figure 4.2: The structure of modified RM codes.

Step 4: Appending a dual code codeword

The last step is to append a codeword from the dual code of the whole code as in Figure 4.2. This step prevents the leakage of the information of the hull.

With these four steps, I can construct the modified RM codes, which are strong against all possible attacks or information leakage.

Then the modified RM codes are used in the process of the signature scheme of *Modified pqsigRM* as in Algorithm 5. *Modified pqsigRM* follows the structure of CFS digital signature scheme. In **Key Generation**, a parity check matrix of modified RM codes $\boldsymbol{H}$, random non-singular matrix $\boldsymbol{S}$, and permutation non-singular matrix $\boldsymbol{Q}$ are set. The public key becomes $\boldsymbol{H'}$, which is a multiplication of $\boldsymbol{S}$, $\boldsymbol{H}$, and $\boldsymbol{Q}$. The secret key includes matrices $\mathbf{S}, \mathbf{H}$, and $\mathbf{Q}$. Then, in **Signing** process, syndrome with a hash function h, message $\boldsymbol{m}$, and counter i are set. Then $\boldsymbol{s'}$, $\boldsymbol{e'}$, and $\boldsymbol{e}$ are computed in order. The decoding algorithm of modified RM codes is used while computing $\boldsymbol{e'}$. The signature becomes $(\boldsymbol{m}, \boldsymbol{e}, i)$. In **Verification**, the signature is verified if the Hamming weight of $\boldsymbol{e}$ is smaller than or equal to the error correcting capability w and if $\boldsymbol{H'e^T}$ is equal to $h(h(\boldsymbol{m}|\boldsymbol{H'})|i)$.

Moreover, *Modified pqsigRM* resists all known attacks against cryptosystems based on the RM codes such as Minder-Shokrollahi attack [51], Chizhov-Borodin attack [52], and square code attack [53]. Also, it resists the attacks on the hull, which were proposed against the original version of *pqsigRM*. Furthermore, they have proven the existential unforgeability under a chosen message attack (EUF-CMA) security with decoding-one-out-of-many (DOOM) problem [24].

4.2 *Improved Modified pqsigRM* : Improving Key Size and Bit-Security of *Modified pqsigRM*

Modified pqsigRM is a promising code-based PQC digital signature scheme. However, it has a problem with large public key sizes. To improve this issue, I propose a scheme

Algorithm 5 Signature scheme of *Modified pqsigRM* [24]

Key Generation :

$\boldsymbol{G}$: $k \times n$ generator matrix of modified RM codes

$\boldsymbol{H}$: $(n-k) \times n$ parity check matrix of modified RM codes

$\boldsymbol{S} \xleftarrow{\$} F_2^{(n-k)\times(n-k)}, \boldsymbol{Q} \xleftarrow{\$} F_2^{n\times n}$

$\boldsymbol{H'} \leftarrow \boldsymbol{SHQ}$

Public key: $\boldsymbol{H'}$

Secret key: $\mathbf{S}, \mathbf{H}, \mathbf{Q}$

Signing :

$\boldsymbol{m}$: Message, $i \hookleftarrow \{0,1\}^{\lambda_0}$: Counter

$\boldsymbol{s} \leftarrow h(h(\boldsymbol{m}|\boldsymbol{H'})|i)$: Syndrome

$\boldsymbol{s'}^T \leftarrow \boldsymbol{S}^{-1}\boldsymbol{s}^T$

$\boldsymbol{e'} \leftarrow$ Decode($\boldsymbol{s'}; \boldsymbol{H}$)

$\boldsymbol{e}^T \leftarrow \boldsymbol{Q}^{-1}\boldsymbol{e'}^T$

Signature: $(\boldsymbol{m}, \boldsymbol{e}, i)$

Verification :

If $wt(\boldsymbol{e}) \leq w$ and $\boldsymbol{H'e^T} = h(h(\boldsymbol{m}|\boldsymbol{H'})|i)$,
return ACCEPT

Else, return REJECT

*h: hash function with $\{0,1\}^* \rightarrow \{0,1\}^{n-k}$

*DECODE: Decoding algorithm of modified RM codes

*$wt(\boldsymbol{a})$: Hamming weight of a vector $\boldsymbol{a}$

*w: error correcting capability of modified RM codes

called *Improved Modified pqsigRM*. I reduce public key sizes to more than half of *Modified pqsigRM*. Also, I improved the exact bit-security compared with the original *Modified pqsigRM*.

4.2.1 Public Key in Systematic Form

Improved Modified pqsigRM follows the same algorithm as *Modified pqsigRM* as in Algorithm 5. The main difference is the usage of the systematic public key. According to the Algorithm 5, *Modified pqsigRM* uses a full $(n-k) \times n$ matrix $\boldsymbol{H' = SHQ}$ as a public key. However, for *Improved Modified pqsigRM*, I make this to be a systematic form as $\boldsymbol{H_{sys} = (I \mid T)}$. Then I just use $\boldsymbol{T}$ as a new public key, which has a size of $(n-k) \times k$. This process reduces the public key size considerably.

4.2.2 Expected Strength for Each Parameter Set

1) Parameter Set Imp-pqsigRM-711

Security level satisfies 2^{80}. It uses (7,11)-modified RM codes with $w = 55, p \geq 140, k_{rep} = 126$, and $k_{app} = 2$.

2) Parameter Set Imp-pqsigRM-412

Security level satisfies 2^{128}, which is the NIST PQC security category 1. It uses (4,12)-modified RM codes with $w = 1,280, p \geq 80, k_{rep} = 14$, and $k_{app} = 2$.

3) Parameter Set Imp-pqsigRM-413

Security level satisfies 2^{256}, which is the NIST PQC security category 5. It uses (4,13)-modified RM codes with $w = 2,900, p \geq 80, k_{rep} = 14$, and $k_{app} = 2$.

4.2.3 Design Rationale

1) Choosing the Parameter Sets

To choose proper parameter sets, I follow the method of *Modified pqsigRM*. First, I choose r and m for the parameters. n and k are determined by these as 2^m and $\sum_{i=0}^{r} \binom{m}{i}$, respectively. From n and k, I can numerically obtain the possible range of Hamming weight of error, which is w. I can compute the security level by the work factor of decoding one out of many (DOOM) problem. I control it by changing n and w. If n is large, the security level increases, however, the key sizes get larger, too. If w is small, the security level also increases, but the number of decoding iterations must be larger, too. I can also control the decoding iterations by reducing the partial permutation number p. However, if p is too small, it becomes similar to the original RM codes. I found the lower bound of p, which does not reduce the randomness of the hull.

If the modified RM codes' hull has the same part as the original RM codes, there exists a security problem. Then, the minimum Hamming weight codeword of the original RM codes can be included in the hull and this allows the Minder-Shokrollahi attack [51]. To avoid this, the modified RM codes' hull should not be the subset of the original RM codes. It means that $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ should be almost the same as the hull. $\mathcal{C}_{pub}$ means the set of the modified RM codes and $\setminus$ means the relative complement [24].

The average dimensions of $hull(\mathcal{C}_{pub})$ and $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ for full permutation ($p = n/4$) are given as Table 4.1. Choosing p without security loss, the dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ should have a similar value as that of full permutation. To find the smallest p, which has a small Hamming weight of errors and does not affect the dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$, I proceeded some numerical experiments. As a result for example, the dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(4,12)}$ for 128-bit security parameters is shown in Figure 4.3. The dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ saturates on

certain value of p. In the same way, p is determined larger than or equal to 140, 80, and 80 for 80, 128, and 256 security levels, respectively.

Moreover, $\mathbf{H}$ can be represented with $\sigma_p^1, \sigma_p^2, k_{rep} = 2^r - 2$ (the maximum value), and $k_{app} = 2$(the minimum value). Therefore, I obtain the values of n, k, w, p, k_{rep}, and k_{app} of *Improved Modified pqsigRM* for each security level as Table 4.2.

Table 4.1: The dimensions of $hull(\mathcal{C}_{pub})$ and $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ for $p = n/4$

(r, m)	(7,11)	(4,12)	(4,13)
n	2,048	4,096	8,192
k	1,817	795	1,094
$dim(hull(\mathcal{C}_{pub}))$	214	684	961
$dim(hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)})$	140	80	80

2) Computing the Sizes of the Public Keys, Secret Keys, and Signatures

As mentioned in the previous subsection, the public keys are obtained from a systematic form of parity check matrix and it requires $(n-k)k$ bits. These are 0.05, 0.31, and 0.93 MB for `Imp-pqsigRM-711`, `Imp-pqsigRM-412`, and `Imp-pqsigRM-413`, respectively.

For the secret keys, if I only take the necessary information, it includes a matrix $\mathbf{Q}$, partial permutations σ_p^1, σ_p^2, $k_{rep} \times 2^r$ repeated replacing codes, $k_{app} \times n$ appending codes, and $1 \times n$ padding codeword of dual code. $\mathbf{Q}$ is an $n \times n$ permutation matrix, which can be expressed with just a number. I use nm bits for $\mathbf{Q}$ because I need $\log_2 n$ bits to express a number and the number is from 0 to n. In the same way, σ_p^1 and σ_p^2 need $n(m-2)/2$ bits. The replacing codes, appending codes, and padding codes need $(2^r - 2) \times 2^r, k_{app} \times n$, and $1 \times n$ bits, respectively. Thus, the sizes of the secret keys are $3nm/2 + k_{app}n + (2^r - 2)2^r$ bits. These are 6,752, 10,268, and 22,044 bytes for `Imp-pqsigRM-711`, `Imp-pqsigRM-412`, and `Imp-pqsigRM-413`,

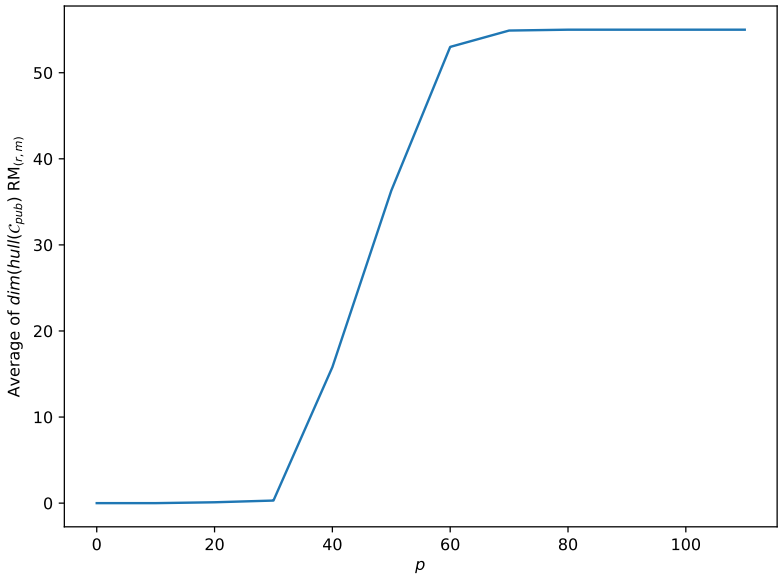

Figure 4.3: The dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(4,12)}$ for variation of p.

Table 4.2: Parameter sets of *Improved Modified pqsigRM* for each security level

λ (security)	80	128	256
(r, m)	(7,11)	(4,12)	(4,13)
n	2,048	4,096	8,192
k	1,817	795	1,094
w	55	1,280	2,900
p	$\geq$ 140	$\geq$80	$\geq$80
k_{rep}	126	14	14
k_{app}	2	2	2

respectively.

On the other hand, I need $(n+64)$ bits for the signature lengths. n is for the length of e, and 64 is for the size of a 64-bit integer i. The signature lengths are 264, 520, and 1,032 bytes for `Imp-pqsigRM-711`, `Imp-pqsigRM-412`, and `Imp-pqsigRM-413`, respectively.

4.2.4 Improving Key Size by Choosing New Parameters

Using the systematic public key, I need a new setting for parameters as in Table 4.4. Assuming n is equal, I should set k near 0 or n to make the systematic public key size small because it is $(n-k) \times k$. For RM codes, I can control k by changing r according to the following Lemma.

Lemma 1. *In RM codes for every r, r' and m, it satisfies*

$$RM(r', m) \subseteq RM(r, m)$$

where $r' \leq r$ [2].

Comparing Table 4.4 with Table 4.3, which is for the parameters of *Modified pqsigRM*, I can see the reduced parameter sizes for the same code length and similar security level. Here, w is the maximum possible Hamming weight of error, and security level means the bit-security of the signature scheme. (7,11)-*Improved Modified pqsigRM* has similar security level with (5,11)-*Modified pqsigRM* and this has 0.20 times of public key size. Additionally, (4,12) and (4,13)-*Improved Modified pqsigRM*s have larger bit-securities and smaller key sizes by 0.40 and 0.23 times, compared with (6,12) and (6,13)-*Modified pqsigRM*s, respectively.

4.2.5 Improving the Bit-Security

Moreover, to improve the bit-security of the signature scheme, I can use (5,13)-*Improved Modified pqsigRM* instead of (4,13)-*Improved Modified pqsigRM*. It remains a little bit more of a security margin but has a trade-off with the public key size.

Table 4.3: Parameters of *Modified pqsigRM*

	Modified pqsigRM		
(r,m)	(5,11)	(6,12)	(6,13)
n	2,048	4,096	8,192
k	1,025	2,511	4,097
w	325	495	1,370
Security level	80	128	256
Bit-security	83	130	259
Public key size (MB)	0.25	0.77	4.00

Table 4.4: Parameters of *Improved Modified pqsigRM*

	Improved Modified pqsigRM			
(r,m)	(7,11)	(4,12)	(4,13)	(5,13)
n	2,048	4,096	8,192	8,192
k	1,817	795	1,094	2,381
w	55	1,280	2,900	2,144
Security level	80	128	256	256
Bit-security	81	148	274	287
Public key size (MB)	0.05	0.31	0.93	1.65

For *Modified pqsigRM* [24] or *Improved Modified pqsigRM*, I obtain the exact bit-security by computing the work factor of solving the DOOM problem [56], which is given as

$$\mathrm{WF}_q^M = \min_{p,l} \left(\frac{C_q(p,l)}{\mathcal{P}_{qM}(p,l)} \right). \tag{4.1}$$

$$C_q(p,l) = \max \left(\sqrt{q\binom{k+l}{p}}, \frac{q\binom{k+l}{p}}{2^l} \right)$$

is the complexity of solving the DOOM problem using Dumer's algorithm, with the condition of $q \leq \binom{k+l}{p}$, and

$$\mathcal{P}_{qM}(p,l) = 1 - \left(1 - \frac{\binom{n-k-l}{w-p}\binom{k+l}{p}}{\binom{n}{w}} \right)^{qM}$$

is the success probability of it. q is the number of instances that the adversary has and M is $\binom{n}{w}/2^{n-k}$.

4.3 Comparison of *Improved Modified pqsigRM* with Other PQC Schemes

4.3.1 Public Key and Signature Sizes of *Improved Modified pqsigRM* Compared with Other PQC Schemes

As in Table 4.5, *Improved Modified pqsigRM* has a smaller public key and signature than *Wave* [22]. *Durandal* has the smallest parameters among these. However, the security of this scheme is not fully proven yet [25]. It also has a similar public key size with *Classic McEliece* [38], which is one of the finalists of the NIST PQC key encapsulation mechanism (KEM). As the public key size of *Classic McEliece* is acceptable for NIST, *Improved Modified pqsigRM* seems to be acceptable also.

On the other hand, let's take a look at Table 4.6, comparing with the NIST PQC finalist schemes, which are *Crystals-Dilithium* [36], *Falcon* [37], and *Sphincs+* [41].

Despite reducing the public key sizes of *Modified pqsigRM*, *Improved Modified pqsigRM* still has very large public key sizes compared with the NIST PQC finalists. However, I can see that it has the smallest signature size. Small signature size has a big advantage because I send a public key once in the signature scheme, but I should send the signature for every signing process.

Table 4.5: Comparison of *Improved Modified pqsigRM* with other code-based PQC schemes

Security level		***Improved Modified pqsigRM***	*Wave*	*Durandal*	*Classic McEliece* (KEM)
128	Public key (MB)	**0.31**	3.10	0.015	0.26
	Signature (byte)	**520**	1,647	4,060	
256	Public key (MB)	**0.93**	12.43	X	1.04
	Signature (byte)	**1,032**	3,293		

4.3.2 Verification Cycles of *Improved Modified pqsigRM* Compared with Other NIST PQC Finalist Schemes.

1) Description of Platform

The following measurements are collected using a desktop computer with an i7-8700 CPU @ 3.20GHz. Turbo Boost is disabled. This machine has 32GB of RAM. Benchmarks have run on one core of the CPU. Since the signing algorithm is a probabilistic algorithm, the number of iterations at signing varies. The following result is the average of 100 experiments.

NIST said that the "NIST PQC Reference Platform" is "an Intel x64 running Win-

Table 4.6: Comparison of *Improved Modified pqsigRM* with other NIST PQC finalist schemes

Security level		***Improved Modified pqsigRM***	*Crystals-Dilithium*	*Falcon*	*Sphincs+*
128	Public key (byte)	**310,000**	1,312	897	32
	Signature (byte)	**520**	2,420	666	7,856
256	Public key (byte)	**930,000**	2,592	1,793	64
	Signature (byte)	**1,032**	4,595	1,280	29,792

dows or Linux and supporting the GCC compiler". The proposed system is an x64 running Linux and supporting the GCC compiler. Beware, however, that different Intel CPUs can output different results.

2) Number of Cycles

The following data are CPU cycles for `Imp-pqsigRM-711`, `Imp-pqsigRM-412`, and `Imp-pqsigRM-413` at i7-8700 CPU @ 3.20GHz. The measurements compared with the finalists are given in Table 4.7. The data of the finalists are from the submitted document to NIST and these can be a little bit different because the implementation conditions are different [36, 37, 41]. However, it is almost the same as *Crystals-Dilithium*.

The verification cycles of *Improved Modified pqsigRM* is 172,669 cycles for average and 152,972 cycles for median with 128 bit-security. For average verification cycles, these are 0.53 times those of *Crystals-Dilithium*. It is the second smallest among the other NIST PQC finalist algorithms. However, these are very large for 256 bit-security. These are about ten times those of *Crystals-Dilithium*.

Table 4.7: Verification CPU cycles of *Improved Modified pqsigRM* compared with the NIST PQC finalist schemes

Security	*Improved Modified pqsigRM*	Verification cycles				
		Avg	Median	*Crystals-Dilithium*	*Falcon*	*Sphincs+*
80	`Imp-pqsigRM-711|`	69,080	66,986	-	-	-
128	`Imp-pqsigRM-412|`	172,669	152,972	327,362	82,340	308,774
256	`Imp-pqsigRM-413|`	7,363,625	6,983,222	871,609	168,498	696,980

4.4 *Enhanced pqsigRM*

Considering the attacks using the information set decoding and finding the minimum-weight codewords, I propose an enhanced version of these, called *Enhanced pqsigRM*. It enhances the security issues of the previous work, but the parameters get worse. Also, I use the systematic public key, minimize the secret key, and simplify the hash function to enhance *Modified pqsigRM*.

4.4.1 *Enhanced pqsigRM* Signature Scheme

The signature scheme of *Enhanced pqsigRM* is given as Algorithm 6. First, I maintained the systematic public key as in *Improved Modified pqsigRM*. It has a public key of $(n-k) \times k$ matrix $\mathbf{T}$ from $\boldsymbol{H_{sys}} = (\boldsymbol{I} \mid \boldsymbol{T})$, which is instead of the whole $(n-k) \times n$ matrix $\boldsymbol{H'} = \boldsymbol{SHQ}$. Second, it minimize the secret key as $\mathbf{Q}$, σ_p^1, σ_p^2, $k_{rep} \times 2^r$ (repeated) replacing codes, $k_{app} \times n$ appending codes, and $1 \times n$ padding codeword of dual code. Third, in signing, it uses the hash function once instead of twice to make syndrome $\mathbf{s}$. Fourth, I consider one more security issue, which is the complexity of finding minimum-weight codewords. However, enhancing the security issue made the parameter sets worse.

Algorithm 6 Signature scheme of *Enhanced pqsigRM*

Key Generation :

$\boldsymbol{G}$: $k \times n$ generator matrix of modified RM codes

$\boldsymbol{H}$: $(n-k) \times n$ parity check matrix of modified RM codes

$\boldsymbol{Q} \xleftarrow{\$} F_2^{n \times n}$

$\mathbf{H_{sys}} = (\mathbf{I}|\mathbf{T}) \leftarrow \boldsymbol{S_{sys}} \boldsymbol{H} \boldsymbol{Q}$

Public key: $\mathbf{T}$

Secret key: $\mathbf{Q}$, σ_p^1, σ_p^2, $k_{rep} \times 2^r$ (repeated) replacing codes, $k_{app} \times n$ appending codes, and $1 \times n$ padding codeword of dual code

Signing :

$\boldsymbol{m}$: Message, $i \hookleftarrow \{0,1\}^{\lambda_0}$: Counter

$\mathbf{s} \leftarrow h(\mathbf{m}|i)$: Syndrome

$\boldsymbol{s}'^T \leftarrow {\boldsymbol{S}_{\boldsymbol{sys}}}^{-1} \boldsymbol{s}^T$

$\boldsymbol{e}' \leftarrow$ Decode($\boldsymbol{s}'; \boldsymbol{H}$)

$\boldsymbol{e}^T \leftarrow \boldsymbol{Q}^{-1} \boldsymbol{e}'^T$

Signature: $(\boldsymbol{m}, \boldsymbol{e}, i)$

Verification :

If $wt(\boldsymbol{e}) \leq w$ and $\mathbf{H_{sys}} e^T = h(\mathbf{m}|i)$,
return ACCEPT

Else, return REJECT

*h: hash function SHAKE-128

*DECODE: Decoding algorithm of modified RM codes

*$wt(\boldsymbol{a})$: Hamming weight of a vector $\boldsymbol{a}$

*w: error correcting capability of modified RM codes

4.4.2 Parameter Set

1) Parameter Set Enh-pqsigRM-613

Security level satisfies 2^{128}, which is the NIST PQC security category 1. It uses (6,13)-modified RM codes with $w = 1,370, p \geq 572, k_{rep} = 62$, and $k_{app} = 2$.

4.4.3 Design Rationale

1) Choosing the Parameter Sets

I should select proper values of n, k, w, p, k_{rep}, and k_{app} for *Enhanced pqsigRM*. I choose these parameter sets of `Enh-pqsigRM-613` in the same way as choosing those of *Improved Modified pqsigRM* as in Section 4.2.3.

I can obtain the average dimension of $hull(\mathcal{C}_{pub})$ and $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ as Table 4.8. Also, I can see that the average of the dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ tends to increase as p increases, and it is saturated when p is above a certain value, as in Figure 4.4. Specifically, the dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ is saturated when p is approximately equal to 572. Hence, p should be set larger than or equal to 572.

Moreover, $\mathbf{H}$ can be represented by $\sigma_p^1, \sigma_p^2, k_{rep} = 2^r - 2$ (the maximum value), and $k_{app} = 2$(the minimum value). Therefore, I obtain the values of n, k, w, p, k_{rep}, and k_{app} of *Enhanced pqsigRM* for 128-bit security level as Table 4.9.

Table 4.8: The dimensions of $hull(\mathcal{C}_{pub})$ and $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)}$ for $p = n/4$

(r, m)	(6,13)
n	8,192
k	4,097
$dim(hull(\mathcal{C}_{pub}))$	2,974
$dim(hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(r,m)})$	572

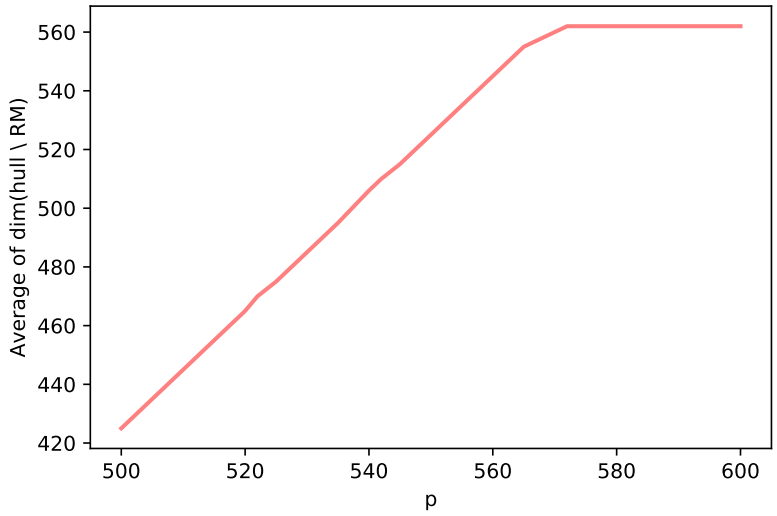

Figure 4.4: The dimension of $hull(\mathcal{C}_{pub}) \setminus \mathrm{RM}_{(6,13)}$ for variation of p.

Table 4.9: Parameter set of *Enhanced pqsigRM*

λ (security)	128
(r, m)	(6,13)
n	8,192
k	4,097
w	1,370
p	$\geq$572
k_{rep}	62
k_{app}	2

2) Complexity of Finding Minimum-Weight Codewords

In *Enhanced pqsigRM*, I consider one more security issue, which is the complexity of finding minimum-weight codewords. Using information set decoding, the probability of successful decoding of the weight-w-error vector is as

$$Prob(Dec) = \frac{\binom{n-k}{w}}{\binom{n}{w}} = \frac{(n-k)(n-k-1)\cdots(n-k-w+1)}{n(n-1)\cdots(n-w+1)} \approx (\frac{n-k}{n})^{w}. \tag{4.2}$$

This probability works the same as finding the minimum-weight codewords problem when syndrome equals 0. Thus, I can get the same equation with (4.2). In other words, the complexity of finding minimum-weight codewords is the inverse of (4.2) substituting w to d_{min} as

$$Complexity = (\frac{n}{n-k})^{d_{min}}. \tag{4.3}$$

I compute this with `Enh-pqsigRM-613` and the result of the complexity is 2^{128}. On the other hand, for the DOOM problem, `Enh-pqsigRM-613` has the work factor of 2^{259}, which is computed from the equation (4.1). That means, considering the complexity of finding minimum-weight codewords, the security level goes down to 128-bit security.

3) Computing the Sizes of the Public Keys, Secret Keys, and Signatures

Regarding the key sizes, first, the public key is a matrix $\boldsymbol{T}$ from a parity check matrix given in the systematic form and it requires $(n-k)k$ bits. The public key size is 2.00 MB for `Enh-pqsigRM-613`. It is 0.5 times the original *Modified pqsigRM* with (6,13)-modified RM codes.

The secret key includes a matrix $\mathbf{Q}$, partial permutations σ_p^1, σ_p^2, $k_{rep} \times 2^r$ repeated replacing codes, $k_{app} \times n$ appending codes, and a $1 \times n$ padding codeword of dual code.

$\mathbf{Q}$ is an $n \times n$ permutation matrix, which can be expressed with just a number. I use nm bits for $\mathbf{Q}$ because I need $\log_2 n$ bits to express a number and the number is from 0 to n. In the same way, σ_p^1 and σ_p^2 need $n(m-2)/2$ bits. The replacing codes, appending codes, and padding codes need $(2^r - 2) \times 2^r$, $k_{app} \times n$, and $1 \times n$ bits, respectively. Thus, the size of the secret key is $3nm/2 + k_{app}n + (2^r - 2)2^r$. It is 22,512 bytes for `Enh-pqsigRM-613`. It is 14,678,016 bytes for the original *Modified pqsigRM* with (6,13)-modified RM codes. Thus, it has the effect of reducing the secret key size to 0.0015 times.

On the other hand, I need $(n+64)$ bits for the signature lengths. n is for the length of e, and 64 is for the size of a 64-bit integer i. The signature length is 1,032 bytes for `Enh-pqsigRM-613`.

4.5 Comparison of *Enhanced pqsigRM* with Other PQC Schemes

4.5.1 Public Key and Signature Size of *Enhanced pqsigRM* Compared with Other PQC Schemes

The public key and signature sizes of *Enhanced pqsigRM*, which satisfies the 128-bit security, are given in Table 4.10. Compared with the NIST PQC finalist schemes [36,37,41], *Enhanced pqsigRM* still has very large public key size and has the smallest signature size except for *Falcon*.

Also, I compare these with the other code-based signature schemes [22, 25, 38] in Table 4.11. *Enhanced pqsigRM* has the smallest signature size among these. Also, it has a smaller public key size than *Wave*. *Durandal* has an extremely small public key, however, its security relies on the security rank metric decoding problem. *Classic McEliece* is a key encapsulation mechanism (KEM), however, I bring this to compare with the proposed scheme. If *Classic McEliece* can be acceptable as a NIST PQC candidate, the proposed scheme is also acceptable regarding the public key size.

Table 4.10: Public key and signature sizes of *Enhanced pqsigRM* compared with the finalist signature schemes of NIST PQC

Security	***Enhanced pqsigRM***		*Crystals-Dilithium*		*Falcon*		*Sphincs+*	
	Public key(MB)	Signature (byte)	Public key(byte)	Signature (byte)	Public key(byte)	Signature (byte)	Public key(byte)	Signature (byte)
128	**2.00**	**1,032**	1,312	2,420	897	666	32	7,856

Table 4.11: Public key and signature sizes of *Enhanced pqsigRM* compared with other code-based signature schemes

Security	***Enhanced pqsigRM***		*Wave*		*Durandal*		*Classic McEliece* (KEM)
	Public key(MB)	Signature (byte)	Public key(MB)	Signature (byte)	Public key(MB)	Signature (byte)	Public key (MB)
128	**2.00**	**1,032**	3.10	1,647	0.015	4,060	0.26

4.5.2 Verification Cycles of *Enhanced pqsigRM* Compared with Other NIST PQC Finalist Schemes.

1) Description of Platform

The following measurements are collected using a desktop computer with an i7-12700 CPU @ 2.10GHz. Turbo Boost is disabled. This machine has 16GB of RAM. Benchmarks have run on one core of the CPU.

2) Number of Cycles

The measurements of CPU cycles compared with the finalists are given in Table 4.12. The data of the finalists are from the submitted documents to NIST and these can be a little bit different because the implementation conditions are different [36, 37, 41]. However, it is almost the same with *Crystals-Dilithium*. Compared with *Crystals-Dilithium*, I have about 0.74 times of average verification cycles for 128 bit-security. It is the second smallest among the NIST PQC finalist algorithms.

Table 4.12: Verification CPU cycles of *Enhanced pqsigRM* compared with the NIST PQC finalists

Security	Verification cycles			
	Enhanced pqsigRM	*Crystals-Dilithium*	*Falcon*	*Sphincs+*
128	**242,901**	327,362	82,340	308,774

The verification CPU cycles of `Enh-pqsigRM-613` are 242,901 (average) and 235,656 (median). The average value is the smallest among other NIST PQC finalists except for *Falcon*.

Furthermore, for key generation, these are 2,034,133,439 (average) and 2,038,358,872 (median). For signing, these are 2,232,288 (average) and 1,366,500 (median).

4.5.3 Memory Usage

`Enh-pqsigRM-613` takes 53,334,140 bytes of memory usage. Also, there is no memory leakage.

Chapter 5

Conclusion

In this dissertation, research works on homomorphic computation in RM codes and improving *Modified pqsigRM* were presented.

In Chapter 3, I suggested a transformation method, called bootstrapping, which facilitates the homomorphic multiplication of the first-order RM codes while preserving the order of the RM codes after the computation (addition and multiplication). For addition, it was evident to match the polynomial addition of messages and vector addition of codewords because RM codes are linear. However, for multiplication, some works were needed. I found a relation between the proposed codeword multiplication $\boldsymbol{c} \odot \boldsymbol{c'}$ and the multiplication of messages. I employed three steps to perform this and called this 'bootstrapping.' In Step 1, I expressed the coefficients of $\boldsymbol{v_i v_j}$ with the components of the codewords $c_0, c_1, \cdots, c_{n-1}, c'_0, c'_1, \cdots$, and c'_{n-1}. In Step 2, I represented the coefficients of x^l with the coefficients of $\boldsymbol{v_i v_j}$. Thus, by merging these two steps, I constructed a relation between the codeword components and coefficients of x^l, which are the components of message polynomial multiplication. I encoded this process by multiplying the generator matrix in Step 3 to obtain the corresponding first-order RM codes' codeword. Thus, this bootstrapping method reset the order of the first-order RM codes and made multiplication possible for infinite times. This research can be a preceding work of constructing a code-based homomorphic encryp-

tion scheme. Furthermore, I proposed some trials of making homomorphic encryption in code-based cryptosystems. I presented a method of homomorphic computation in Reed-Muller codes with errors, a hybrid McEliece cryptosystem using constant weight constant distance codes, and additive homomorphism on the Niederreiter cryptosystem with fixed errors.

In Chapter 4, I improved *Modified pqsigRM* with key size and bit-security. The *Modified pqsigRM* is one of the promising algorithms in code-based PQC digital signature schemes. It was a modified version of *pqsigRM*, which was presented in the first round of the NIST PQC standardization process. It had a big advantage of small signature sizes compared with the other NIST PQC finalist schemes, and this was an important feature because signatures should be sent in every signing process. However, it had the disadvantage of large public key sizes. In this dissertation, I improved the parameters of *Modified pqsigRM* by making them into a systematic form of the public key and resetting the parameters. I called this proposed scheme as *Improved Modified pqsigRM*. In conclusion, I reduced the public key sizes to smaller than half of the original *Modified pqsigRM*. Also, I could improve the exact bit-security, too. However, the public key sizes of *Improved Modified pqsigRM* were still large compared with the other digital signature finalist schemes of the NIST PQC standardization process. Thus, I could reduce these to 0.20, 0.40, and 0.23 times of public key sizes compared to the *Modified pqsigRM* parameters for 80, 128, and 256 security levels, respectively. Also, I obtained a larger exact bit-security for these parameters than *Modified pqsigRM*. Compared with NIST PQC finalist algorithms, public key sizes were still large, but signature sizes were the smallest among all. Moreover, I calculated the verification cycles compared with the NIST PQC finalist algorithms. For 128 bits of classical security, the signature size of the proposed signature scheme was 520 bytes, which corresponds to 0.21 times that of *Crystals-Dilithium*, and it was the second smallest among the NIST PQC finalist algorithms. The number of average verification cycles was 172,669, which corresponds to 0.53 times that of *Crystals-Dilithium*, and it was

the second smallest among the NIST PQC finalist algorithms.

Furthermore, I proposed an enhanced version of these, called *Enhanced pqsigRM*, considering the attacks using the information set decoding and finding the minimum-weight codewords. Thus, I enhanced the security issues of the previous work, but the parameters got worse. I maintained the systematic public key as in *Improved Modified pqsigRM*. Also, I minimized the secret key size by taking the essential parts. Moreover, I simplified the hashing process by using the hash function just once instead of twice. Compared with the original *Modified pqsigRM*, the public key size was reduced to 2.0 MB, which is 0.5 times the previous one, and the secret key size was reduced to 22,512 bytes, which is 0.0015 times the previous one.

Note that *Enhanced pqsigRM* had an advantage of a short signature size. It is a very important feature in digital signature schemes because the signature should be sent for every signing process, while the public key is made once. Also, note that it had the advantage of fast verification cycles. It is also a significant issue because on the aspect of a consumer, verifying the signature fast is important. NIST also mentioned that these two features will be considered in the additional digital signature scheme submission. The signature size was 1,032 bytes, which corresponds to 0.42 times that of *Crystals-Dilithium*, and it was the second smallest among the NIST PQC finalist algorithms. The number of average verification cycles was 242,901, which corresponds to about 0.74 times that of *Crystals-Dilithium*, and it was the second smallest among the NIST PQC finalist algorithms. Moreover, *Enhanced pqsigRM* was based on the traditional security problem called syndrome decoding problem, which has not been broken for more than 40 years. Also, this could be easier to make hardware implementations because it was based on binary operations, while the lattice-based cryptosystems used modulo operations. At last, this can be a good PQC digital signature alternate scheme to diversify the categories.

For future works, I can consider the homomorphic computation of higher-order RM codes. Moreover, I can implement the computation in the presence of noise,

which can be useful in homomorphic cryptography. In the long run, this work can be a preceding work for constructing a homomorphic encryption scheme with code-based cryptosystems. To make this possible, schemes, which can decode errors after homomorphic operations are needed. Also, the key sizes should be reduced to use the homomorphic scheme feasibly.

Despite enhancing the parameters of *Modified pqsigRM*, *Enhanced pqsigRM* still had a large public key size compared with the other digital signature finalist schemes of the NIST PQC standardization process. Thus, diverse research works are needed to improve *Enhanced pqsigRM* more.

Bibliography

[1] S. Lin and D. Costello, *Error Control Coding*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2001, pp. 105–118.

[2] F. J. Macwilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, vol. 16, New York, NY, USA: Elsevier/North-Holland Inc., 1977, pp. 370–479.

[3] R.J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *DSN Progress Report*, pp. 114–116, Jan. 1978.

[4] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.

[5] R. Meissen, "A mathematical approach to fully homomorphic encryption," Ph.D. dissertation, Worcester Polytechnic Institute, 2012.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Sec.* New York, NY, USA: Springer, 2017, pp. 409–437.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "TFHE: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[8] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," Int. Assoc. Cryptol. Res., Tech. Rep. 2018/1043, 2018. [Online]. Available: https://eprint.iacr.org/2018/1043.

[9] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," *Cryptographers' Track at the RSA Conference*, Springer, Cham, Feb. 2020, pp. 364–390.

[10] J.-W. Lee, Y.-S. Kim, and J.-S. No, "Analysis of modified Shell sort for fully homomorphic encryption," *Cryptology ePrint Archive*, 2019/1497.

[11] A. R. Calderbank, E. M. Rains, P. M. Shor, and N. J. A. Sloane, "Quantum error correction via codes over GF(4)," *IEEE Trans. Inf. Theory*, vol. 44, no. 4, pp. 1369–1387, Jul. 1998.

[12] D. MacKay, G. Mitchison, and P. McFadden, "Sparse-graph codes for quantum error correction," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2315–2330, Oct. 2004.

[13] C. Y. Lai, Y. C. Zheng, and T. A. Brun, "Fault-tolerant preparation of stabilizer states for quantum Calderbank-Shor-Steane codes by classical error-correcting codes," *Phys. Rev. Lett.*, 95, 032339, 2017.

[14] F. Lacerda, J. Renes, and R. Renner, "Classical leakage resilience from fault-tolerant quantum computation," *Journal of Cryptology*, vol. 32, no. 4, pp. 1071–1094, 2019.

[15] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[16] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2418–2422.

[17] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, Oct. 2019.

[18] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," *arXiv preprint arXiv:1612.03301*, 2016.

[19] E. Abbe, A. Shpilka, and A. Wigderson, "Reed-Muller codes for random erasures and errors," *IEEE Trans. Inf. Theory*, vol. 61, no. 10, pp. 5229–5252, Oct. 2015.

[20] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[21] N. T. Courtois, M. Finiasz, and N. Sendrier, "How to achieve a McEliece based digital signature scheme," in *Proc. Asiacrypt, Gold Coast*, Australia, Dec. 2001, pp. 157–174.

[22] T. Debris-Alazard, N. Sendrier, and J.-P. Tillich, "Wave: A new family of trapdoor one-way preimage sampleable functions based on codes," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur*, Kobe, Japan: Springer, 2019, pp. 21–51.

[23] W. Lee, Y. S. Kim, Y. W. Lee, and J. S. No, "Post quantum signature scheme based on modified Reed–Muller code pqsigRM," in *First Round Submission to the NIST Postquantum Cryptography Call*, Nov. 2017. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Round-1-Submissions.

[24] Y. Lee, W. Lee, Y. S. Kim and J. -S. No, “Modified pqsigRM: RM Code-Based Signature Scheme,” *IEEE Access*, vol. 8, pp. 177506–177518, 2020.

[25] N. Aragon, O. Blazy, P. Gaborit, A. Hauteville, and G. Zémor, “Durandal: A rank metric based signature scheme,” in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Darmstadt, Germany: Springer, pp. 728–758, 2019.

[26] I. Reed, “A class of multiple-error-correcting codes and the decoding scheme,” *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, Sep. 1954.

[27] D. E. Muller, “Application of Boolean algebra to switching circuit design and to error detection,” *Transactions of the I.R.E. Professional Group on Electronic Computers*, vol. EC-3, no. 3, pp. 6–12, Sep. 1954.

[28] I. Dumer, “Recursive decoding and its performance for low-rate Reed-Müller codes,” *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 811–823, May 2004.

[29] W. Lee, J.-S. No, and Y.-S. Kim, “Punctured Reed–Muller code-based McEliece cryptosystems,” *IET Commun.*, vol. 11, no. 10, pp. 1543–1548, Jul. 2017.

[30] J. Cho, Y. Kim, and J. No, “Homomorphic computation in Reed–Muller codes,” *IEEE Access*, vol. 8, pp. 108 622–108 628, 2020.

[31] I. Duursma and J. Shen, “Multiplicative secret sharing schemes from Reed–Muller type codes,” in *2012 IEEE International Symposium on Information Theory Proceedings IEEE*, 2012, pp. 264–268.

[32] M. Vajha, V. Ramkumar, and P. Vijay Kumar, “Binary, shortened projective Reed–Muller codes for coded private information retrieval,” in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2648–2652.

[33] J. Preskill, “Quantum computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, 2018

[34] H. S. Zhong, H. Wang, Y. H. Deng, M. C. Chen, L. C. Peng, Y. H. Luo, J. Qin, D. Wu, X. Ding, Y, Hu, P. Hu, X. Y. Yang, W. J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N. L. Liu, C. Y. Lu, and J.W. Pan, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020

[35] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-Kyber," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[36] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehle, and S. Bai, "Crystals-Dilithium," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[37] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[38] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. Maurich, R. Misoczki, R. Niederhagen, K. G. Patterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang, "Classic McEliece," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[39] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor, "BIKE," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[40] C. A. Melchor, J.-C. Deneuville, N. Aragon, P. Gaborit, S. Bettaieb, E. Persichetti, L. Bidoux, J.-M. Robert, O. Blazy, P. Veron, J. Bos, and G. Zemor, "HQC," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[41] A. Hulsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, J.-P. Aumasson, B. Westerbaan, and W. Beullens, "Sphincs+," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[42] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, D. Urbanik, G. Pereira, K. Karabina, and A. Hutchinson, "SIKE," *NIST PQC Standardization Process Round 3 Submission*, 2021.

[43] C. Peikert, "Lattice cryptography for the internet," *International Workshop on Post-Quantum Cryptography*, Springer, Cham, 2014.

[44] O. Goldreich, S. Goldwasser, and S. Halevi, "Public-key cryptosystems from lattice reduction problems," *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, Springer, London, pp. 112–131, 1997.

[45] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Prob. Contr. Inf. Theory*, vol.15, pp. 159–166, 1986.

[46] V. M. Sidelnikov and S. O. Shestakov, "On insecurity of cryptosystems based on generalized Reed-Solomon codes," *Discrete Mathematics and Applications*, vol.1, no. 4, pp. 439–444, 1992.

[47] V. M. Sidelnikov, "A public-key cryptosystem based on binary Reed-Muller codes," *Discrete Mathematics and Applications*, vol.4, no. 3, pp. 191–208, 1994.

[48] M. Esmaeili, M. Dakhilalian, and T. A. Gulliver, "New secure channel coding scheme based on randomly punctured quasi-cyclic-low density parity check codes," *IET Commun.*, vol.4, no. 3, pp. vol. 8 no. 14, 2556–2562, Sep. 2014.

[49] S. R. Shrestha and Y. -S. Kim, "New McEliece cryptosystem based on polar codes as a candidate for post-quantum cryptography," *Proc. ISCIT 2014 Incheon, Korea*, Sep. 2014, pp. 368–372.

[50] J. -C. Faugere, V. Gauthier-Umana, A. Otmani, L. Perret, and J. -P. Tillich, "A distinguisher for high-rate McEliece cryptosystems," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6830–6844, Oct. 2013.

[51] L. Minder and A. Shokrollahi, "Cryptanalysis of the Sidelnikov cryptosystem," in *Proc. Eurocrypt, in Lecture Notes in Computer Science*, vol. 4515. Barcelona, Spain: Springer, 2007, pp. 347–360.

[52] I. V. Chizhov and M. A. Borodin, "The failure of McEliece PKC based on Reed-Müller codes," in *IACR Cryptol. ePrint Arch., Tech. Rep. 2013/287*, 2013.

[53] A. Otmani and H. T. Kalachi, "Square code attack on a modified Sidelnikov cryptosystem," in *Proc. C2SI*, 2015, pp. 173–183.

[54] B. Biswas and N. Sendrier, "McEliece cryptosystem implementation: Theory and practice," *PQCrypto 2008*, 2008 Proceedings 2, pp. 47–62, Oct. 2008.

[55] R. C. Bose, "On the construction of balanced incomplete block designs," *Annals of Eugenics*, vol. 9, no. 4, pp. 353–399, 1939.

[56] N. Sendrier, "Decoding one out of many," *International Workshop on Post-Quantum Cryptography*, Springer, Berlin, Heidelberg, 2011.

초 록

이 학위 논문에서는 다음 두 가지의 연구가 이루어졌다: i) Reed-Muller(RM) 부호의 동형 계산 및 ii) *Modified pqsigRM*의 키 크기 및 비트 보안성 개선.

먼저, RM 부호의 동형 연산 방법을 제안한다. 인공 지능(AI), 빅 데이터 및 클라우드 서비스의 지속적인 개발과 함께 완전 동형 암호(FHE)는 기계 학습 시스템에서 개인 정보 보호 및 보안을 유지하기 위한 해결책으로 고려되고 있다. 현재 대부분의 기존 FHE 체계는 격자 기반 암호화를 사용한다. 최신 알고리즘에서는 동형 곱셈과 더 많은 연산을 위해 암호문을 새로 고치는 데 필요한 해당 부트스트래핑에 엄청난 양의 계산 리소스가 필요하다. 따라서 실제 적용을 위해 계산 복잡도를 줄일 수 있는 FHE에 대한 새로운 혁신적인 접근 방식을 찾는 것이 필요하다. 격자 기반 암호에 국한되지 않는 다양한 연구 또한 필요하다. 부호 기반 암호는 이에 대한 새로운 해결책이 될 수 있다.

본 논문에서는 RM 부호를 통해 부호 기반 동형 연산 기법을 제안한다. 선형 부호는 덧셈에 대해 닫힌 것으로 알려져 있지만 선형 부호에서 곱셈 동형 연산을 달성하는 것은 지금까지 불가능했다. 나는 RM 부호를 사용하여 덧셈과 곱셈을 동시에 지원할 수 있는 완전 동형 부호 체계를 제안하여 이 문제를 해결하려고 노력한다. 이는 부호 기반 FHE 방식을 구축하기 위한 선행 단계라고 할 수 있다. 나는 이것을 1차 RM 부호의 연산으로 제한한다. 곱셈 후 RM 부호의 차수가 증가하게 되는데 이 과정의 RM 부호의 차수를 줄여 많은 수의 연산을 수행하는 부트스트래핑 기법이 필요하다. 나는 메시지에 대한 연산과 RM 부호의 해당 부호어에 대한 연산 사이에 일대일 관계를 생성하는 3개의 연속적인 선형 변환을 제안하여 덧셈 또는 곱셈 후 RM 부호의 차수를 보존하는 부트스트래핑 기술을 제안한다. 추가적으로, 부호 기반

암호에서의 동형 암호를 위한 몇 가지 시도들도 제안한다.

두 번째로는, *Modified pqsigRM* 방식의 키 크기 및 비트 보안 개선 방안을 제안한다. 양자 알고리즘에도 안전한 포스트 양자 암호 (PQC)의 중요성이 점점 커지고 있다. *pqsigRM*은 국립 표준 기술 연구소 (NIST)의 PQC 표준화 과정의 1라운드에서 제시된 부호 기반 PQC 디지털 서명 체계이다. NIST는 미국의 표준화 기구이다. 이 체계는 NIST PQC 표준화 프로세스에서 논의를 거쳐 알려진 모든 취약점을 제거하여 *Modified pqsigRM*으로 개정되었다. 그 장점으로는 효율적인 복호화 과정과 작은 서명 크기가 있다. 모든 서명 과정에서 서명을 보내야 하기 때문에 작은 서명 크기는 전자 서명 체계에서 매우 유용하다. 그러나 공개 키 크기가 크다는 문제가 있다.

본 논문에서는 *Modified pqsigRM*의 공개 키 크기를 줄이고 구체적인 비트 보안을 향상시키는 방법을 제안한다. 공개 키를 체계적인 형태로 변경하고 매개변수를 개선하며 각 매개변수에 대한 비트 보안을 미세 조정한다. 따라서 80, 128 및 256 비트의 보안 수준에 대해서 *Modified pqsigRM* 매개변수에 비해 각각 0.20, 0.40 및 0.23배 작은 공개 키 크기로 줄일 수 있다. 또한 이러한 매개변수에 대해 *Modified pqsigRM*보다 더 큰 값의 구체적인 비트 보안을 얻는다. NIST PQC 최종 알고리즘들 *Crystals-Dilithium, Falcon,* 그리고 *Sphincs+*와 비교할 때 공개 키 크기는 여전히 크지만 서명 크기는 모든 보안 레벨에 대해 가장 작다. 128 비트의 기존 보안에 대해 제안하는 서명 방식의 서명 크기는 520 바이트로 *Crystals-Dilithium*의 0.21 배에 해당한다. 또한 NIST PQC 최종 후보 알고리즘과 비교하여 검증 싸이클 수를 계산한다. 검증 싸이클 수의 평균 값은 172,669로 *Crystals-Dilithium*의 그것의 0.53 배에 해당하며 이것은 네 개의 NIST PQC 최종 알고리즘들 중에 두 번째로 작다.

추가적으로, 정보 집합 복호화 및 부호어의 최소 무게를 계산하는 공격을 고려하여 *Enhanced pqsigRM*이라고 하는 향상된 버전을 제안한다. *Modified pqsigRM*과의 차이점은 체계적인 공개 키를 사용하고 비밀 키를 최소화하며 해시 기능을 단순화하고 보안 문제를 개선하는 것이다. 매개 변수 값이 안 좋아진다는 트레이드 오프가 있다. 하지만, *Modified pqsigRM*에 비해 공개키 크기는 0.5배인 2.0 MB로 줄어들고, 비밀키 크기는 0.0015배인 22,512 바이트로 줄어든다. 또한, 서명 크기가 작고 검증 주기가 빠르다는 장점이 여전히 존재한다. 이들은 디지털 서명 체계에서 매우 중요

한 기능이다. 서명 크기는 1,032 바이트로 *Crystals-Dilithium*의 0.42 배에 해당하며 이것은 네 개의 NIST PQC 최종 알고리즘들 중에 두 번째로 작다. 평균 검증 싸이클 수는 242,901로 *Crystals-Dilithium*의 약 0.74배에 해당하며 이것은 네 개의 NIST PQC 최종 알고리즘들 중에 두 번째로 작다.

주요어: 오류 정정 부호(ECCs), 완전 동형 암호(FHE), 동형 연산, 포스트 양자 암호(PQC), Reed-Muller(RM) 부호, 디지털 서명 시스템, 부호 기반 암호 시스템.
학번: 2017-26340