博士論文

Security Evaluation of Public-Key Cryptography against Side-Channel Attacks and Quantum Attacks

（公開鍵暗号に対するサイドチャネル攻撃および量子攻撃の安全性評価）

大西　健斗

Copyright © 2020, Kento Oonishi.

Abstract

Public-key cryptography is an indispensable technology for the information society, and the security of public-key cryptosystems must be guaranteed to use them. The standard definitions of security are based on computational problems, which take a long time to solve. However, both side-channel and quantum attacks threaten security. Side-channel attacks may recover sensitive information using data leaked physically, depending on an actual implementation. The standard definitions of security do not consider the actual implementations, and cryptosystems will be insecure even if the security is guaranteed. Thus, security against side-channel attacks must be considered based on the actual implementations. In addition to side-channel attacks, threats of quantum computers also exist. Quantum computers have more computational power than the currently used computers. Ideal quantum computers break the currently used public-key cryptosystems such as the RSA scheme. The National Institute of Standards and Technology (NIST) promotes the standardization of post-quantum cryptography against quantum attacks. We must also consider the threats of quantum attacks to the currently used public-key cryptosystems because it takes a long time to migrate to new cryptosystems.

In this thesis, we evaluate the security of public-key cryptosystems against side-channel and quantum attacks. First, we evaluate the security of the CRT-RSA scheme against side-channel attacks. The CRT-RSA scheme is a currently used public-key cryptosystem. We focus on the CRT-RSA scheme using the sliding window method. We evaluate security against sliding window leakage. Next, we evaluate the security of the Ring-LWE and Module-LWE based schemes, which are next-generation public-key cryptosystems, against side-channel attacks. We focus on Lyubashevsky et al.'s cryptosystem, the origin of these schemes among NIST candidates. We evaluate the security of Lyubashevsky et al.'s cryptosystem against leakage from a number theoretic transform. Finally, we evaluate the security of the RSA scheme against quantum attacks. We reduce the cost of a controlled modular adder, which is the core arithmetic of Shor's algorithm. Based on our construction of a controlled modular adder, we evaluate the security of the RSA scheme.

Contents

Chapter 1

Introduction

1.1 Background

This section introduces the field of public-key cryptography [24]. First, Section 1.1.1 introduces public-key cryptography itself. Then, Section 1.1.2 and 1.1.3 describe threats of side-channel attacks [56] and quantum attacks [103] on public-key cryptosystems, respectively.

1.1.1 Public-Key Cryptography

Information communication technology is indispensable for the information society. Many services are developed based on information communication technology. For example, we communicate through e-mail in everyday life. We often buy goods and use banking services on the Internet. Moreover, cloud services are widely used to store and manage a large amount of data. We must provide a significant amount of information to use such services, including personal information and transactions conveniently. Sensitive information must be protected from attackers to use these services securely.

However, the risk of exposure to attackers is incurred when we send information using a communication channel. If attackers wiretap this communication channel, they can extract the content of the communication. Thus, if we send sensitive information through a communication channel, it will be leaked to attackers. Moreover, risks are not limited to information leakage. Attackers can also replace information on a communication channel. Such risks prevent secure communication, and we must therefore consider countermeasures for overcoming them.

Cryptography is a technology for dealing with sensitive information in a secure manner. Sensitive information is protected from attackers owing to the use of cryptography. We now review public-key encryption and digital signature schemes, which are fundamental schemes applied in public-key cryptography.

Public-Key Encryption Scheme

The encryption schemes protect sensitive information and are composed of three types of algorithms:

- Key generation
- Encryption
- Decryption

These three types of algorithms can be described as follows:

- **Key Generation.** Encryption keys and decryption keys are generated. A sender has the encryption keys, and the receiver has the decryption keys.

- **Encryption.** The sender scrambles a message using the encryption keys and generates the ciphertext.
- **Decryption.** The receiver recovers the message in the ciphertext using decryption keys.

We now describe the symmetric-key and the public-key encryption schemes. These schemes differ in the way the keys are distributed. Although this thesis focuses on public-key encryption schemes, we also review the use of the symmetric-key encryption schemes for comparison.

In the symmetric-key encryption schemes, the encryption keys and the decryption keys are the same. These keys are called the secret keys. Before applying a symmetric-key encryption scheme, the sender and the receiver must share the secret keys. This key-exchange step is the major problem in this type of scheme. If we communicate with many people, different keys must be generated and exchanged with each individual. The AES [78] is a currently used symmetric-key encryption scheme.

The public-key encryption schemes employ different keys for the encryption and decryption keys [24], which is different from the symmetric-key encryption schemes. Although only an authorized receiver has the decryption keys, everyone can obtain the encryption keys. The encryption keys are called the public keys, and the decryption keys are called the secret keys.

In the public-key encryption schemes, anyone can generate a ciphertext, and the authorized receiver only decrypts this ciphertext. The key-exchange used in the public-key encryption schemes is more useful than that used in the symmetric-key encryption schemes. However, the public-key encryption schemes require a longer time for encryption and decryption than the symmetric-key encryption schemes. Thus, a public-key encryption scheme is used for the key-exchange scheme of a symmetric-key encryption scheme [24]. RSA [76, 99] and elliptic curve [55, 74] schemes are currently used public-key encryption schemes.

Digital Signature Scheme

The digital signature schemes verify whether a sender indeed sent the message and prevent the forgery of messages. The digital signature schemes are composed of three algorithms:

- Key generation
- Signing
- Verification

These three algorithms are described as follows:

- **Key Generation.** Signing keys and verification keys are generated. The sender has the signing keys, and a receiver has the verification keys.
- **Signing.** The sender generates a signature on the message by using the signing keys.
- **Verification.** The receiver verifies the signature on the message by using the verification keys.

The digital signature schemes verify the integrity of a message and its sender. Thus, the verification keys are public to everyone, and only an authorized sender has the signing keys. The signing keys are called the secret keys, and the verification keys are called the public keys. RSA [76, 99], DSA [79], and elliptic curve [55, 74] schemes are widely used for digital signature schemes.

Standard Definitions of Security of Public-Key Cryptosystem

We now focus on the standard definitions of security of public-key cryptosystems. There are many definitions of security of the public-key encryption schemes and the digital signature schemes. In both schemes, security is defined based on the following:

- The security level of the public-key cryptosystem
- The model of attackers

Security is guaranteed by proving that the public-key cryptosystem achieves the security level under the attackers.

A computational problem requiring a long time to solve, e.g., a super-polynomial time, is used for guaranteeing security. For example, the security of the RSA scheme [99] is based on the integer factoring problem. For security used in public-key cryptosystems, the following are performed:

- Guarantee the hardness of breaking the cryptosystem based on the computationally difficult problem
- Evaluate time complexity for solving the computational problem

We will now describe these in more detail.

First, we focus on guaranteeing the hardness of breaking the cryptosystem. The security proof tries to solve the computational problem by assuming that the attackers break the cryptosystem. Then, breaking the cryptosystem is harder than solving the computational problem. Thus, the security of the cryptosystem is guaranteed.

Second, we focus on evaluating time complexity in solving the computational problem. As noted above, security is based on a computational problem that is believed to require a long time to solve. The computational problem must not be solved in a polynomial time to guarantee the security of a cryptosystem. We must clarify the hardness of the computational problem. We provide an efficient algorithm for solving the computational problem to show the hardness. Then, the cryptosystem security is evaluated by the time complexity for solving the corresponding computational problem. The security is given as λ-bit security when the corresponding computational problem is solved with 2^{λ}-time. Development of computational power updates the appropriate value of λ. Then, the appropriate parameters must be determined both theoretically and experimentally.

Standardization Project of Next-Generation Public-Key Cryptosystem

As noted above, RSA [99] and elliptic curve [55, 74] schemes are widely used. The RSA and elliptic curve schemes are based on the integer factoring problem and the elliptic curve discrete logarithm problem. However, quantum computers can solve the integer factorization problem and the elliptic curve discrete logarithm problem in polynomial time [103]. The secret keys are then calculated on the currently used public-key cryptosystem, such as the RSA scheme. Thus, currently used cryptosystems are vulnerable to quantum computers.

Based on this fact, the National Institute of Standards and Technology (NIST) promotes the standardization of next-generation public-key cryptosystems, which provides resistance for quantum computers [80]. This standardization is in the third round, and seven candidates survive for the next-generation public-key cryptosystems. These cryptosystems are based on computational problems that are believed to be resistant to quantum computers. Moreover, studies on new security models against quantum computers are proceeding [44].

1.1.2 Threat of Side-Channel Attacks on Public-Key Cryptosystems

We now introduce more practical security. We explain side-channel attacks [56] on a public-key cryptosystem. Side-channel attacks recover the secret keys from physically leaked data and break the cryptosystem. Physically leaked data is extracted from various ways. For example, data is leaked as the decryption time [56], power consumption [47, 57], sound [32], and cache access [10, 50, 89, 119, 120].

It is well known that side-channel attacks are significant threats to cryptosystems used on computers, smartcards, and other devices. However, threats of side-channel attacks are not considered in the standard definition of security discussed above because physically leaked data depends on how the cryptosystems are implemented. Thus, security is not guaranteed when a cryptosystem is exposed to threats of side-channel attacks. This thesis focuses on a method for evaluating the security of a public-key cryptosystem against side-channel attacks.

We now explain the general strategy of previous side-channel attacks on a public-key cryptosystem. Previous studies on side-channel attacks have mainly focused on extracting the secret keys of the public-key cryptosystem. These studies have considered methods for extracting physically leaked data related to the secret keys, and methods for recovering the secret keys from the extracted data have been proposed.

However, physically leaked data includes various noises. These errors occur because of measurement errors, an environment of measurement, and other reasons. Most studies have tried to eliminate such noise by measuring data many times and averaging the measurement results. Moreover, most studies have researched methods for measuring and processing data to obtain physically leaked data as accurately as possible. Then, the security of the public-key cryptosystem against side-channel attacks is evaluated.

However, physically leaked data may not be observed many times because unauthorized people access a device. It is difficult to eliminate noise in physically leaked data from a few observations. When attackers treat the data corresponding to each secret key separately, the entire secret keys will not be recovered because of noise.

Mathematical structure in public-key cryptosystems causes the recovery of the entire secret keys from noisy data [42, 43, 95]. These methods allow attackers to recover the secret keys from a few observations. These attacks are serious, and we must evaluate the threats of these attacks. Previous studies provided a mathematical model on the extracted data and evaluated security on this model. These studies clarify how much accuracy of extracted data is required to recover the secret keys.

An efficient implementation may be one of the reasons for attacks using mathematical structure. Many cryptosystems are attacked based on their efficient implementation. For example, the following public-key cryptosystems are attacked:

- The RSA scheme [99] using Chinese Remainder Theorem (CRT-RSA) [76]
- Ring-LWE and Module-LWE based schemes

These schemes adopt efficient calculation methods by preparing additional secret keys and transforming the secret keys, respectively. The same secret keys are inputted multiple times in these schemes. Thus, the extracted data has redundancy, and the secret keys are recovered even if there is noise in the extracted data. This thesis focuses on security, even when noise is present in the extracted data. We now describe previous side-channel attacks on these schemes.

Previous Side-Channel Attacks on CRT-RSA Scheme

The CRT-RSA scheme is widely used. In the CRT-RSA scheme, following side-channel attacks are proposed:

- Attacks on modular exponentiations [56]
- Bit extraction through cold-boot attacks [40]
- Attacks on key generation [15]

This thesis focuses on the first type of attacks. Before describing these attacks, we briefly explain the second and third attacks for comparison. The second attacks observe the DRAM data remanence and read the CRT-RSA secret keys. Many studies have developed methods for the recovery of the CRT-RSA secret keys based on these attacks. Heninger and Shacham [43] proposed a method for recovering the CRT-RSA secret keys from noisy CRT-RSA secret keys with random erasure bits. Their method uses the redundancy of the CRT-RSA secret keys to decrease the number of candidates. Heninger and Shacham also proposed a recovery method for the CRT-RSA secret key using a binary tree. Based on their method, many studies have been conducted under various noise conditions. For example, Henecka et al. [42] considered error bits, and Kunihiro et al. [59] considered both erasure and error bits. Similarly, recovery methods for the CRT-RSA secret keys have been extended to more general cases [58, 60, 87]. The third type of attacks [15] focus on the key-generation step and extract operations during the calculation of the greatest common divisor using the secret keys.

We now review the first type, i.e., attacks on modular exponentiations. The secret keys are used as the exponents in modular exponentiations. The modular exponentiations are implemented using the binary method, the fixed window method, or the sliding window method. These methods repeat the squaring and multiplication to calculate the modular exponentiation. With the binary method, only one multiplier is used for each multiplication. The parameter called window size w is adopted in the other methods. These methods are composed of precomputing 2^{w-1} multipliers and calculating the exponentiation. One multiplier is chosen for each multiplication. When a larger w is used, the exponentiation is calculated more quickly by reducing multiplications. However, calculating and storing more precomputed multipliers is required. Thus, an appropriate w is chosen from the number of bits of an exponent and memory capacity. For example, the currently used 2048-bit CRT-RSA scheme is calculated fastest when $w = 6$. A smaller w must be used in devices with a small memory capacity.

We now describe side-channel attacks on modular exponentiations. These side-channel attacks extract the secret exponent. The first side-channel attack [56] is applied to the binary method and extracts the implementation time. Messerges et al. [73] then found that squaring and multiplication are distinguishable. The modular exponentiation is monitored during the decryption, and the RSA secret keys are extracted as the square-and-multiply sequence. Based on this fact, many side-channel attacks on the standard RSA and CRT-RSA schemes have been proposed using extracted square-and-multiply sequences. We now review side-channel attacks on each modular exponentiation.

In the binary method, the secret exponent is in one-to-one correspondence with the square-and-multiply sequence. The binary method is therefore vulnerable to side-channel attacks when the correct square-and-multiply sequences are extracted. The secret exponent cannot be determined immediately only from the square-and-multiply sequence in fixed or sliding window methods. There are many candidates for the multiplier in each multiplication. Walter [117] addressed this problem by proposing the Big Mac attack. This attack obtains the multiplier in each multiplication, and the secret exponent is immediately recovered. Many studies have followed this strategy [31, 47, 50, 89, 120], and the

authors have discussed how to obtain multipliers as efficiently and correctly as possible.

However, if extracting multiplier is failed, the CRT-RSA secret keys are not immediately recovered because of 2^{w-1} candidates in each multiplication. To address this issue, a method for recovering the CRT-RSA secret keys without knowing the multiplier is proposed [10]. Their method recovers the CRT-RSA secret keys from the correct square-and-multiply sequences. This method uses the redundancy of the CRT-RSA secret keys to decrease the number of candidates of the multiplier by applying Heninger-Shacham's method [43]. Their result shows that the CRT-RSA secret keys are recovered in polynomial time when $w \leq 4$. Moreover, their method recovers 13% of the CRT-RSA secret keys when $w = 5$ for the currently used 2048-bit CRT-RSA scheme. It should be noted that the CRT-RSA secret keys are not recovered in polynomial time when $w = 5$, but threats of side-channel attacks should be discussed.

Unfortunately, previous work [10] has failed to address noisy square-and-multiply sequences. The previous method fails to recover the CRT-RSA secret keys because there are errors in the extracted square-and-multiply sequences. Experiments showed that errors occur at an average rate of 0.011 in square-and-multiply sequences, which corresponds to 22 errors [10]. They claimed that these errors could be corrected through majority voting on 20 square-and-multiply sequences. However, there is no guarantee that obtaining 20 square-and-multiply sequences is possible because the decryption or signing process depends on the authorized user. Therefore, we should evaluate the security of the CRT-RSA scheme based on a single square-and-multiply sequence with errors.

To address a security analysis based on noisy square-and-multiply sequences, Oonishi and Kunihiro [84] proposed a method for recovering the CRT-RSA secret keys from noisy square-and-multiply sequences. They proposed a recovering method for $w = 1$. The CRT-RSA secret keys are recovered in polynomial time when the error rate is less than 0.058 on square-and-multiply sequences by using their method. However, their method cannot be applied to $w > 1$. A recovery method is proposed for a general w [83], although the analysis is given only for $w = 2$. Thus, we should evaluate security on a more extensive range of w.

Previous Side-Channel Attacks on Ring-LWE and Module-LWE Based Scheme

Ring-LWE and Module-LWE based schemes are candidates for lattice-based cryptosystems in the NIST standardization project and are based on the Ring-LWE [66] and the Module-LWE [12] problems, respectively. These cryptosystems are defined on a polynomial ring. They are efficiently implemented using a number theoretic transform (NTT) [38, 63]. The NTTs reduce the computational cost of the multiplication of two polynomials and realize a more efficient scheme. Two Module-LWE based cryptosystems survive the third round of the NIST standardization project, namely, CRYSTALS-KYBER [102] and CRYSTALS-DILITHIUM [65].

Primas et al. [95] proposed a method for attacking the NTT on the Lyubashevsky et al.'s cryptosystem (LPR cryptosystem) [67]. The LPR cryptosystem is the basis of the NIST candidates, and their attack is proposed before the beginning of the NIST selection. Until now, many side-channel attacks have been proposed for Ring-LWE and Module-LWE based cryptosystems in the NIST standardization project [3, 5, 7, 90, 96, 97, 118]. Albrecht et al. [3] proposed cold-boot attacks [40] at the secret keys stored using an NTT on CRYSTALS-KYBER and NewHope [92]. Amiet et al. [5] found a vulnerability in the message encoding of NewHope. Aysu et al. [7] proposed a method for attacking the standard polynomial multiplication. Primas and Pessl [90] proposed the attack on CRYSTALS-KYBER by extending Primas et al.'s attack [95]. Ravi et al. [96] found a vulnerability in the message decoding of CRYSTALS-KYBER, LAC [64], and NewHope. Such vulnerability of a message decoding can be combined with leakage at the

output of the inverse NTT on CRYSTALS-KYBER [118]. Ravi et al. [97] also found a vulnerability in the error-correction of LAC. This research also found a vulnerability in the implementation of a Fujisaki-Okamoto transformation [30].

This thesis focuses on Primas et al.'s attack [95] because it causes a severe threat on the NTTs, which are the core arithmetic for realizing efficient Ring-LWE and Module-LWE based schemes. The NTTs use a similar structure as a Fast Fourier Transformation. The Cooley-Tukey NTT and Gentleman-Sande inverse NTT (INTT) [63] are used as NIST candidates. The NTTs convert a polynomial into another polynomial using butterfly operations composed of addition, subtraction, and multiplication. The NTTs are composed of a butterfly's layer. A polynomial is converted into a different polynomial in each layer.

Primas et al.'s attack is a serious threat for Ring-LWE and Module-LWE based schemes using an NTT because it extracts the same secret with high redundancy. Primas et al. applied template attacks [16] on all butterfly operations and extracted the information of addition, subtraction, and multiplication as probability distributions. High redundancy occurs because probability distributions related to the same polynomial are extracted from all layers of the NTTs. The secret keys are recovered from these probability distributions by using this redundancy. The authors then evaluated the security of the LPR cryptosystem.

However, their approach just heuristically provides an evaluation of the security. Their method adopts an iterative method to recover the secret keys. This iterative method is not assured to converge to the accurate values, and the property of this iterative method has not been studied well. Primas et al.'s analysis should be performed by an exact method to clarify threats on the NTTs.

1.1.3 Threat of Quantum Computers on Public-Key Cryptosystems

In addition to side-channel attacks, threats of quantum computers on public-key cryptosystems must be considered. Quantum computers are new types of computers, although perfect versions have not been realized. Shor [103] proposed a polynomial-time quantum algorithm for the integer factoring and elliptic curve discrete logarithm problems. By solving these problems, the secret keys of the RSA and elliptic curve schemes are calculated. Thus, the currently used public-key cryptosystems break down when quantum computers are realized. However, it will take a long time to migrate to next-generation cryptosystems. Thus, the threats of quantum computers must be evaluated on the currently used cryptosystems.

We now review quantum computers. Quantum computers have more computational power than currently used computers, which are called classical computers herein. In classical computers, only **0** and **1** bits are used. In quantum computers, quantum bits (qubits) are used, which can take not only values of **0** and **1** but also a superposition of **0** and **1**. Specifically, a qubit is represented as

$$a\,|0\rangle + b\,|1\rangle$$

by two complex numbers a and b satisfying $|a|^2 + |b|^2 = 1$. When this qubit is measured, **0** is obtained with probability $|a|^2$, and **1** is obtained with probability $|b|^2$. Once a qubit is measured, the information of the original qubit is lost.

In more general terms, n qubits represent the superposition of 2^n states. By using

$|x\rangle = |x_{n-1}\rangle \dots |x_0\rangle$, where $x = \sum_{i=0}^{n-1} x_i 2^i$, the superposition of 2^n states is represented as

$$\sum_{x=0}^{2^n-1} a_x |x\rangle \text{, where } a_0, \dots, a_{2^n-1} \in \mathbb{C} \text{ and } \sum_{x=0}^{2^n-1} |a_x|^2 = 1.$$

We now define $\boldsymbol{a}$ as $\boldsymbol{a} = \begin{bmatrix} a_0 & \dots & a_{2^n-1} \end{bmatrix}^{\mathrm{T}}$. A quantum computation realizes the desired computational results by applying quantum gates on qubits. A quantum gate application corresponds to calculating $U\boldsymbol{a}$, where U is the unitary matrix in 2^n dimensions corresponding to the quantum gate. The desired computational results are then given by measuring the qubits.

We now explain the current development of quantum computers. As noted above, perfect quantum computers are not realized. However, imperfect but functional quantum computers, called Noisy Intermediate-Scale Quantum (NISQ) computers [94], have emerged with machines from IBM [19, 48], Google [6], Rigetti [104], IonQ [77], and Honeywell [91], all accessible through the web. The next decade is likely to see a considerable rise in NISQ computers in the number of qubits and executable circuit sizes.

Unfortunately, NISQ computers have high error rates in each operation. These errors propagate as the calculation proceeds. The errors then accumulate in each quantum gate's output, and the correct result cannot be extracted. Thus, NISQ computers cannot realize an arbitrary large-scale quantum computation, and the error rate in quantum computers must be reduced.

To realize a computation with high accuracy, research on Fault-Tolerant Quantum (FTQ) computers is ongoing [23, 93, 106]. In FTQ computers, an accurate qubit called a logical qubit is realized by applying an error-correction circuit on many qubits called physical qubits. By using logical qubits, FTQ computers realize large-scale quantum algorithms such as Shor's [103] and Grover's [39] algorithms. Error-correction circuits are crucial in FTQ computers, and many researchers have studied error-correction circuits [23, 62, 109].

We now review threats of quantum attacks on the currently used public-key cryptosystems, such as the RSA and elliptic curve schemes. As noted above, Shor's algorithm [103] solves the integer factoring and elliptic curve discrete logarithm problems in polynomial-time. The detailed circuit of Shor's algorithm must be considered to discuss this threat more accurately. After the proposal, there are many studies on constructing Shor's algorithm [8, 9, 22, 27, 28, 35, 41, 68, 88, 100, 113, 114, 116, 121]. The first detailed construction is given by Vedral et al. [116], and the following research minimized the number of qubits, the number of gates, or the depth of the quantum circuit. Specifically, the depth of the quantum circuit provides the time for running the quantum algorithm. Thus, we must minimize the depth when we evaluate the security against quantum computers.

The quantum circuit is composed of one-qubit gates, two-qubit gates, and Toffoli gates. The Toffoli gate is one of the three-qubit gates and corresponds to an AND gate in classical computers. Specifically, Toffoli gates are calculated as

$$|x\rangle |y\rangle |z\rangle \to |x\rangle |y\rangle |z \oplus (x \wedge y)\rangle ,$$

which has a one-to-one correspondence between the input and output states. The Toffoli gate is composed of one- and two-qubit gates [81] and requires a higher cost than quantum gates with fewer qubits. The number of Toffoli gates is commonly used as the computational cost, particularly on FTQ computers [35, 52, 114]. In the most recent study, Gidney and Ekerå [35] evaluated the computational cost of Shor's algorithm based on the number of Toffoli gates. This study is based on the physical construction of Toffoli

gates, namely, CCZ factories [36, 37] and shows that the 2048-bit RSA scheme can be broken within eight hours. Thus, this study describes the security of the RSA scheme in a realistic environment.

Reducing the number of Toffoli gates is one of the strategies for optimization, and another strategy exists. As noted above, the Toffoli gate can be decomposed into quantum gates with fewer qubits. By decomposing the Toffoli gate, the depth of a quantum circuit is reduced by reordering and combining quantum gates [71, 88]. Moreover, the depth of a quantum circuit can be reduced by using relative-phase Toffoli gates [33, 70]. The relative-phase Toffoli gates do not realize the Toffoli gate entirely. Using relative-phase Toffoli gates appropriately, the accurate computation is realized in total [70, 111]. Significantly, Gidney [33] proposed the relative Toffoli gates optimized on FTQ computers, and Maslov [70] proposed the relative Toffoli gates optimized on NISQ computers. We should discuss the computational cost more precisely by the optimal decomposition of Toffoli gates.

1.2 Our Motivation and Contribution

In this thesis, we apply a security analysis of public-key cryptosystems against side-channel attacks and quantum attacks. First, we evaluate security against side-channel attacks on the CRT-RSA scheme [76] which is a currently used public-key cryptosystem. We propose a new recovery method for the CRT-RSA secret keys when the sliding window leakage is extracted. Chapter 3 describes the security of the CRT-RSA scheme when the correct sliding window leakage is extracted. Chapter 4 gives the security analysis of the CRT-RSA scheme against noisy sliding window leakage. Next, Chapter 5 gives the security analysis against the side-channel attacks on the Ring-LWE and Module-LWE based schemes, which are next-generation public-key cryptosystems. We evaluate the security of the most basic scheme, namely, the LPR cryptosystem [67]. We focus on the security of the LPR cryptosystem against leakage from the NTT. Finally, Chapter 6 describes the computational cost for a controlled modular adder, which is the core arithmetic of Shor's algorithm. We evaluate the security of the RSA scheme against quantum computers.

1.2.1 Improved CRT-RSA Secret Key Recovery Method from Sliding Window Leakage

In the previous research [10], a method for recovering the CRT-RSA secret keys from the correct square-and-multiply sequences is proposed. Two methods are used to recover the CRT-RSA secret keys. In the first method, the CRT-RSA secret keys are recovered partially, after which all bits are recovered. In the second method, the CRT-RSA secret keys are recovered directly from square-and-multiply sequences. Both of these methods employ Heninger-Shacham's method [43] to recover the CRT-RSA secret keys. The second method recovers more secret keys than the first method. The second method specifically recovers the CRT-RSA secret keys in polynomial time when $w \leq 4$. Moreover, the second method recovers 13% of the CRT-RSA secret keys when $w = 5$ at the 2048-bit CRT-RSA scheme.

Their results might have been more interesting if we focus on information, partially recovered CRT-RSA secret keys, from the correct square-and-multiply sequences. Their method deals with two methods separately, but their two methods similarly recover the CRT-RSA secret keys by Heninger-Shacham's method. Thus, more secret keys are recovered by combining these two methods. We then aim to recover more secret keys when $w = 5$.

To improve the original method, we study in more depth a technique for recovering bits of the CRT-RSA secret keys and propose a new recovery method for CRT-RSA secret keys. First, we give the exact rate of recovered bits of the CRT-RSA secret keys from only a square-and-multiply sequence in Section 3.3. Next, we propose a new method for recovering additional bits with high accuracy in Section 3.4. Finally, we propose a new method for recovering the CRT-RSA secret keys in Section 3.5. In our proposed method, we combine two original methods [10]. Moreover, we extend the original method by combining it with Kunihiro et al.'s method [59]. Our method then recovers 21% of the CRT-RSA secret keys when $w = 5$, whereas the original method recovers 13% of the CRT-RSA secret keys.

1.2.2 Recovering CRT-RSA Secret Keys from Noisy Square-and-Multiply Sequences

We evaluate the security to be a more realistic situation than previous research [10]. The previous research constructs the recovery method for the CRT-RSA secret key on the correct square-and-multiply sequences. This recovery method heavily uses the correctness of square-and-multiply sequences.

However, in practice, the previous method [10] fails to recover the CRT-RSA secret keys because there are errors in extracted square-and-multiply sequences. As noted in Section 1.1.2, Oonishi and Kunihiro [83, 84] consider a noisy case. Their studies only give the theoretical bound of the error rate when $w \leq 2$, and we should give a theoretical bound on a more extensive w.

We describe a theoretical analysis on recovering the CRT-RSA secret keys from noisy square-and-multiply sequences. First, we propose a new method for recovering the CRT-RSA secret keys from noisy square-and-multiply sequences in Section 4.3. Our method is faster than the previous method [83]. Next, we calculate the number of errors for which our method is applied in Section 4.4. Specifically, we theoretically show the tolerable error rates of our methods when $w \leq 4$. Finally, we conduct numerical experiments of our method and verifies that our algorithm matches our analysis in Section 4.5.

1.2.3 Exact Security Analysis of the Ring-LWE and Module-LWE Based Schemes against NTT Leakage

Section 1.1.2 explained Primas et al.'s attack [95] briefly. Their attack uses a belief propagation to recover the LPR secret keys. More correctly, their attack adopts the loopy belief propagation in an NTT. Primas et al. then tried to evaluate threats on actual power traces and simulated power traces generated from the noisy Hamming weight model. Moreover, they measure how many secret keys are recovered in each trace.

However, the fundamental problem is that convergence of the loopy belief propagation is not guaranteed. The loopy belief propagation does not always converge to the accurate values, and the property of the loopy belief propagation is not studied well. Their security evaluation is then based on the experiments using obtained traces. Thus, we should provide the exact security analysis to clarify the threats to the NTTs.

We analyze the security of the LPR cryptosystem [67] against side-channel attacks based on the exact values. We focus on the LPR cryptosystem using the Cooley-Tukey NTT and the Gentleman-Sande INTT. To analyze the security of the LPR cryptosystem, we adopt the erasure model on the multiplication. We assume that we fail to obtain the input of the multiplication with probability δ. First, we propose an algorithm for recovering coefficients of the secret polynomial in Section 5.3. Next, we analyze our algorithm in Section 5.4. To

calculate the number of recovered coefficients, we adopt an iterative method, which always converges. We analyze the full recovery of LPR secret keys in Section 5.5 and show that our method recovers the secret keys when $\delta \leq 0.78$ at the current computational power. Finally, we described the numerical experiments conducted, verifying that our method recovers the secret keys when $\delta \leq 0.78$.

1.2.4 Security Evaluation of RSA Scheme under an Efficient Construction of a Quantum Controlled Modular Adder

We discuss the security of the RSA scheme against quantum attacks. Shor's algorithm breaks the RSA scheme, and the computational cost of this algorithm must be evaluated. In Shor's algorithm, a modular exponentiation step dominates the total cost. Many studies on its construction have thus been conducted [8, 9, 22, 28, 35, 68, 88, 113, 114, 116, 121]. One strategy realizes modular exponentiation through the repeated calling of controlled modular additions. Thus, if the cost of a controlled modular adder is reduced, the total cost will also be reduced. Therefore, the computational cost of a controlled modular adder must be accurately evaluated.

We discuss a controlled modular adder with a small depth for FTQ and NISQ computers. We provide a more efficient construction compared to Van Meter-Itoh's approach [113] based on a carry-lookahead adder [26]. In Section 6.3, we propose the constriction of a new carry-lookahead adder. Our construction realizes the 2/3 depth of the original construction. In Section 6.4, we discuss the decomposition of a Toffoli gate and reduce depth for FTQ or NISQ computers, respectively. In Section 6.5, we evaluate the security of the RSA scheme against quantum attacks. We evaluate the security against FTQ computers based on our proposed controlled modular adder. By focusing on the executable depth based on IBM's plan for the development of quantum computers [49], we estimate when the RSA scheme will be broken. Our estimation shows that quantum computers will break the 2048-bit RSA scheme 28.2 years later, which is 1.4 years earlier than Van Meter-Itoh's construction.

1.3 Organization of this Thesis

This thesis has seven chapters. In Chapter 2, we describe the preliminaries. In Chapters 3 and 4, we evaluate the security of the CRT-RSA scheme against side-channel attacks. We propose an improved recovery method for the CRT-RSA secret keys from the correct sliding window leakage in Chapter 3. We propose the recovery method for the CRT-RSA secret keys from the noisy sliding window leakage in Chapter 4. In Chapter 5, we evaluate the security of the Ring-LWE and Module-LWE based schemes against side-channel attacks. In Chapter 6, we propose an efficient construction of a controlled modular adder by quantum computers. Moreover, we evaluate the security of the RSA scheme against quantum attacks. In Chapter 7, we conclude this thesis.

Chapter 2

Preliminaries

This chapter provides the preliminaries of this thesis. Section 2.1 introduces the preliminaries on the CRT-RSA scheme. Section 2.2 describes the preliminaries on the Ring-LWE and Module-LWE based schemes. Section 2.3 introduces the preliminaries on Shor's algorithm. Specifically, we give the decomposition of Shor's algorithm into quantum controlled modular adders.

2.1 Preliminaries on the CRT-RSA Scheme

This section describes the preliminaries on security analysis of the CRT-RSA scheme. Section 2.1.1 introduces the CRT-RSA scheme [76]. Section 2.1.2 explains the left-to-right sliding window method used in the CRT-RSA scheme.

2.1.1 CRT-RSA Scheme

We now introduce the CRT-RSA scheme, particularly the CRT-RSA encryption and signature scheme. Before introducing the CRT-RSA encryption and signature scheme, we introduce the standard RSA scheme [99]. This scheme comprises public keys (N, e) and a secret key (p, q, d). Here, p and q are $n/2$ bit prime numbers; in addition, public keys (N, e) and a secret key d satisfy $N = pq$ and $ed \equiv 1 \bmod (p-1)(q-1)$. In the standard RSA encryption scheme, a plaintext m is encrypted by calculating $c = m^e \bmod N$, and ciphertext c is decrypted by calculating $m = c^d \bmod N$. In the standard RSA signature scheme, a signature on m is generated by calculating $\sigma = h(m)^d \bmod N$, and verify the signature σ by checking $h(m) = \sigma^e \bmod N$. In this signature scheme, h is a secure hash function. These two RSA schemes are composed of two modular exponentiations: $x^e \bmod N$, using a public key e, and $x^d \bmod N$, using a secret key d. Although the small public key e is used, such as $2^{16} + 1 = 65537$, a larger secret key d is also applied. Therefore, the implementation time may be longer during decryption and signing than encryption and verification, respectively.

The CRT-RSA scheme [76] realizes faster decryption and signing by applying the Chinese Remainder Theorem (CRT) decomposition on a secret key d. The secret keys $d_p := d \bmod p - 1$, $d_q := d \bmod q - 1$, and $q_p := q^{-1} \bmod p$ are added into the CRT-RSA scheme. Encryption is the same as that of the standard RSA scheme. In decryption or signing, two modular exponentiations, $x^{d_p} \bmod p$ and $x^{d_q} \bmod q$, are calculated using secret keys. Then, $x^d \bmod N$ is calculated by applying CRT on these two values. The calculation of $x^d \bmod N$ is approximately 4-times faster in the CRT-RSA scheme than the standard RSA scheme because bases and exponents are half of the bits.

Algorithm 1 Left-to-right sliding window method [72]

Input: $c, d = (d_t, d_{t-1}, \cdots, d_0)_2$, window size $w \geq 1$
Output: c^d
Step 1: Precomputation
  $c_1 = c, c_2 = c^2$
  for $i = 1$ **to** $2^{w-1} - 1$
    $c_{2i+1} = c_{2i-1} \cdot c_2$
  end for
Step 2: Exponentiation
  $A = 1, i = t$
  while $i \geq 0$
    if $d_i = 0$
      $A = A^2$ (Squaring)
      $i = i - 1$
    else
      Find the longest bit-string $d_i \cdots d_l$ such that $i - l + 1 \leq w$ and $d_l = 1$.
      $A = A^{2(i-l+1)} \cdot c_{(d_i \cdots d_l)_2}$ (Squaring and Multiplication)
      $i = l - 1$
    end if
  end while
return A

Table 2.1: Computational cost of modular exponentiation using 1024-bit exponent

	w	1	2	3	4	5	6	7
#(Multiplication)	Precomputation	0	1	3	7	15	31	63
	Exponentiation	512	341	256	205	171	146	128
	Total	512	342	259	212	186	177	191
	Memory [KByte]	0.13	0.26	0.51	1.0	2.0	4.1	8.2

2.1.2 Left-to-Right Sliding Window Method

The left-to-right sliding window method is given in Algorithm 1. Before a modular exponentiation is calculated, the values of c^i for odd i values satisfying $1 \leq i \leq 2^w - 1$ are precomouted. Then, bits are read from the MSB side to the LSB side. If the value of bit is **0**, a squaring (**S**) is conducted once. If the value of bit is **1**, more $(w-1)$-bits are read, and w squarings (**S**) and one multiplication (**M**) are conducted. Therefore, the number of **S** is the same as the number of bits of the secret exponent d.

In Algorithm 1, the exponentiation given as Step 2 can be calculated with fewer multiplications for a larger w, whereas the number of squaring operations does not change. Specifically, when an n-bit exponent is used, the number of multiplications is $n/(w+1)$. However, to deal with the bits simultaneously, the precomputation given as Step 1 is required. Multiple candidate values of c^i in memory must be stored, and the number of stored values grows exponentially large at w. When an n-bit exponent is used, the number of multiplications is $2^{w-1} - 1$, and the amount of memory used is $2^{w-1}n/8$ bytes.

We give an example of the cost of exponentiation using a 1024-bit exponent used in the 2048-bit CRT-RSA scheme. Table 2.1 shows the cost of exponentiation using a 1024-bit exponent. Table 2.1 describes that the number of multiplications is minimized at $w = 6$.

However, a smaller w must be used in devices with small memory, such as a smartcard.

2.2 Preliminaries on the Ring-LWE and Module-LWE Based Schemes

This section describes preliminaries on the security analysis of the Ring-LWE and Module-LWE based schemes. Section 2.2.1 introduces notations on the Ring-LWE and Module-LWE based schemes. Section 2.2.2 provides the lattice problems, the Ring-LWE [66] and Module-LWE problem [12]. Section 2.2.3 introduces the Lyubashevsky et al.'s cryptosystem (LPR cryptosystem) [67]. Section 2.2.4 describes the construction of a Number Theoretic Transform (NTT), particularly the Cooley-Tukey NTT and Gentleman-Sande INTT [63].

2.2.1 Notations

We now introduce the notations. Here, $\mathbb{Z}$ is defined as the set of integers, namely, $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$. Let q be a positive integer. $\mathbb{Z}_q$ is defined as the set of remainders of the division of the elements in $\mathbb{Z}$ by q, namely, $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$. $\mathbb{Z}$ and $\mathbb{Z}_q$ are rings, and the addition and multiplication are defined on $\mathbb{Z}$ and $\mathbb{Z}_q$. Additions and multiplications on elements in $\mathbb{Z}$ and $\mathbb{Z}_q$ will be applied. These operations are conducted after converting the element in $\mathbb{Z}$ into $\mathbb{Z}_q$ by calculating the remainder of the division by q.

Next, we define the polynomial ring. Let N be a positive integer. $\mathbb{Z}[x]$ is defined as the set of polynomials whose coefficients are in $\mathbb{Z}$. The set of polynomials $\mathbb{Z}[x]/\left\langle x^N + 1 \right\rangle$ is defined as the set of remainders of the division of the elements in $\mathbb{Z}[x]$ by $x^N + 1$. In the following discussion, $\mathbb{Z}[x]/\left\langle x^N + 1 \right\rangle$ is simply written as $\mathcal{R}$. $\mathcal{R}$ is a ring, and the addition and multiplication are defined on $\mathcal{R}$. $\mathbb{Z}_q[x]$ is also defined as the set of polynomials whose coefficients are in $\mathbb{Z}_q$. $\mathbb{Z}_q[x]/\left\langle x^N + 1 \right\rangle$ is defined as the set of remainders of the division of the elements in $\mathbb{Z}_q[x]$ by $x^N + 1$. In the following discussion, $\mathbb{Z}_q[x]/\left\langle x^N + 1 \right\rangle$ is simply written as $\mathcal{R}_q$. $\mathcal{R}_q$ is also a ring, and the addition and multiplication are defined on $\mathcal{R}_q$.

2.2.2 Lattice Problems

In this subsection, we introduce the Ring-LWE [66] and the Module-LWE [12] problems. The Ring-LWE and Module-LWE problems are variants of the learning with errors (LWE) problem [98]. Before introducing the Ring-LWE and Module-LWE problems, we introduce the LWE problem as Definition 1.

Definition 1 (LWE problem [98]) Let l and m be positive integers, q be a prime number, and χ be a probability distribution over $\mathbb{Z}$. We then choose $\boldsymbol{s} \in \mathbb{Z}_q^l$, $\boldsymbol{a}_i \in \mathbb{Z}_q^l$, and $e_i \in \mathbb{Z}$, where $1 \leq i \leq m$. $\boldsymbol{s}$ is chosen uniformly from $\mathbb{Z}_q^l$, and $\boldsymbol{a}_i$'s are chosen independently and uniformly from $\mathbb{Z}_q^l$. e_i's are chosen independently according to χ. Let $b_i = \boldsymbol{a}_i^T \boldsymbol{s} + e_i \bmod q$, where T indicates the transpose. The LWE problem is defined as the problem of recovering $\boldsymbol{s}$ from m pairs of $(\boldsymbol{a}_i, b_i)$.

In the LWE problem, e_i's are the small error terms. Typically, we set χ as the discrete Gaussian distribution D_σ defined as Definition 2.

Definition 2 (Discrete Gaussian distribution on $\mathbb{Z}$ [98]) Let σ be a positive real number.

The discrete Gaussian distribution is defined on $x \in \mathbb{Z}$ as

$$D_\sigma(x) = \frac{\exp\left(-\frac{x^2}{2\sigma^2}\right)}{\sum_{y\in\mathbb{Z}} \exp\left(-\frac{y^2}{2\sigma^2}\right)}$$

When e_i's are all 0, we can easily recover $\boldsymbol{s}$. The error terms e_i's are why the LWE problem is difficult. Many studies have solved the LWE problem using the BKW algorithm [2, 11, 54] or a lattice reduction [17, 61]. Significantly, the LWE problem's challenge is held [112] to evaluate the difficulty of the LWE problem.

We now define the Ring-LWE [66] and Module-LWE [12] problems. In the following discussion, a polynomial is sampled uniformly from $\mathcal{R}_q$ by choosing each coefficient independently and uniformly from $\mathbb{Z}_q$.

The Ring-LWE problem is an extension of the LWE problem on the polynomial ring, namely, $\mathcal{R}_q$. The Ring-LWE problem is defined as Definition 3.

Definition 3 (Ring-LWE problem [66]) Let m be a positive integer, N be a power of 2, and q be a prime number. Moreover, let χ be a probability distribution over $\mathbb{Z}$. Then, we choose $s \in \mathcal{R}_q$, $a_i \in \mathcal{R}_q$, and $e_i \in \mathcal{R}$, where $1 \leq i \leq m$. s is chosen uniformly from $\mathcal{R}_q$, and a_i's are chosen independently and uniformly from $\mathcal{R}_q$. The coefficients of e_i's are chosen independently according to χ. Then, let $b_i = a_i s + e_i$. The Ring-LWE problem is defined as the problem of recovering s from m pairs of (a_i, b_i).

The Module-LWE problem is a generalization of both the LWE and Ring-LWE problems. The Module-LWE problem is defined on a vector of polynomials, namely, $\mathcal{R}_q^l$. The Module-LWE problem is defined as Definition 4.

Definition 4 (Module-LWE problem [12]) Let l and m be positive integers, N be a power of 2, and q be a prime number. Moreover, let χ be a probability distribution over $\mathbb{Z}$. We then choose $\boldsymbol{s} \in \mathcal{R}_q^l$, $\boldsymbol{a}_i \in \mathcal{R}_q^l$, and $e_i \in \mathcal{R}$, where $1 \leq i \leq m$. Each element of $\boldsymbol{s}$ is chosen independently and uniformly from $\mathcal{R}_q$. Each element of $\boldsymbol{a}_i$ is chosen independently and uniformly from $\mathcal{R}_q$. Coefficients of e_i's are chosen independently according to χ. Let $b_i = \boldsymbol{a}_i^T \boldsymbol{s} + e_i$, where T indicates a transpose. The Module-LWE problem is defined as the problem of recovering $\boldsymbol{s}$ from m pairs of $(\boldsymbol{a}_i, b_i)$.

When $N = 1$, the Module-LWE problem is as same as the LWE problem. When $l = 1$, the Module-LWE problem is as same as the Ring-LWE problem.

2.2.3 LPR Cryptosystem [67]

First, we introduce the parameters used for the LPR cryptosystem. In the LPR cryptosystem, three parameters (N, q, σ) are used. Let N be the power of two, namely, $N = 2^n\,(n \in \mathbb{N})$. Then, let q be a prime number satisfying $q \equiv 1 \bmod 2N$. Moreover, let σ be a real positive number $\Theta\left(\sqrt{q}N^{-1/4}\right)$. Based on the parameters, N, q, and σ, we describe the key generation, encryption, and decryption in the LPR cryptosystem. These operations are applied on the polynomial ring $\mathcal{R}_q$.

Key Generation. Two polynomials $s, e \in \mathcal{R}$ are generated by sampling each coefficient of s and e from the discrete Gaussian distribution D_σ. Then, a polynomial $a \in \mathcal{R}_q$ is sampled randomly, and b is computed as $e - as$. At last, public keys (a, b) and a secret key s are outputted.

Encryption. We explain the encryption of a plaintext $m \in \{0,1\}^N$. The plaintext m is encoded on the elements in $\mathcal{R}_q$. Encoding is executed using

$$\bar{m} = \sum_{i=1}^{N} \left\lfloor \frac{q}{2} \right\rceil m_i x^{i-1},$$

where $\lfloor x \rceil$ is defined as the nearest integer of x, and m_i is the i-th bit of m. Then, polynomials r, e_1, and $e_2 \in \mathcal{R}_q$ are sampled. Each coefficient of r, e_1, and e_2 is sampled from the discrete Gaussian distribution D_σ. At last, ciphertexts (c_1, c_2) are computed by $c_1 = ar + e_1$ and $c_2 = br + e_2 + \bar{m}$.

Decryption. We explain the decryption of the ciphertext (c_1, c_2). First, a polynomial $m^* = c_1 s + c_2$ is calculated. Plaintext m is recovered as follows:

- If $\lceil q/4 \rceil \leq m_i^* \leq \lfloor 3q/4 \rfloor$, m_i is set as 1.
- Otherwise, m_i is set as 0.

In decryption, m^* satisfies

$$\begin{aligned} m^* &= c_1 s + c_2 = (ar + e_1)\, s + (br + e_2 + \bar{m}) \\ &= \bar{m} + re + e_1 s + e_2, \end{aligned}$$

and $re + e_1 s + e_2$ is sufficiently small compared to q. Thus, decryption succeeds with a sufficiently high probability.

2.2.4 Number Theoretic Transform (NTT)

The Ring-LWE and Module-LWE based cryptosystems employ the multiplication on $\mathcal{R}_q$. If we conduct a multiplication of polynomials naively, it costs $O\left(N^2\right)$ operations. However, using an NTT and the inverse of the NTT (INTT), the multiplication of polynomials $a, b \in \mathcal{R}_q$ is realized by

$$ab = \mathrm{INTT}(\mathrm{NTT}(a) * \mathrm{NTT}(b)),$$

where $*$ is a coefficient-wise multiplication. Because the NTTs are $O\left(N \log N\right)$ time transformations, a multiplication of polynomials can be conducted in $O\left(N \log N\right)$ time. Moreover, the addition and subtraction of polynomials are calculated as follows: $a + b = \mathrm{INTT}(\mathrm{NTT}(a) + \mathrm{NTT}(b))$, and $a - b = \mathrm{INTT}(\mathrm{NTT}(a) - \mathrm{NTT}(b))$, respectively.

We now describe NTT and INTT in more detail. To explain the NTT, we introduce ω and γ satisfying $\omega^N \equiv 1 \bmod q$ and $\gamma^2 \equiv \omega \bmod q$. These ω and γ exist because $q \equiv 1 \bmod 2N$. Then, the NTT of $a = \sum_{i=0}^{N-1} a[i] x^i \in \mathcal{R}_q$ is given as $\mathrm{NTT}(a) = \sum_{j=0}^{N-1} \hat{a}[j] x^j$, where $\hat{a}[j]$ is

$$\hat{a}[j] = \sum_{k=0}^{N-1} a[k] \gamma^k \omega^{jk} \bmod q \ (0 \leq j \leq N-1).$$

The inverse of $\mathrm{NTT}(a)$ is given as

$$a[l] = N^{-1} \gamma^{-l} \sum_{j=0}^{N-1} \hat{a}[j] \omega^{-jl} \bmod q \ (0 \leq l \leq N-1).$$

Algorithm 2 Cooley-Tukey NTT [63]

Input: N, q, $a \in \mathcal{R}_q$, and γ
Output: NTT$(a) \in \mathcal{R}_q$ in bit-reversed order
$e = 1$, $t = N/2$
for $i = 0$ **to** $n - 1$
  $s = 0$
  for $j = 0$ **to** $e - 1$
    Calculate e', the bit-reversed value of $e + j$
    for $k = s$ **to** $s + t - 1$
      Butterfly Operation
      $\text{temp}[k] = a[k]$
      $\text{temp}\,[k + t] = a\,[k + t]\,\gamma^{e'} \bmod q$
      $a[k] = \text{temp}[k] + \text{temp}[k + t] \bmod q$
      $a[k + t] = \text{temp}[k] - \text{temp}[k + t] \bmod q$
    end for
    $s = s + 2t$
  end for
  $e = 2e$, $t = t/2$
end for
Return NTT$(a) = a$

Algorithm 3 Gentleman-Sande INTT [63]

Input: N, q, $a \in \mathcal{R}_q$ in bit-reversed order, and γ
Output: INTT$(a) \in \mathcal{R}_q$
$e = N/2$, $t = 1$
for $i = n - 1$ **down to** 0
  $s = 0$
  for $j = 0$ **to** $e - 1$
    Calculate e', the bit-reversed value of $e + j$
    for $k = s$ **to** $s + t - 1$
      Butterfly Operation
      $\text{temp}[k] = a[k] + a\,[k + t] \bmod q$
      $\text{temp}\,[k + t] = a[k] - a\,[k + t] \bmod q$
      $a[k] = \text{temp}[k]$
      $a[k + t] = \text{temp}\,[k + t]\,\gamma^{-e'} \bmod q$
    end for
    $s = s + 2t$
  end for
  $e = e/2$, $t = 2t$
end for
for $k = 0$ **to** $N - 1$
  $a[k] = a[k] \cdot N^{-1} \bmod q$
end for
Return INTT$(a) = a$

Next, we describe the implementation of the Cooley-Tukey NTT and Gentleman-Sande INTT, adopted by NIST candidates. These NTTs are implemented using butterfly oper-

ations [38, 63] similar to a fast Fourier transformation. Algorithm 2 shows the Cooley-Tukey NTT, and Algorithm 3 shows the Gentleman-Sande INTT. In the Cooley-Tukey NTT and Gentleman-Sande INTT, N coefficients of a are inputted. These N values are renewed in each i by repeating the operation in each butterfly. It should be noted that the coefficients of an NTT-form polynomial are stored in bit-reversed order to realize efficient computation. From Algorithms 2 and 3, the NTT and INTT are composed of $(N \log N)/2$ butterfly operations. As a difference, a multiplication of $N^{-1} \bmod q$ occurs in each coefficient in the INTT.

2.3 Preliminaries on Shor's Algorithm

This section reviews the decomposition of Shor's algorithm into controlled modular adders. Section 2.3.1 introduces the quantum gate set used in this thesis. Section 2.3.2 describes Shor's algorithm. Section 2.3.3 details the decomposition of the modular exponentiation into controlled modular adders.

2.3.1 Quantum Gate Set

This subsection describes the set of quantum gates. A quantum gate set is composed of the following gates:

- **One-qubit gate.** X gate, Y gate, Z gate, H gate, S gate, and T gate
- **Two-qubit gate.** CNOT gate

As noted in Section 1.1.3, these gates can be represented as the unitary matrices. Specifically, the one-qubit gates can be represented as the unitary matrix in two dimensions, and the two-qubit gates can be represented as the unitary matrix in four dimensions.

One-Qubit Gate

A qubit is represented as $a|0\rangle + b|1\rangle$, and we represent this qubit as the vector $\begin{bmatrix} a \\ b \end{bmatrix}$. We now describe the X gate. The corresponding unitary matrix of the X gate is given as follows: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. When the X gate is applied on $a|0\rangle + b|1\rangle$,

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} b \\ a \end{bmatrix},$$

and $b|0\rangle + a|1\rangle$ is obtained. Thus, $|0\rangle$ and $|1\rangle$ are swapped.

Similar to the X gate, the corresponding unitary matrices of the other gates are given as follows:

$$Y = \begin{bmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{i} \end{bmatrix}, \text{ and } T = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{\mathrm{i}\pi}{4}\right) \end{bmatrix}.$$

Two-Qubit Gate

We now describe the CNOT gate. The CNOT gate realizes the following operation:

$$|x\rangle|y\rangle \to |x\rangle|y \oplus x\rangle.$$

Specifically, two qubits are represented as

$$a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle,$$

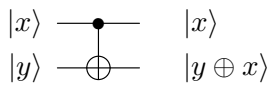

Fig. 2.1: CNOT gate. In the CNOT gate, the control bit is the first qubit, and the target bit is the second qubit.

and the CNOT gate converts these qubits into

$$a_0 |00\rangle + a_1 |01\rangle + a_3 |10\rangle + a_2 |11\rangle .$$

The corresponding unitary matrix of the CNOT gate is given as follows:

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} .$$

In this thesis, the CNOT gate is represented as Figure 2.1.

Universal Computation in Quantum Computers

In classical computers, any computation is composed of only NAND gates. Similarly, in quantum computers, any computation is realized with arbitrary precision by H gates, T gates, and CNOT gates [81]. It should be noted that S gates are required to realize any computation in FTQ computers.

We now describe the decomposition of Toffoli gates as an example. Toffoli gates are useful in quantum computations and realize the following:

$$|x\rangle |y\rangle |z\rangle \to |x\rangle |y\rangle |z \oplus (x \wedge y)\rangle . \tag{2.1}$$

In this thesis, the standard Toffoli gate (ST) is represented as Figure 2.2. Specifically, the corresponding unitary matrix of ST in the computational basis is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} . \tag{2.2}$$

The standard Toffoli gate decomposition [81] is given in Figure 2.3. It should be noted that gates with † indicate their inverse gates.

Cost of Gates in Actual Quantum Computers

In the above discussion, we describe the quantum gate set. However, these gates do not have the same cost in FTQ or NISQ computers. In FTQ computers, the T gate has a significantly higher cost than other gates. We describe this in more detail in Section 6.2.1. In NISQ computers, the CNOT gate has a significantly higher cost because running a CNOT gate requires a longer time and causes a higher error than other gates [19, 48]. Thus, we should consider these properties in each device. In our results proposed in Chapter 6, we focus on a reduction of these gates.

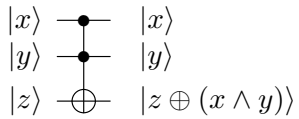

Fig. 2.2: In the Toffoli gate, the control bits are the first and second qubits, and the target bit is the third qubit.

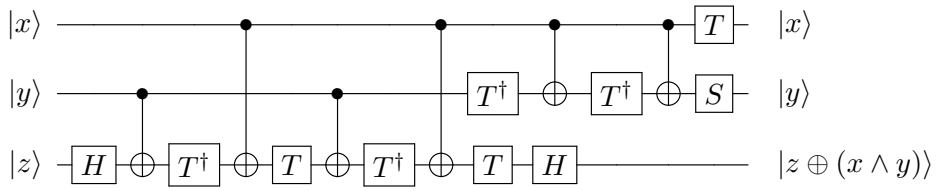

Fig. 2.3: Standard decomposition of Toffoli gate [81].

2.3.2 Shor's Algorithm

We now describe Shor's algorithm [103]. Shor's algorithm calculates the order defined as Definition 5.

Definition 5 (Order [81]) Let a and N be the positive integers, which are relatively prime and $a < N$. The order of a in modulo N is defined as the smallest positive integer r satisfying $a^r \bmod N = 1$.

The order finding problem is defined as the calculating order of a in modulo N. By solving the order finding problem, the integer factorization and discrete logarithm problems are solved. Each problem is defined as follows:

Definition 6 (Integer factorization problem) Let N be a product of two distinct prime numbers p and q. The integer factorization problem is defined as the problem of calculating p and q when N is given.

Definition 7 (Discrete logarithm problem) Let a and N be positive integers that are relatively prime, where $a < N$. Moreover, let b be the positive integer represented as $b = a^x \bmod N$ with a positive integer x. The discrete logarithm problem is defined as calculating x when a, b, and N are given.

Moreover, the elliptic curve discrete logarithm problem is solved by extending the order finding problem to the commutative group. The integer factorization problem is solved as Algorithm 4. In this algorithm, $\gcd(a, b)$ is defined as the common divisor of a and b. Moreover, quantum computation is only used in finding the order, and the other parts are conducted in classical computers.

We now explain Shor's algorithm solving the order finding problem. Figure 2.4 shows Shor's algorithm. The first register has $\lceil 2 \log N \rceil$ qubits, and the second register has $\lceil \log N \rceil$ qubits. As Figure 2.4 indicates, Shor's algorithm is composed of modular exponentiation and the inverse of a quantum Fourier transform. This thesis focuses on modular exponentiation because the computational cost is dominant.

Algorithm 4 Solving the integer factorization problem [103]

Input: The composite number N
Output: One of the divisors of N

1: Return 2 if N is an even number.
2: Search $c \geq 1$ and $d \geq 2$ satisfying $N = c^d$. If we find such c and d, we return c.
3: Choose a satisfying $1 \leq a \leq N-1$ randomly. Return $\gcd(a, N)$ if $\gcd(a, N) > 1$.
4: Calculate r as the order of a in modulo N by using Shor's algorithm.
5: Go to 6 if r is an even number and satisfies $a^{r/2} \neq -1 \bmod N$. Otherwise, go to 3.
6: Calculate $\gcd\left(a^{r/2}-1, N\right)$ and $\gcd\left(a^{r/2}+1, N\right)$. Return this value if one of these is not 1. Otherwise, go to 3.

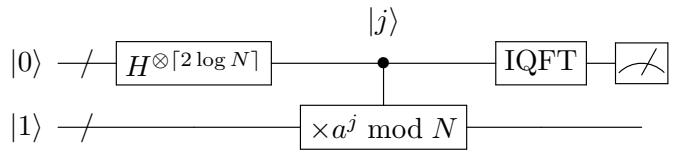

Fig. 2.4: Block-level Shor's algorithm [103] calculating the order of a in modulo N. The first register has $\lceil 2\log N\rceil$ qubits, and the second register has $\lceil \log N\rceil$ qubits. In this figure, $H^{\otimes\lceil 2\log N\rceil}$ means that the H gates are applied on all qubits in the first register, and the superposition of all values in $0 \leq j \leq 2^{\lceil 2\log N\rceil}-1$ is realized. Here, $a^j \bmod N$ is conducted based on the first register $|j\rangle$. IQFT indicates the inverse of the Quantum Fourier Transform.

The above construction is a method for solving the order finding problem. However, a more efficient method for solving the order finding problem of the integer factorization problem is proposed [27], and we describe the method with a small modification [35]. In this method, the integer factorization problem is replaced with the discrete logarithm problem. From Euler's totient theorem, $a^{(p-1)(q-1)} = 1 \bmod N$. Now, the exponent $(p-1)(q-1)$ is divided by r, and

$$(p-1)(q-1) = 0 \bmod r \Leftrightarrow p+q = N+1 \bmod r.$$

Thus, $p+q = N+1 \bmod (p-1)(q-1)$. Therefore, $a^{p+q} = a^{N+1} \bmod N$, and $p+q$ is computed by solving the discrete logarithm problem. The discrete logarithm problem is solved by the circuit given as Figure 2.5. Now, the values of p and q are calculated by solving equations $x^2 - (p+q)x + N = 0$. Thus, the integer factorization problem is solved using the discrete logarithm problem. In this construction, the number of exponent bits is $3m \sim 1.5\lceil \log N\rceil$ qubits, whereas the original construction uses $2\lceil \log N\rceil$ qubits. Therefore, this construction requires approximately 3/4 of the cost of the original construction.

2.3.3 Construction of Modular Exponentiation by Controlled Modular Adders

This subsection describes the construction of modular exponentiation by controlled modular adders. We focus on the efficient construction given in Figure 2.5. Let n be $\lceil \log N\rceil$. The first register then has $1.5n$ qubits, and the second register has n qubits in Figure 2.5.

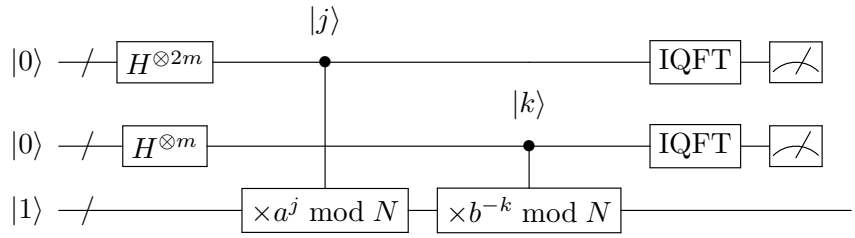

Fig. 2.5: Block-level algorithm solving the discrete logarithm problem given as $a^{p+q} = a^{N+1} \bmod N$ [35]. In this figure, let m be the smallest positive integer satisfying $p+q < 2^m$. The first register has $2m$ qubits, and the second register has m qubits. Moreover, let b be $b = a^{N+1} \bmod N$. In this figure, $H^{\otimes 2m}$ means that the H gates are applied on all qubits in the first register and realize the superposition of all values in $0 \leq j \leq 2^{2m}-1$. Similarly, $H^{\otimes m}$ means that the H gates are applied on all qubits in the second register and realize the superposition of all values in $0 \leq k \leq 2^m - 1$. In addition, $a^j b^{-k} \bmod N$ is applied based on the first register $|j\rangle$ and the second register $|k\rangle$. IQFT indicates the inverse of the Quantum Fourier Transform.

We describe the decomposition as follows:

- Construction of modular exponentiation by controlled modular multipliers
- Decomposition of a controlled modular multiplier into controlled modular adders

Construction of Modular Exponentiation by Controlled Modular Multipliers

In modular exponentiation, a and N satisfying $a < N$ are input. Moreover, $b = a^{N+1} \bmod N$ is input. Then,

$$|j\rangle |k\rangle |1\rangle \rightarrow |j\rangle |k\rangle |a^j b^{-k} \bmod N\rangle \tag{2.3}$$

is conducted. In addition, $|j\rangle$ is composed of n qubits, and $|k\rangle$ is composed of $n/2$ qubits. We rewrite j as $j = \sum_{l_1=0}^{n-1} j_{l_1} 2^{l_1}$, where j_{l_1} indicates each qubit in j. Similarly, we rewrite k as $k = \sum_{l_2=0}^{n/2-1} k_{l_2} 2^{l_2}$, where k_{l_2} indicates each qubit in k. Then,

$$a^j b^{-k} \bmod N = \prod_{l_1=0}^{n-1} \left(a^{2^{l_1}} \bmod N\right)^{j_{l_1}} \prod_{l_2=0}^{n/2-1} \left(b^{-2^{l_2}} \bmod N\right)^{k_{l_2}} \bmod N.$$

Thus, (2.3) is realized by $1.5n$-times controlled modular multipliers. It should be noted that the value of the multiplier can be precomputed and not contained in qubits.

Decomposition of a Controlled Modular Multiplier into Controlled Modular Adders

We decompose a controlled modular multiplier into controlled modular adders through the calculation given as (2.5). In a controlled modular multiplier, a and N satisfying $a < N$ are input, and

$$|x\rangle |j\rangle \rightarrow |x\rangle |ja^x \bmod N\rangle \tag{2.4}$$

is applied. $|x\rangle$ is a single qubit, and we multiply the second register by a based on $|x\rangle$. To realize (2.4), we prepare more n qubits, and conduct the following calculation:

$$|x\rangle |j\rangle |b\rangle \rightarrow |x\rangle |j\rangle |b + xja \bmod N\rangle . \tag{2.5}$$

In (2.5), the third register has n additional qubits. Using (2.5), we calculate the controlled modular multiplier as

$$|x\rangle |j\rangle |0\rangle \rightarrow |x\rangle |j\rangle |xja\rangle \rightarrow |x\rangle |j - xjaa^{-1}\rangle |xja\rangle ,$$

where the final state is

$$|x\rangle |j - xjaa^{-1}\rangle |xja\rangle = \begin{cases} |0\rangle |ja^0\rangle |0\rangle & \text{if } x = 0 \\ |1\rangle |0\rangle |ja^1\rangle & \text{otherwise} \end{cases} .$$

Thus, we swap the second and third registers when $x = 1$, and (2.4) is realized. From the above discussion, the controlled modular multiplier requires two-time calculations, which is given as (2.5).

We now describe the decomposition of (2.5) into controlled modular adders. $|j\rangle$ is constructed using n qubit, and we rewrite j as $j = \sum_{k=0}^{n-1} j_k 2^k$, where j_k indicates each qubit in j. Then,

$$b + xja \bmod N = b + \sum_{k=0}^{n-1} xj_k \left(a2^k \bmod N\right) \bmod N.$$

Thus, (2.5) is realized by n-times the controlled modular adders. Specifically, we conduct a modular adder of $a2^k \bmod N$ with control of $|x\rangle$ and $|j_k\rangle$, which can be summarized using a Toffoli gate. It should be noted that the value of $a2^k \bmod N$ can be precomputed and not contained in qubits. In conclusion, modular exponentiation is constructed by $3n^2$ controlled modular adders. Chapter 6 will provide an efficient controlled modular adder and evaluate the total cost.

Chapter 3

Improved CRT-RSA Secret Key Recovery Method from Sliding Window Leakage

3.1 Introduction

3.1.1 Background

This chapter focuses on the CRT-RSA scheme [76] implemented using the left-to-right sliding window method. We propose the new recovery method for the CRT-RSA secret keys based on the correct square-and-multiply sequences. In a previous study [10], a way to recover the CRT-RSA secret keys from square-and-multiply sequences without a multiplier was discussed. The CRT-RSA secret keys are recovered using two methods. With the first method, the CRT-RSA secret keys are partially recovered, and all secret keys are recovered by using partially recovered bits. With the second method, the CRT-RSA secret keys are recovered directly from square-and-multiply sequences. Both of these methods employ Heninger-Shacham's method [43] to recover the CRT-RSA secret keys. We describe these methods in more detail below.

The first method considers how to recover the CRT-RSA secret keys from square-and-multiply sequences partially. Specifically, a method for recovering bits of partial CRT-RSA secret keys is proposed. Their method applies Heninger-Shacham's method to the recovered bits and recovers the CRT-RSA secret keys. They succeeded in recovering the secret keys of the 1024-bit CRT-RSA scheme when $w = 4$.

The second method researched the recovery method of the CRT-RSA secret keys directly from square-and-multiply sequences. Their algorithm works in polynomial time when $w \leq 4$. Moreover, their method succeeded in recovering 13% of the secret keys of the 2048-bit CRT-RSA scheme when $w = 5$, which is not a range of polynomial time. Thus, we must evaluate the security of the CRT-RSA scheme when $w = 5$.

However, their method for recovering the CRT-RSA secret keys directly from square-and-multiply sequences is simple. Therefore, by extending their method, there is an opportunity to recover more CRT-RSA secret keys when $w = 5$.

To improve their algorithm, we should research why more CRT-RSA secret keys are recovered from square-and-multiply sequences directly than partially recovered bits. We need to conduct more research on bit information of a square-and-multiply sequence. However, this requires more in-depth research. First, the exact behavior for recovering bits of the partial CRT-RSA secret keys has not been thoroughly researched. Although it is shown that more bits can be recovered when applying the method repeatedly, this

situation is discarded owing to rare occurrences. Second, the difference between the two methods is not discussed thoroughly.

Van Vredendaal [115] addressed these problems. First, an optimal recovery method for bits of the CRT-RSA secret key is proposed only from a square-and-multiply sequence. According to this method, bits taking the same value in all candidates are recovered. Second, a method to calculate the number of input candidates corresponding to a given square-and-multiply sequence is proposed. However, this result also requires further research to improve the original method.

3.1.2 Our Contribution

In this study, we obtain three results. This chapter is based on our paper published at the 22nd Annual International Conference on Information Security and Cryptology (ICISC2019) [82].

First, we calculated the exact rate of recovered bits of the secret exponent from only a square-and-multiply sequence. For this purpose, we use the renewal reward theorem [105]. A previous attempt was made to calculate the recovery rate of bits of the secret exponent using the renewal reward theorem [10]. However, only the upper and lower bounds of the recovery rate were calculated. We revisited their analysis and calculated the exact recovery rate from only a square-and-multiply sequence.

Second, we extract embedding information from the non-recovered bits. Specifically, we focus on bits we can determine with high accuracy from among non-recovered bits from the likelihood of **0** or **1**. We developed a random sampling method of bit sequences corresponding to a given square-and-multiply sequence to calculate this likelihood. We developed our method based on Van Vredendaal's approach [115] to calculate the number of input candidates corresponding to a given square-and-multiply sequence. This calculation indicates bits with a large bias regarding the likelihood of each bit value among non-recovered bits.

Finally, we propose a new CRT-RSA secret key recovery algorithm using the likelihood of each bit value as additional information. With our proposed method, we apply Kunihiro et al.'s algorithm [59] on the obtained likelihood of each bit value. We extend the original method [10] based on square-and-multiply sequences directly in combination with Kunihiro et al.'s algorithm. Our new algorithm recovers 21% of the CRT-RSA secret keys when $w = 5$. This result is a significant improvement compared to the 13% recovery of the original method.

3.1.3 Organization of this Chapter

This chapter has six sections. In Section 3.2, we describe the original methods for recovering the CRT-RSA secret keys from square-and-multiply sequences [10]. In Section 3.3, we analyze the recovery rate of bits of the CRT-RSA secret keys from square-and-multiply sequences. In Section 3.4, we propose a method for extracting embedding information from the non-recovered bits. In Section 3.5, we propose a new algorithm for recovering the CRT-RSA secret keys. In Section 3.6, we conclude this chapter.

3.2 Previous Methods for Recovering the CRT-RSA Secret Keys

This section describes two methods proposed in the original study [10]. Before describing the original method, we explain Heninger-Shacham's approach [43] in Section 3.2.1, which is used in both methods of the original research. We then describe the two methods in

Section 3.2.2 and 3.2.3, respectively.

3.2.1 Heninger-Shacham's Method [43]

Heninger and Shacham proposed a method for constructing a CRT-RSA key candidate tree. They construct a CRT-RSA key candidate tree when the public keys (N, e) and parameters $(k_p, k_q) \in \mathbb{Z}^2$ satisfying $ed_p = 1 + k_p(p-1)$ and $ed_q = 1 + k_q(q-1)$ are given. In addition, (k_p, k_q) are initially unknown. However, these values satisfy $0 < k_p, k_q < e$. Moreover, k_p and k_q satisfy $(k_p - 1)(k_q - 1) \equiv k_p k_q N \bmod e$ [50, 120]. Therefore, the number of candidates of (k_p, k_q) is at most $e - 1$.

After calculating (k_p, k_q), the Heninger-Shacham's method recovers the CRT-RSA secret keys from the LSB side. To explain the construction of the candidate tree of the CRT-RSA secret keys, $\tau(x)$ is defined as $\tau(x) = \max_{m \in \mathbb{Z}} 2^m | x$. At each depth of the CRT-RSA secret key candidate tree, partial LSBs of p, q, d_p and d_q are stored. These partial bits are represented as p', q', d'_p and d'_q. At the i-th depth, p' and q' have $(i+1)$ LSBs, d'_p has $(i + 1 + \tau(k_p))$ LSBs, and d'_q has $(i + 1 + \tau(k_q))$ LSBs. Next, we describe the procedure for constructing the candidate tree.

First, Heninger-Shacham's method calculates the LSBs of p, q, d_p, and d_q. Because p and q are odd numbers, $p' = 1$ and $q' = 1$. Moreover, d'_p and d'_q are calculated by using $ed'_p \equiv 1 \bmod 2^{\tau(k_p)+1}$ and $ed'_q \equiv 1 \bmod 2^{\tau(k_q)+1}$, respectively.

Next, we calculate the leaves at the i-th depth of the candidate tree after calculating the leaves at the $(i-1)$-th depth. If the i-th bits from the LSB side of x are denoted as $x[i-1]$, the following equations hold.

$$p[i] + q[i] \equiv (N - p'q')[i], \tag{3.1}$$

$$d_p[i + \tau(k_p)] + p[i] \equiv \left(k_p(p' - 1) + 1 - ed'_p\right)[i + \tau(k_p)], \tag{3.2}$$

$$d_q[i + \tau(k_q)] + q[i] \equiv \left(k_q(q' - 1) + 1 - ed'_q\right)[i + \tau(k_q)]. \tag{3.3}$$

These simultaneous equations have two solutions of $(p[i], q[i], d_p[i + \tau(k_p)], d_q[i + \tau(k_q)])$. Therefore, two leaves are generated from one leaf. Redundancy of the CRT-RSA secret keys is used in these equations. The generation of leaves is repeated until reaching the $(n/2 - 1)$-th depth, and there is always one correct leaf, including the correct CRT-RSA secret keys, out of $2^{n/2}$ candidates. However, finding this correct leaf takes a tremendous amount of time, which increases exponentially with n. Therefore, to search efficiently for the correct CRT-RSA secret keys, we adopt a bound-and-branch strategy using side-channel information, such as square-and-multiply sequences, as conducted in the previous study [10].

3.2.2 Recovering CRT-RSA Secret Keys through Partially Recovered Bits

This subsection describes the method for recovering the CRT-RSA secret keys through partially recovered bits. With this recovery method, the CRT-RSA secret keys are partially recovered, and the CRT-RSA secret keys are recovered entirely using Heninger-Shacham's method. Specifically, once the CRT-RSA secret keys are partially recovered, the branch-and-bound strategy is given by repeating the following:

1. Calculate new bits using Eqs. (3.1)–(3.3)
2. Discard leaves if the calculated bits differ from the partially recovered bits in the CRT-RSA secret keys

The secret keys are searched from the final candidates at the $(n/2 - 1)$-th depth. We now focus on how to recover bits of the secret exponent from a square-and-multiply sequence.

Original Method for Recovering Bits of a Secret Exponent [10]
Modular exponentiations are calculated by repeating the squaring (**S**) and multiplication (**M**) applied by the sliding window method. With the CRT-RSA scheme, square-and-multiply sequences of two modular exponentiations, $x^{d_p} \bmod p$ and $x^{d_q} \bmod q$, are extracted using side-channel attacks [10]. However, because there are many candidates of the multiplier in each **M**, the CRT-RSA secret keys d_p and d_q cannot be immediately determined from square-and-multiply sequences.

As noted above, not all multipliers can be recovered immediately, and bits of a secret exponent are partially recovered. The method for recovering bits of the secret exponent comprises the four rules, i.e., Rules 0–3. We describe these rules using an example for $w = 4$, as follows.

SSSMSSSSSSMSSSMSSSSSMSMSSSSSSMSSSSSSM

Before applying Rules 0–3, **SM**s are converted into $\underline{\mathbf{x}}$, and the remaining **S**s are converted into **x**. This **x** and $\underline{\mathbf{x}}$ sequence is a bit sequence with a value of **0** or **1**. By applying this conversion, our example is converted into the following.

xx$\underline{\mathbf{x}}$xxxxxx$\underline{\mathbf{x}}$xxx$\underline{\mathbf{x}}$xxxxx$\underline{\mathbf{xx}}$xxxxxx$\underline{\mathbf{x}}$xxxxxx$\underline{\mathbf{x}}$

The following Rules 0–3 are applied to the given sequence:

- **Rule 0**: $\underline{\mathbf{x}} \rightarrow \underline{\mathbf{1}}$.
- **Rule 1**: $\underline{\mathbf{1}}\mathbf{x}^i\underline{\mathbf{1}}\mathbf{x}^{w-1-i} \rightarrow \underline{\mathbf{1}}\mathbf{x}^i\underline{\mathbf{1}}\mathbf{0}^{w-1-i}$ for $0 \leq i \leq w-2$.
- **Rule 2**: $\mathbf{x}^{w-1}\underline{\mathbf{11}} \rightarrow \mathbf{1}\mathbf{x}^{w-2}\underline{\mathbf{11}}$.
- **Rule 3**: $\underline{\mathbf{1}}\mathbf{x}^i\mathbf{x}^{w-1}\underline{\mathbf{1}} \rightarrow \underline{\mathbf{1}}\mathbf{0}^i\mathbf{x}^{w-1}\underline{\mathbf{1}}$ for $i > 0$.

Their method recovers multiplication bits with Rule 0, trailing zeros with Rule 1, leading ones with Rule 2, and leading zeros with Rule 3. Note that more bits are recovered by applying a more extended situation with Rule 1, and this extended Rule 1 is adopted in the previous method. In the extended Rule 1, $\underline{\mathbf{1}}$ is searched from the MSB to the LSB. In the highest-order $\underline{\mathbf{1}}$, if $\underline{\mathbf{1}}$ satisfies $\mathbf{x}^i\underline{\mathbf{1}}\mathbf{x}^{w-1-i}$ for $0 \leq i \leq w-2$, **x** is recovered after $\underline{\mathbf{1}}$ as **0**, namely,

$$\mathbf{x}^i\underline{\mathbf{1}}\mathbf{x}^{w-1-i} \rightarrow \mathbf{x}^i\underline{\mathbf{1}}\mathbf{0}^{w-1-i}.$$

In other $\underline{\mathbf{1}}$s, if $\underline{\mathbf{1}}$ satisfies (**0** or $\underline{\mathbf{1}}$)$\mathbf{x}^i\underline{\mathbf{1}}\mathbf{x}^{w-1-i}$ for $0 \leq i \leq w-2$, **x** is recovered after the latter $\underline{\mathbf{1}}$ as **0**, namely,

$$(\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{x}^i\underline{\mathbf{1}}\mathbf{x}^{w-1-i} \rightarrow (\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{x}^i\underline{\mathbf{1}}\mathbf{0}^{w-1-i}.$$

By applying Rules 0–3 in this order, the recovery bits are given as follows:

Applying Rule 0: xx$\underline{\mathbf{1}}$xxxxx$\underline{\mathbf{1}}$xx$\underline{\mathbf{1}}$xxxx$\underline{\mathbf{11}}$xxxxx$\underline{\mathbf{1}}$xxxxx$\underline{\mathbf{1}}$.

Applying Rule 1: xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$xx$\underline{\mathbf{1}}$0xxx$\underline{\mathbf{11}}$000xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$.

Applying Rule 2: xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$xx$\underline{\mathbf{1}}$01xx$\underline{\mathbf{11}}$000xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$.

Applying Rule 3: xx$\underline{\mathbf{1}}$00xxx$\underline{\mathbf{1}}$xx$\underline{\mathbf{1}}$01xx$\underline{\mathbf{11}}$000xx$\underline{\mathbf{1}}$00xxx$\underline{\mathbf{1}}$.

Optimal Method for Recovering Bits of a Secret Exponent [115]
In the original method, more bits are recovered by repeating the rules. However, additional recovery is not considered in this study because of a rare occurrence [10]. Van Vredendaal [115] tackled this problem more rigorously and proposed a new method for recovering bits of a secret exponent. We now describe recovering bits of a secret exponent

using a toy example. We consider **SSMSSM** in $w = 2$. Bit sequences **0101**, **1101**, and **1111** do not contradict with **SSMSSM**. Bits are the same at the second and fourth; thus, we can recover as **x1x1**. In this sense, Van Vredendaal's recovery method for bits of the secret exponent has optimality.

We now describe Van Vredendaal's method. First, bits of the secret exponent d are indexed as $d_{n-1}d_{n-2}\dots d_0$. The set of indexes of the multiplication bits is defined as $M = \{k_0, k_1, \dots, k_l\}$ with $k_0 > k_1 > \cdots > k_l$. In each multiplication bit, k_j, a multiplier width m_{k_j} is defined as the number of bits used in determining the multiplier. For example, if the multiplier is $5 = \mathbf{101}_2$, the multiplier width is 3. Next, we calculate $m^+_{k_j} := \max m_{k_j}$ in each window from the MSB sides using a greedy algorithm determining each window as near as MSB sides, which determined each window as being as near as the MSB sides. Similarly, we calculate $m^-_{k_j} := \min m_{k_j}$ in each window from LSB sides. For example,

$$\mathbf{xx\underline{1}xxxxx\underline{1}xx\underline{1}xxxx\underline{11}xxxxx\underline{1}xxxxx\underline{1}}.$$

In addition, $m^+_{k_j}$ is calculated by dividing the following:

$$\mathbf{xx\underline{1}x\ x\ xxx\underline{1}\ xx\underline{1}x\ xxx\underline{1}\ \underline{1}xxx\ xx\underline{1}x\ x\ xxx\underline{1}},$$

and $m^+_{k_0} = 3, m^+_{k_1} = 4, m^+_{k_2} = 3, m^+_{k_3} = 4, m^+_{k_4} = 1, m^+_{k_5} = 3, m^+_{k_6} = 4$. Similarly, $m^-_{k_j}$ is calculated by dividing the following:

$$\mathbf{x\ x\underline{1}xx\ xxx\underline{1}\ xx\underline{1}x\ xxx\underline{1}\ \underline{1}xxx\ xx\ \underline{1}xxx\ xx\ \underline{1}},$$

and $m^-_{k_0} = 2, m^-_{k_1} = 4, m^-_{k_2} = 3, m^-_{k_3} = 4, m^-_{k_4} = 1, m^-_{k_5} = 1, m^-_{k_6} = 1$.

After calculating $m^+_{k_j}$ and $m^-_{k_j}$, bits of the secret exponent are recovered as follows:

$$d'_i = \begin{cases} \mathbf{1} & \text{if } i \in M \\ \mathbf{1} & \text{else if } k_j + m^-_{k_j} - 1 = i = k_j + m^+_{k_j} - 1 \text{ for some } k_j \in M \\ \mathbf{x} & \text{else if } k_j + m^-_{k_j} - 1 \le i \le k_j + m^+_{k_j} - 1 \text{ for some } k_j \in M \\ \mathbf{0} & \text{otherwise} \end{cases}$$

and recovered bits of the secret exponent by Van Vredendaal's method are given as

$$\mathbf{xx\underline{1}001xx\underline{1}1x\underline{1}01xx\underline{11}000xx\underline{1}00xxx\underline{1}}.$$

The second **1** corresponds to Rule 2 and last **0** corresponds to Rules 1 and 3.

Optimal Recovery Rules for the Secret Exponent

By considering Van Vredendaal's method in more detail, the same recovery is realized using a small modification of Rules 0-3. In particular, some bits of the secret exponent can be recovered when $m^+_{k_j}$ and $m^-_{k_j}$ are calculated. First, the calculation of $m^+_{k_j}$ corresponds to Rule 1. Second, the calculation of $m^-_{k_j}$ corresponds to Rule 2. After calculating $m^+_{k_j}$ and $m^-_{k_j}$, **0**s are recovered. Therefore, Van Vredendaal's method is realized through a small modification of Rules 0-3.

Now, Van Vredendaal's recovery methods are rewritten into Rules 0-3. Rules 0 and 1 are the same as in [10]. In Rule 2, $\underline{\mathbf{1}}$ is searched from the LSB to the MSB. If $\underline{\mathbf{1}}$ satisfies $\mathbf{x}^{w-1-i}\underline{\mathbf{1}}\mathbf{0}^i(\mathbf{1} \text{ or } \underline{\mathbf{1}})$ for $0 \le i \le w-2$,

$$\mathbf{x}^{w-1-i}\underline{\mathbf{1}}\mathbf{0}^i(\mathbf{1} \text{ or } \underline{\mathbf{1}}) \to \mathbf{1}\mathbf{x}^{w-2-i}\underline{\mathbf{1}}\mathbf{0}^i(\mathbf{1} \text{ or } \underline{\mathbf{1}}).$$

Note that Rule 2 in [10] only dealt with $i = 0$ because another event rarely occurred. Because Rule 2 is modified, we modify Rule 3 corresponding to Rule 2. In Rule 3, $\underline{\mathbf{1}}$ is searched from the MSB to the LSB. In the highest-order $\underline{\mathbf{1}}$, if $\underline{\mathbf{1}}$ satisfies $\mathbf{x}^i\mathbf{1}\mathbf{x}^k\underline{\mathbf{1}}$ or

$\mathbf{x}^i\mathbf{x}^{w-1}\underline{\mathbf{1}}$ for $i \geq 0$,

$$\mathbf{x}^i\mathbf{1}\mathbf{x}^k\underline{\mathbf{1}} \text{ or } \mathbf{x}^i\mathbf{x}^{w-1}\underline{\mathbf{1}} \rightarrow \mathbf{0}^i\mathbf{1}\mathbf{x}^k\underline{\mathbf{1}} \text{ or } \mathbf{0}^i\mathbf{x}^{w-1}\underline{\mathbf{1}}.$$

In other $\underline{\mathbf{1}}$s, if $\underline{\mathbf{1}}$ satisfies $(\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{x}^j\mathbf{1}\mathbf{x}^k\underline{\mathbf{1}}$ or $(\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{x}^j\mathbf{x}^{w-1}\underline{\mathbf{1}}$,

$$(\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{x}^j\mathbf{1}\mathbf{x}^k\underline{\mathbf{1}} \rightarrow (\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{0}^j\mathbf{1}\mathbf{x}^k\underline{\mathbf{1}},$$

$$(\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{x}^j\mathbf{x}^{w-1}\underline{\mathbf{1}} \rightarrow (\mathbf{0} \text{ or } \underline{\mathbf{1}})\mathbf{0}^j\mathbf{x}^{w-1}\underline{\mathbf{1}}.$$

Note that there are no overlaps in the recovered bits in each rule in the optimal bit recovery rules, whereas there are overlaps in Rules 1 and 3 in the original rules. By applying modified Rules 2 and 3, our example is recovered as follows:

Applying Rule 1: xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$xx$\underline{\mathbf{1}}$0xxx$\underline{\mathbf{11}}$000xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$.

Applying Modified Rule 2: xx$\underline{\mathbf{1}}$0x1xx$\underline{\mathbf{1}}$1x$\underline{\mathbf{1}}$01xx$\underline{\mathbf{11}}$000xx$\underline{\mathbf{1}}$0xxxx$\underline{\mathbf{1}}$.

Applying Modified Rule 3: xx$\underline{\mathbf{1}}$001xx$\underline{\mathbf{1}}$1x$\underline{\mathbf{1}}$01xx$\underline{\mathbf{11}}$000xx$\underline{\mathbf{1}}$00xxx$\underline{\mathbf{1}}$.

The result of the recovery is the same as that of Van Vredendaal's method. We call these rules the optimal recovery rules.

3.2.3 Recovery Method for the CRT-RSA Secret Keys from Square-and-Multiply Sequences Directly [10]

This subsection describes the previous method for recovering the CRT-RSA secret keys from square-and-multiply sequences directly. With this method, the following are repeated:

- Generate some leaves by Heninger-Shacham's method
- Convert recovered bits into square-and-multiply sequences in each leaf
- Compare with the given square-and-multiply sequence

However, incorrect square-and-multiply sequences may be obtained during the conversion. The sliding window method calculates exponentiations from the MSB side and summarizes w bits. However, the secret keys are recovered from the LSB side, and different w bits are used for the conversion into square-and-multiply sequences. Thus, we must clarify the position of bits correctly converted into square-and-multiply sequences.

To realize a correct comparison, we focus on **S**. Initially, we search **S**s that are next to an **M** or at the beginning of w **S**s in the square-and-multiply sequences from the final operation. If such **S**s are found, we count the number of **S**s from the final operation and calculate the leaves until reaching the same depth. We then convert the calculated bits into a square-and-multiply sequence. If there are mismatches between the given square-and-multiply sequences and the calculated square-and-multiply sequences, we discard the leaf. By doing this, the CRT-RSA secret keys are recovered from the correct square-and-multiply sequences. In theory, the CRT-RSA secret keys are recovered in polynomial time in n when the window size w is less than or equal to 4.

3.3 Recovery Rate of Bits of CRT-RSA Secret Keys from Square-and-Multiply Sequences

This section clarifies how many bits of the secret exponent are recovered by Theorem 1, and we verify this theoretical analysis by the numerical experiments. Section 3.3.1 gives the theoretical analysis. Section 3.3.2 and 3.3.3 give numerical experiments on random

exponents and CRT-RSA secret keys, respectively.

3.3.1 Exact Rate of Recovered Bits of the Secret Exponent from Square-and-Multiply Leakage

We provide the exact rate of recovered bits of the secret exponent only from a square-and-multiply sequence, which is given by Theorem 1.

Theorem 1 Suppose that a secret exponent is generated randomly. If $w \geq 2$, the average rate of recovered bits of the secret exponent is given by

$$\frac{2}{w+1} + \frac{\sum_{k=0}^{w-2} f_w(k)g(k)}{2(w+1)} + \frac{2^w - 1}{2^{w-1}\left(2^{w-1}+1\right)} \frac{1}{3(w+1)}$$

where

$$f_w(k) = \frac{2}{3 \cdot 2^k}\left(1 - \frac{1}{2^{w-k}}\right)\left(1 - \frac{2}{2^{w-k}}\right), g(k) = 2\left(1 - \frac{2^k}{2^{k+2}-1}\right)\prod_{j=1}^{k} \frac{2^{j-1}}{2^{j+1}-1}.$$

To prove Theorem 1, we analyze the optimal recovery rules for bits of a secret exponent. The first term $2/(w+1)$ corresponds to Rule 0 and Rule 3, the second term corresponds to Rule 1, and the third term corresponds to Rule 2.

To prove Theorem 1, we use the renewal reward theorem [105]. This theorem is given as Theorem 2 with notation in [10].

Theorem 2 (Renewal Reward Theorem [105]) We are given i.i.d. random values (X_i, Y_i) where $i \in \mathbb{N}$. Moreover, we define $S_n = \sum_{i=1}^{n} X_i$, $N_t = \sum_{n=1}^{\infty} \mathbf{1}\left(S_n \leq t\right)$, and $R_t = \sum_{i=1}^{N_t} Y_i$, where $n \in \mathbb{N}$ and t is a positive real number. If $E\left[X_1\right] < \infty$, $E\left[Y_1\right] < \infty$,

$$\lim_{t\to\infty} \frac{R_t}{t} = \frac{E\left[Y_1\right]}{E\left[X_1\right]}.$$

In the renewal reward theorem, the time that satisfies some conditions is defined as a renewal. In Theorem 2, X_i is regarded as the inter-arrival time, S_n as the arrival time of n-th elements, and N_t as the number of arrivals in time t. Renewal occurs in each X_i. Moreover, the reward is defined in each renewal. In Theorem 2, Y_i is regarded as the reward in each inter-arrival time X_i. Then, R_t is regarded as the reward in time t. We now regard the length of the secret exponent as t and that of the recovering bits as reward R_t. It was attempted to calculate the recovery rate of bits of the secret exponent by using the renewal reward theorem [105] under the same settings [10]. However, they only calculate the upper bound and the lower bound of recovering bits of the secret exponent. Thus, they failed to calculate the exact recovery rate of bits of the secret exponent.

We define i.i.d. (X_i, Y_i) in the analysis of each Rule j as follows:

- X_i: The length of bit sequences until the designated bit pattern in Rule j occurs
- Y_i: The number of recovered bits in X_i

To define (X_i, Y_i) as i.i.d., we determine the definition of X_i in each rule. Specifically, we define (X_i, Y_i) based on the multiple windows, while the original research [10] only considered one window. We now explain (X_i, Y_i) in each Rule 0–3.

First, we explain (X_i, Y_i) in Rule 0. In Rule 0, a multiplication bit is recovered in each window. Thus, the number of recovered bits is independent between each window. Therefore, we define X_i as the number of bits until we hit a window.

Next, we explain (X_i, Y_i) in Rule 1. In Rule 1, trailing zeros are recovered by the greedy algorithm calculating $m^+_{k_j}$. If we focus on one window, the difference between $m^+_{k_j}$ and the actual m_{k_j} affects the value of $m^+_{k_{j+1}}$. Thus, there is dependency on Y_j and Y_{j+1} except when $m^+_{k_j} = m_{k_j}$. We now recall the definition of $m^+_{k_j} := \max m_{k_j}$. When we hit the window whose $m_{k_j} = w$, then $m^+_{k_j} = m_{k_j}$, and Y_j and Y_{j+1} have no dependency. Thus, we define X_i as the number of bits until we hit window $\mathbf{1x} \ldots \mathbf{x\underline{1}}$, whose m_{k_j} is w.

The same X_i as Rule 1 is used in Rule 3, recovering leading zeros. This is because the recovery bits are determined based on $m^+_{k_j}$, and the difference between $m^+_{k_j}$ and the actual m_{k_j} propagates to the value of $m^+_{k_{j+1}}$. Thus, there is dependency on Y_j and Y_{j+1} except when $m^+_{k_j} = m_{k_j}$. When $m_{k_j} = w$, then $m^+_{k_j} = m_{k_j}$, and we do not need to consider the dependency with the next window. Therefore, we use the same X_i in Rule 3 as Rule 1.

Finally, we explain (X_i, Y_i) in Rule 2. In Rule 2, leading one is recovered when $m^+_{k_j} = m^-_{k_j}$ during the calculation of $m^-_{k_j}$. Thus, if $m^+_{k_{j+1}} \neq m^-_{k_{j+1}}$ is assured, there is no dependency between Y_j and Y_{j+1}. Especially, if $m^+_{k_{j+1}} \neq m_{k_{j+1}}$ is assured, there is no dependency between Y_j and Y_{j+1}. Therefore, we define X_i as the number of bits until we hit the window that moves certainly.

We now give the proof of Theorem 1. In this proof, we give the exact recovery rate of the secret exponent in each rule. The recovery rate of the secret exponent on Theorem 1 is given by summing up the following average recovery rate of the secret exponent.

Bit Recovery Rate in Rule 0

We prove the following Lemma 1.

Lemma 1 Suppose that a secret exponent is generated randomly. The average recovery rate of the secret exponent obtained by Rule 0 is given by $1/(w+1)$.

Before we prove Lemma 1, we give an intuition of our proof. In Rule 0, we recover a multiplication bit in each window, and thus, each window is independent. Therefore, renewal is defined as the hitting window. We now provide rigorous proof.

First, we define X_i and then calculate $E[X_i]$. In this proof, we write the windows as $\mathbf{W}$, which is an abbreviation of **W**indow. We read bits from the MSB side, and define X_i as the length of the bit sequence until hitting $\mathbf{W}$, namely, the pattern $\mathbf{0} \ldots \mathbf{0W}$, because $\mathbf{W}$ occurs when we hit $\mathbf{1}$. When the length of $\mathbf{0}$s is r $(r \geq 0)$, $X_i = r + w$. Now, when we read bits from MSB side, $\Pr[\mathbf{0}] = 1/2$ and $\Pr[\mathbf{W}] = 1/2$. Thus,

$$\Pr[X_i = r + w] = \left(\frac{1}{2}\right)^r \frac{1}{2}.$$

Therefore,

$$E[X_i] = \sum_{r=0}^{\infty} (r + w) \left(\frac{1}{2}\right)^r \frac{1}{2} = w + 1.$$

Next, we calculate $E[Y_i]$. The number of multiplication bits is always 1, and thus, Y_i is always 1. Therefore, $E[Y_i] = 1$.

In conclusion, the average recovery rate of the secret exponent in Rule 0 is

$$\frac{E[Y_i]}{E[X_i]} = \frac{1}{w+1}.$$

This proves Lemma 1. □

Bit Recovery Rate in Rule 1

We prove the following Lemma 2.

Lemma 2 Suppose that a secret exponent is generated randomly. If $w \geq 2$, the average recovery rate of the secret exponent according to Rule 1 is given by

$$\frac{\sum_{k=0}^{w-2} f_w(k)g(k)}{2(w+1)}.$$

Before we prove Lemma 2, we provide an intuition of our proof. In Rule 1, we consider the highest-order candidate in each window, and when there are low-order bits compared to $\underline{\mathbf{1}}$, we recover them as **0**s. Then, the actual candidate is moved to the highest-order candidate. When each window is moved to the highest-order candidate, there is a dependency on each window. However, the window **1x** ... **x**$\underline{\mathbf{1}}$ does not move because this is already the highest-order candidate. Thus, we set renewal as the hitting window **1x** ... **x**$\underline{\mathbf{1}}$ and calculate the recovery rate. We now provide rigorous proof.

First, we define X_i and then calculate $E[X_i]$. In this proof, we write the window **1x** ... **x**$\underline{\mathbf{1}}$ as **F**, which is an abbreviation of **F**ixed window. Moreover, a window except for **1x** ... **x**$\underline{\mathbf{1}}$ as **L**, which is an abbreviation of **L**ifting window to the MSB side. We read bits from the MSB side, and define X_i as the length of the bit sequence until hitting **F**. Then, we read the followings in this order until hitting **F**:

1. **0** ... **0L** r times ($r \geq 0$)
2. **0** ... **0F**

When we read bits from MSB side, $\Pr[\mathbf{0}] = 1/2$, $\Pr[\mathbf{F}] = 1/4$, and $\Pr[\mathbf{L}] = 1/4$. We now calculate $E[X_i]$ by considering the length and corresponding probability of this bit sequence.

First, we consider $r = 0$. Then, the bit sequence is given by **0** ... **0F**. When the length of **0**s in **0** ... **0F** is z ($z \geq 0$), $X_i = z + w$, and the probability is $1/2^{z+2}$.

Second, we consider the case of $r > 0$. We define the length of **0**s in **0** ... **0F** as z ($z \geq 0$) and that of **0**s in each **0** ... **0L** as z_i ($z_i \geq 0$). Then, X_i is given by

$$X_i = w(r+1) + z + \sum_{i=1}^{r} z_i$$

and the probability is

$$\left(\frac{1}{4}\right)^{r+1} \left(\frac{1}{2}\right)^{z} \prod_{i=1}^{r} \left(\frac{1}{2}\right)^{z_i}.$$

From this, we calculate the expectation of X_i as $E[X_i] = 2w + 2$.

Next, we calculate $E[Y_i]$. When we move each window to the highest-order candidate, the amount of movement is the minimum of the following two amounts:

- Lower-order bits of $\underline{\mathbf{1}}$ in the window.
- (The amount of movement in the neighboring higher-order window)+(The number of **0**s between the windows).

In the first one, the amount of movement is less than $w - 1$ because the window's size is w. In the second one, the upper bound of the movement restricted by the higher-order window is calculated. Based on these, we calculate $E[Y_i]$.

We now consider the situation where we read the followings in this order until hitting **F**:

1. **0** ... **0L** r times ($r \geq 0$)
2. **0** ... **0F**

When $r = 0$, the number of recovery bits is 0. We now consider the case of $r > 0$. In the final **0** ... **0F**, the bit is not recovered. Thus, we consider recovering bits in the windows **L**. When we focus on a window **L**, we define $p_{a,b,l}$ as the probability of followings:

- The neighbor higher-order window moves a.
- The current window moves b.
- l bits are recovered in the current window by Rule 1.

Note that this value does not depend on the position of the window. The current window satisfies $0 \leq b \leq w-1$ and $0 \leq l \leq w-1$, and these include all patterns. Therefore,

$$\sum_{b=0}^{w-1} \sum_{l=0}^{w-1} p_{a,b,l} = \frac{1}{2}. \tag{3.4}$$

Based on these, when l_i bits are recovered in each **0** ... **0L**, the number of recovered bits is $\sum_{i=1}^{r} l_i$. Then, the probability of this bit sequence is $\frac{1}{2}\left(\prod_{j=0}^{r-1} p_{a_j,a_{j+1},l_{j+1}}\right)$ with $a_0 = 0$. Therefore,

$$\begin{aligned} E[Y_i] &= \frac{1}{2}\sum_{r=1}^{\infty} \sum_{a_1,\ldots,a_r=0}^{w-1} \sum_{l_1,\ldots,l_r=0}^{w-1} \left(\sum_{i=1}^{r} l_i\right)\left(\prod_{i=0}^{r-1} p_{a_j,a_{j+1},l_{j+1}}\right) \\ &= \frac{1}{2}\sum_{r=1}^{\infty} \sum_{a_1,\ldots,a_r=0}^{w-1} \sum_{l_1,\ldots,l_r=0}^{w-1} \sum_{i=1}^{r} \left(l_i \prod_{i=0}^{r-1} p_{a_j,a_{j+1},l_{j+1}}\right) \end{aligned} \tag{3.5}$$

In Eq. (3.5), the coefficient $1/2$ is the probability of occurrence of the final **0** ... **0F** and the remaining indicates that the pattern of **0** ... **0L** occurs r times. Now, in each r,

$$\begin{aligned} &\sum_{a_1,\ldots,a_r=0}^{w-1} \sum_{l_1,\ldots,l_r=0}^{w-1} \sum_{i=1}^{r} \left(l_i \prod_{j=0}^{r-1} p_{a_j,a_{j+1},l_{j+1}}\right) \\ &\leq \sum_{a_1,\ldots,a_r=0}^{w-1} \sum_{l_1,\ldots,l_r=0}^{w-1} \sum_{i=1}^{r} \left((w-1) \prod_{j=0}^{r-1} p_{a_j,a_{j+1},l_{j+1}}\right) \\ &= (w-1)r \sum_{a_1,\ldots,a_r=0}^{w-1} \sum_{l_1,\ldots,l_r=0}^{w-1} \prod_{j=0}^{r-1} p_{a_j,a_{j+1},l_{j+1}} \\ &= (w-1)r \prod_{j=0}^{r-1} \left(\sum_{a_{j+1}=0}^{w-1} \sum_{l_{j+1}=0}^{w-1} p_{a_j,a_{j+1},l_{j+1}}\right) \\ &= (w-1)r \left(\frac{1}{2}\right)^r \text{ (From the Eq. (3.4))}, \end{aligned}$$

thus,

$$E[Y_i] \leq \frac{1}{2}\sum_{r=1}^{\infty} \left((w-1)r\left(\frac{1}{2}\right)^r\right) \leq w-1.$$

Therefore, $E[Y_i]$ is absolute convergence. Therefore, we swap the item in Eq. (3.5) as

$$\begin{aligned}
E[Y_i] &= \frac{1}{2}\sum_{r=1}^{\infty}\sum_{i=1}^{r}\sum_{a_1,\ldots,a_r=0}^{w-1}\sum_{l_1,\ldots,l_r=0}^{w-1}\left(l_i\prod_{j=0}^{r-1}p_{a_j,a_{j+1},l_{j+1}}\right)\\
&= \frac{1}{2}\sum_{r=1}^{\infty}\sum_{i=1}^{r}\sum_{a_1,\ldots,a_i=0}^{w-1}\sum_{l_1,\ldots,l_i=0}^{w-1}\left(l_i\left(\prod_{j=i}^{r-1}\sum_{a_{j+1}=0}^{w-1}\sum_{l_{j+1}=0}^{w-1}p_{a_j,a_{j+1},l_{j+1}}\right)\prod_{j=0}^{i-1}p_{a_j,a_{j+1},l_{j+1}}\right)\\
&= \frac{1}{2}\sum_{r=1}^{\infty}\sum_{i=1}^{r}\sum_{a_1,\ldots,a_i=0}^{w-1}\sum_{l_1,\ldots,l_i=0}^{w-1}\left(l_i\left(\frac{1}{2}\right)^{r-i}\prod_{j=0}^{i-1}p_{a_j,a_{j+1},l_{j+1}}\right)\\
&= \frac{1}{2}\sum_{i=1}^{\infty}\sum_{r=i}^{\infty}\sum_{a_1,\ldots,a_i=0}^{w-1}\sum_{l_1,\ldots,l_i=0}^{w-1}\left(l_i\left(\frac{1}{2}\right)^{r-i}\prod_{j=0}^{i-1}p_{a_j,a_{j+1},l_{j+1}}\right)\\
&= \frac{1}{2}\sum_{i=1}^{\infty}\left(\sum_{r=i}^{\infty}\left(\frac{1}{2}\right)^{r-i}\right)\sum_{a_1,\ldots,a_i=0}^{w-1}\sum_{l_1,\ldots,l_i=0}^{w-1}\left(l_i\prod_{j=0}^{i-1}p_{a_j,a_{j+1},l_{j+1}}\right)\\
&= \sum_{i=1}^{\infty}\sum_{a_1,\ldots,a_i=0}^{w-1}\sum_{l_1,\ldots,l_i=0}^{w-1}\left(l_i\prod_{j=0}^{i-1}p_{a_j,a_{j+1},l_{j+1}}\right). \qquad (3.6)
\end{aligned}$$

We now rewrite Eq. (3.6) as the product of matrices. We define $p_{a,b}$ as $p_{a,b}=\sum_{l=0}^{w-1}p_{a,b,l}$. Then, we can regard $p_{a,b}$ as the probability of followings:

- The neighboring higher-order window moves a.
- The current window moves b.

This depends on only a and b. We now define A as the matrix in w dimensions whose $(a+1,b+1)$ element is $p_{a,b}$. Moreover, we define $Z_{a_{i-1}}$ as $\sum_{a_i=0}^{w-1}\sum_{l_i=0}^{w-1}l_ip_{a_{i-1},a_i,l_i}$. Then,

$$\sum_{a_1,\ldots,a_i=0}^{w-1}\sum_{l_1,\ldots,l_i=0}^{w-1}\left(l_i\prod_{j=0}^{i-1}p_{a_j,a_{j+1},l_{j+1}}\right)=\begin{bmatrix}1 & 0 & \ldots & 0\end{bmatrix}A^{i-1}\begin{bmatrix}Z_0\\Z_1\\\vdots\\Z_{w-1}\end{bmatrix}.$$

Therefore,

$$E[Y_i]=\begin{bmatrix}1 & 0 & \ldots & 0\end{bmatrix}\left(\sum_{i=0}^{\infty}A^i\right)\begin{bmatrix}Z_0\\Z_1\\\vdots\\Z_{w-1}\end{bmatrix} \qquad (3.7)$$

We now obtain the exact form of matrix A by calculating $p_{a,b}$. We focus on the value of b, which is the minimum of the following two amounts:

- Lower-order bits of $\underline{\mathbf{1}}$ in window
- (The amount of movement in the neighboring higher-order window)+(The number of **0**s between windows)

First, we consider $a = 0$. $b = 0$ is satisfied when the current window follows the neighboring higher-order window continuously. Thus, $p_{0,0} = 1/4$.

Second, $1 \leq b \leq w-2$ is satisfied when one of the following is satisfied:

- The number of **0**s between the current and neighboring higher-order windows is b and that of the lower-order bits of $\underline{\mathbf{1}}$ is more than b in the current window.
- The number of **0**s between the current and neighboring higher-order windows is more than b and that of the lower-order bits of $\underline{\mathbf{1}}$ is b in the current window.

Then, the probability is given as

$$p_{0,b} = \left(\frac{1}{2}\right)^b \frac{1}{2^{b+1}} + \left(\frac{1}{2}\right)^b \frac{1}{2^{b+2}}.$$

Finally, $b = w-1$ is satisfied when the number of **0**s between the current and the MSB window is more than $w-1$ and the current window is $\underline{\mathbf{1}}\mathbf{0}\ldots\mathbf{0}$; therefore,

$$p_{0,w-1} = \frac{1}{2^{w-2}} \frac{1}{2^w}.$$

Next, we consider $a > 0$. First, $p_{a,0} = 0$. Second, $1 \leq b \leq a-1$ is satisfied when the lower-order bit of $\underline{\mathbf{1}}$ is b in the current window. Therefore, $p_{a,b} = 2/2^{b+2}$. Third, $a \leq b \leq w-2$ is satisfied when one of the following is satisfied:

- The number of **0**s between the current and the neighboring higher-order window is $b-a$ and that of the lower-order bits of $\underline{\mathbf{1}}$ is more than b in the current window.
- The number of **0**s between the current and neighboring higher-order windows is more than $b-a$ and the number of lower-order bits of $\underline{\mathbf{1}}$ is b in the current window.

Therefore,

$$p_{a,b} = \left(\frac{1}{2}\right)^{b-a} \frac{1}{2^{b+1}} + \left(\frac{1}{2}\right)^{b-a} \frac{1}{2^{b+2}}.$$

Finally, $b = w-1$ is satisfied when the number of **0**s between the current and the neighboring higher-order window is more than $w-1-a$ and the current window is $\underline{\mathbf{1}}\mathbf{0}$ $\ldots$ **0**; therefore,

$$p_{a,w-1} = \frac{1}{2^{w-2-a}} \frac{1}{2^w}.$$

Therefore, for $w \geq 2$,

$$A = \begin{bmatrix} \frac{1}{4} & \frac{1}{2}\frac{1}{4} + \frac{1}{2}\frac{1}{8} & \frac{1}{4}\frac{1}{8} + \frac{1}{4}\frac{1}{16} & \cdots & \frac{1}{2^{w-2}}\frac{1}{2^{w-1}} + \frac{1}{2^{w-2}}\frac{1}{2^w} & \frac{1}{2^{w-2}}\frac{1}{2^w} \\ 0 & \frac{1}{4} + \frac{1}{8} & \frac{1}{2}\frac{1}{8} + \frac{1}{2}\frac{1}{16} & \cdots & \frac{1}{2^{w-3}}\frac{1}{2^{w-1}} + \frac{1}{2^{w-3}}\frac{1}{2^w} & \frac{1}{2^{w-3}}\frac{1}{2^w} \\ 0 & \frac{2}{8} & \frac{1}{8} + \frac{1}{16} & \cdots & \frac{1}{2^{w-4}}\frac{1}{2^{w-1}} + \frac{1}{2^{w-4}}\frac{1}{2^w} & \frac{1}{2^{w-4}}\frac{1}{2^w} \\ & & & \vdots & & \\ 0 & \frac{2}{8} & \frac{2}{16} & \cdots & \frac{2}{2^w} & \frac{2}{2^w} \end{bmatrix}.$$

Now, the sum of rows in A is less than $1/2$ because of (3.4). Thus, $||A||_\infty = 1/2$. Therefore,

$\sum_{i=0}^{\infty} A^i = (I-A)^{-1}$. Hence, from Eq. (3.7),

$$E[Y_i] = \begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}(I-A)^{-1}\begin{bmatrix}Z_0\\Z_1\\\vdots\\Z_{w-1}\end{bmatrix}. \tag{3.8}$$

From Eq. (3.8), we calculate $E[Y_i]$ by $\begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}(I-A)^{-1}$ and Z_i. Thus, we calculate these values.

We now calculate

$$\begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}(I-A)^{-1},$$

namely the first row of $(I-A)^{-1}$. This is given by

$$\begin{bmatrix}g(0) & \dots & g(w-2) & 2\prod_{j=1}^{w-1}\frac{2^{j-1}}{2^{j+1}-1}\end{bmatrix}, \tag{3.9}$$

and we proof this.

First, we prove that the first row of $(I-A)^{-1}$ is given by Eq. (3.9) when $w=2$. The matrix A is given as

$$A = \begin{bmatrix}\frac{1}{4} & \frac{1}{4}\\ 0 & \frac{2}{4}\end{bmatrix}.$$

Then,

$$(I-A)^{-1} = \begin{bmatrix}\frac{4}{3} & \frac{2}{3}\\ 0 & 2\end{bmatrix},$$

and the first row of $(I-A)^{-1}$ is $\begin{bmatrix}g(0) & 2\prod_{j=1}^{2-1}\frac{2^{j-1}}{2^{j+1}-1}\end{bmatrix}$.

Next, we prove that the first row of $(I-A)^{-1}$ is given by Eq. (3.9) when $w \geq 3$. $I-A$ is given as

$$I-A = \begin{bmatrix}\frac{3}{4} & -\frac{1}{2}\frac{1}{4}-\frac{1}{2}\frac{1}{8} & \dots & -\frac{1}{2^{w-2}}\frac{1}{2^w}\\ 0 & 1-\frac{1}{4}-\frac{1}{8} & \dots & -\frac{1}{2^{w-3}}\frac{1}{2^w}\\ & & \vdots & \\ 0 & -\frac{2}{8} & \dots & 1-\frac{2}{2^w}\end{bmatrix}.$$

We now calculate the inner product of $\mathbf{g} = \begin{bmatrix}g(0) & 2\prod_{j=1}^{w-1}\frac{2^{j-1}}{2^{j+1}-1}\end{bmatrix}$ and each column of $I-A$. Then, we prove followings:

- The inner product of $\mathbf{g}$ and the first column of $I-A$ is 1.
- Otherwise, 0.

From $g(0) = 4/3$, the inner product of $\mathbf{g}$ and the first column of $I - A$ is 1.

Next, we focus on the second to $(w-1)$-th column in $I - A$. We now focus on the i-th column. When we focus on each element of i-th column, the following is satisfied:

- The k-th $(1 \le k \le i-1)$ elements are $-\left(\frac{1}{2}\right)^{i-k}\frac{1}{2^i} - \left(\frac{1}{2}\right)^{i-k}\frac{1}{2^{i+1}}$.
- The i-th element is $1 - \frac{1}{2^i} - \frac{1}{2^{i+1}}$.
- The $(i+1)$-th element to the w-th element are $-\frac{2}{2^{i+1}}$.

Thus, the inner product of $\mathbf{g}$ and the i-th column of $I - A$ is

$$
\begin{aligned}
&-\sum_{k=1}^{i-1}\left(\left(\frac{1}{2}\right)^{i-k}\frac{1}{2^i} + \left(\frac{1}{2}\right)^{i-k}\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right)\\
&+\left(1-\frac{1}{2^i}-\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{i-1}}{2^{i+1}-1}\right)\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\right)\\
&-\sum_{k=i+1}^{w-1}\frac{2}{2^{i+1}}\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) - \frac{2}{2^{i+1}}2\prod_{j=1}^{w-1}\frac{2^{j-1}}{2^{j+1}-1}\\
&=-\sum_{k=1}^{i-1}\left(\left(\frac{1}{2}\right)^{i-k}\frac{1}{2^i} + \left(\frac{1}{2}\right)^{i-k}\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right)\\
&+\left(1-\frac{1}{2^i}-\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{i-1}}{2^{i+1}-1}\right)\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\right) - \frac{4}{2^{i+1}}\prod_{j=1}^{i}\frac{2^{j-1}}{2^{j+1}-1}\\
&=-\sum_{k=1}^{i-1}\left(\left(\frac{1}{2}\right)^{i-k}\frac{1}{2^i} + \left(\frac{1}{2}\right)^{i-k}\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right)\\
&+\left(1-\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{i-1}}{2^{i+1}-1}\right)\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\right) - \frac{4}{2^{i+1}}\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\\
&=-\sum_{k=1}^{i-1}\left(\left(\frac{1}{2}\right)^{i-k}\frac{1}{2^i} + \left(\frac{1}{2}\right)^{i-k}\frac{1}{2^{i+1}}\right)\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right)\\
&+\frac{2\left(2^{i+1}-2^{i-1}-1\right)}{2^{i+1}}\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1} - \frac{4}{2^{i+1}}\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\\
&=-\sum_{k=1}^{i-1}\left(\left(\frac{1}{2}\right)^{i-k}\frac{3}{2^{i+1}}\right)\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) + \frac{3\left(2^i-2\right)}{2^{i+1}}\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\\
&=\frac{3}{2^{i+1}}\left(-\sum_{k=1}^{i-1}\left(\frac{1}{2}\right)^{i-k}\left(2\left(1-\frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) + \left(2^i-2\right)\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}\right).
\end{aligned}
$$

We now define $z(i)$ as

$$z(i) = -\sum_{k=1}^{i-1} \left(\frac{1}{2}\right)^{i-k} \left(2\left(1 - \frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) + \left(2^i - 2\right)\prod_{j=1}^{i-1}\frac{2^{j-1}}{2^{j+1}-1}$$

If $z(i) = 0$, the inner product of $\mathbf{g}$ and the i-th column of $I - A$ is 0 when $i \geq 2$. We now proof $z(i) = 0$ by mathematical induction. When $i = 2$,

$$z(2) = -\left(1 - \frac{1}{3}\right) + \left(2^2 - 2\right)\frac{1}{3} = 0.$$

We now assume that $z(i) = 0$ when $i = l$ $(l \geq 2)$. When $i = l + 1$,

$$\begin{aligned}
z(l+1) &= -\sum_{k=1}^{l} \left(\frac{1}{2}\right)^{l+1-k} \left(2\left(1 - \frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) + \left(2^{l+1} - 2\right)\prod_{j=1}^{l}\frac{2^{j-1}}{2^{j+1}-1} \\
&= -\frac{1}{2}\sum_{k=1}^{l-1} \left(\frac{1}{2}\right)^{l-k} \left(2\left(1 - \frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) \\
&\quad - \left(1 - \frac{2^{l-1}}{2^{l+1}-1}\right)\prod_{j=1}^{l-1}\frac{2^{j-1}}{2^{j+1}-1} + \left(2^{l+1} - 2\right)\prod_{j=1}^{l}\frac{2^{j-1}}{2^{j+1}-1} \\
&= -\frac{2^l - 2}{2}\prod_{j=1}^{l-1}\frac{2^{j-1}}{2^{j+1}-1} - \left(1 - \frac{2^{l-1}}{2^{l+1}-1}\right)\prod_{j=1}^{l-1}\frac{2^{j-1}}{2^{j+1}-1} \\
&\quad + \left(2^{l+1} - 2\right)\frac{2^{l-1}}{2^{l+1}-1}\prod_{j=1}^{l-1}\frac{2^{j-1}}{2^{j+1}-1} \\
&= \left(-\frac{2^l - 2}{2} + \frac{-2^{l+1} + 1 + 2^{l-1} + 2^{l+1}2^{l-1} - 2^l}{2^{l+1}-1}\right)\prod_{j=1}^{l-1}\frac{2^{j-1}}{2^{j+1}-1} \\
&= \left(-\frac{2^l - 2}{2} + 2^{l-1} - 1\right)\prod_{j=1}^{l-1}\frac{2^{j-1}}{2^{j+1}-1} = 0.
\end{aligned}$$

Thus, $z(i) = 0$ for all i satisfying $2 \leq i \leq w - 1$. Therefore, the inner product of $\mathbf{g}$ and the i-th column of $I - A$ is 0 when $2 \leq i \leq w - 1$.

Lastly, the inner product of $\mathbf{g}$ and the w-th column of $I - A$ is

$$\begin{aligned}
&-\sum_{k=1}^{w-1} \frac{1}{2^{w-1-k}}\frac{1}{2^w}\left(2\left(1 - \frac{2^{k-1}}{2^{k+1}-1}\right)\prod_{j=1}^{k-1}\frac{2^{j-1}}{2^{j+1}-1}\right) + \left(1 - \frac{2}{2^w}\right)2\prod_{j=1}^{w-1}\frac{2^{j-1}}{2^{j+1}-1} \\
&= \frac{2z(w)}{2^w} = 0.
\end{aligned}$$

As a result, we proved the followings:

- The inner product of $\mathbf{g}$ and the first column of $I - A$ is 1.
- Otherwise, 0.

Therefore, the first row of $(I - A)^{-1}$ is $\left[g(0) \quad \ldots \quad g(w-2) \quad 2\prod_{j=1}^{w-1}\frac{2^{j-1}}{2^{j+1}-1}\right]$.

We now calculate Z_i. We recover j $(0 \leq j \leq w-i-1)$ bits when the neighbor higher-order window moves i $(0 \leq i \leq w-1)$, the number of **0**s between the current and neighboring higher-order windows is k, and the number of lower-order bits of $\underline{\mathbf{1}}$ is $i+j+k$ in the current window. Thus,

$$\begin{aligned}\sum_{a_m=0}^{w-1} p_{i,a_m,j} &= \sum_{k=0}^{w-i-j-2} \left(\frac{1}{2}\right)^k \frac{2^{w-(i+j+k+2)}}{2^w} + \left(\frac{1}{2}\right)^{w-i-j-1} \frac{1}{2^w} \\ &= \frac{1}{3}\left(\frac{1}{2}\right)^{i+j} + \frac{1}{3}\frac{1}{2^w}\left(\frac{1}{2}\right)^{w-i-j-1}.\end{aligned}$$

Therefore,

$$Z_i = \frac{1}{3 \cdot 2^i} \sum_{j=0}^{w-i-1} j\left(\frac{1}{2}\right)^j + \frac{2^{i+1}}{3 \cdot 2^{2w}} \sum_{j=0}^{w-i-1} j2^j = f_w(i).$$

Now, because of $f_w(w-1) = 0$, $E[Y_i] = \sum_{k=0}^{w-2} f_w(k)g(k)$.

In conclusion, the average recovery rate of the secret exponent of Rule 1 is

$$\frac{E[Y_i]}{E[X_i]} = \frac{\sum_{k=0}^{w-2} f_w(k)g(k)}{2(w+1)}.$$

This proves Lemma 2. □

Bit Recovery Rate in Rule 2

We prove the following Lemma 3.

Lemma 3 Suppose that we generate a secret exponent randomly. If $w \geq 2$, the average recovery rate of the secret exponent by Rule 2 is given by

$$\frac{2^w - 1}{2^{w-1}(2^{w-1}+1)} \frac{1}{3(w+1)}.$$

Before we prove Lemma 3, we give the intuition of our proof. In Rule 2, we recover bits when the position of the neighboring lower-order window is fixed. Thus, if the window's position does not move, there is a dependency on nearby windows. However, if the window's position moves to the highest-order candidate, there is no dependency on the nearby windows. Therefore, we set the renewal as the hitting window that moves certainly. Then, we calculate the recovery rate of the secret exponent. We now provide rigorous proof.

First, we define X_i and then calculate $E[X_i]$. In this proof, we write the window **1x** ... **x1** as **F** and a window except for **1x** ... **x1** as **L**, similar as the proof of Lemma 2. We define X_i as the length of the bit sequence until

1. **0** ... **0L** occurs, or
2. **0** ... **0F** occurs and continues until **0** ... **0L** occurs with more than one **0**s before **L**.

Before we explain these conditions, we check that the neighboring higher-order window moves certainly to the highest-order candidate. In the first condition, the current window moves certainly to the highest-order candidate, because of the following reasons:

- The neighboring higher-order window moves to the highest-order candidate.
- There is space for allowing the current window to move to the MSB side.
- A window **L** can be moved to the MSB side.

We call this condition pattern 1. Next, in the second condition, **0** ... **0F** is always a highest-order candidate, and this window does not move. Thus, we ensure that the window certainly moves when satisfying the following two conditions:

- There is space for allowing the current window to move to the MSB side.
- A window **L** can be moved to the MSB side.

Therefore, only when we hit the bit sequence of **0** ... **0L** with more than one **0**s before **L**, the window moves certainly. We call this condition pattern 2. We now consider these patterns.

In pattern 1, when the length of **0**s is z ($z \geq 0$), $X_i = z + w$ and the probability of this event is $1/2^{z+2}$. Next, we consider pattern 2. We write window without considering contents as **W**. Pattern 2 is constructed in the following order:

1. After **0** ... **0F**, **W** occurs u ($u \geq 0$) times.
2. After **0** ... **0F** with more than one **0**s, **W** occurs u_j ($u_j \geq 0$) times. This pattern occurs r ($r \geq 0$) times.
3. **0** ... **0L** with more than one **0**s occur.

We now set the number **0**s in first **0** ... **0F** as z_f ($z_f \geq 0$), the number of **0**s in repeated **0** ... **0F** r times as z_j ($z_j \geq 1$), and the number of **0**s in final **0** ... **0L** as z_c ($z_c \geq 1$); then,

$$X_i = z_f + \sum_{j=1}^{r} z_j + z_c + w \left(r + 2 + u + \sum_{j=1}^{r} u_j \right).$$

The probability of this event is

$$\left(\frac{1}{2}\right)^{z_f} \frac{1}{4} \left(\frac{1}{2}\right)^{u} \left(\frac{1}{2}\right)^{z_c} \frac{1}{4} \prod_{j=1}^{r} \left(\left(\frac{1}{2}\right)^{z_j} \frac{1}{4} \left(\frac{1}{2}\right)^{u_j} \right).$$

From these calculations, we calculate the expectation of X_i as $E[X_i] = 3w + 3$.

Next, we calculate $E[Y_i]$. We now write $\underline{\mathbf{1}}\mathbf{00}$ ... **0** window as **U**, which is the abbreviation of the **U**ltra-short window called in the original study. Then, a bit sequence can be decomposed into the following:

1. **0** ... **0F**
2. **W** except for **U** r times, and **U** ($r \geq 0$)
3. **W** except for **U** c times ($c \geq 0$)
4. **0** ... **0L**

We call these patterns $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{P}_4$, respectively. By dividing a bit sequence following $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{P}_4$ and when we write a bit sequence by the index of the pattern, we can write a bit sequence as

$$\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3 \ldots \mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3\mathbf{P}_4,$$

namely, many times $\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3$ and once $\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3\mathbf{P}_4$. In Rule 2, bits are recovered in $\mathbf{P}_1$ and $\mathbf{P}_2$. Thus, in the bit strings $\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3$ or $\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3\mathbf{P}_4$, followings are satisfied:

- If the number of $\mathbf{P}_2$ represented r is more than one, we recover $r + 1$ bits.

- Otherwise, we recover no bits.

We now write $\mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_2\mathbf{P}_3$ as $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ omitting the repetition of $\mathbf{P}_2$. We set the repeat count of $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ without first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as s $(s \geq 0)$, and set as follows:

- The number of **0**s in **0** ... **0F** in the first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as z_f $(z_f \geq 0)$
- The number of $\mathbf{P}_2$ in the first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as r $(r \geq 0)$ and the number of **W** in each $\mathbf{P}_2$ as c_j $(c_j \geq 0)$
- The number of **W** in $\mathbf{P}_3$ in the first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as c_e $(c_e \geq 0)$
- The number of **0**s in **0** ... **0F** except fot the first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as z_i $(z_i \geq 1)$
- The number of $\mathbf{P}_2$ except fot the first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as r_i $(r_i \geq 0)$, the number of **W** in each $\mathbf{P}_2$ as $c_{i,j}$ $(c_{i,j} \geq 0)$
- The number of **W** in $\mathbf{P}_3$ except fot the first $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ as $c_{i,e}$ $(c_{i,e} \geq 0)$
- The number of **0**s in $\mathbf{P}_4$ as z_c $(z_c \geq 1)$

Then, the number of recovered bits is given by Rule 2 as

$$\mathbf{1}(r \geq 1) + \sum_{j=1}^{r} c_j + \sum_{i=1}^{s} \left(\mathbf{1}(r_i \geq 1) + \sum_{j=1}^{r_i} c_{i,j} \right),$$

where $\mathbf{1}(x)$ is an indicator function. Then, the probability of the above bit sequences is

$$\left(\frac{1}{2}\right)^{z_f} \frac{1}{4} \left(\frac{1}{2} - \frac{1}{2^w}\right)^{c_e} \left(\frac{1}{2}\right)^{z_c} \frac{1}{4} \prod_{j=1}^{r} \left(\left(\frac{1}{2} - \frac{1}{2^w}\right)^{c_j} \frac{1}{2^w} \right)$$
$$\times \prod_{i=1}^{s} \left(\left(\frac{1}{2}\right)^{z_i} \frac{1}{4} \left(\frac{1}{2} - \frac{1}{2^w}\right)^{c_{i,e}} \prod_{j=1}^{r_i} \left(\left(\frac{1}{2} - \frac{1}{2^w}\right)^{c_{i,j}} \frac{1}{2^w} \right) \right).$$

Therefore, $E[Y_i] = \dfrac{2^w - 1}{2^{w-1}(2^{w-1} + 1)}$.

In conclusion, the average recovery rate of the secret exponent of Rule 2 is

$$\frac{E[Y_i]}{E[X_i]} = \frac{2^w - 1}{2^{w-1}(2^{w-1} + 1)} \frac{1}{3(w+1)}.$$

This proves Lemma 3. □

Bit Recovery Rate in Rule 3

We prove the following Lemma 4.

Lemma 4 Suppose that we generate a secret exponent randomly. The average recovery rate of the secret exponent by Rule 3 is then given by $1/(w+1)$.

Before we prove Lemma 4, we give the intuition of our proof. In Rule 3, we consider the highest-order candidate in each window and recover bits not included in windows. Then, we move the actual candidate to the highest-order candidate. Therefore, we consider the same as that in Rule 1, set the renewal as hitting window **1x** ... **x1** and calculate the bit recovery rate. We now give rigorous proof.

In proof of Lemma 4, we define X_i same as the proof in Lemma 2. Therefore, $E[X_i] = 2w + 2$.

Next, we calculate $E[Y_i]$. We set the number of **0**s in **0** ... **0F** as z $(z \geq 0)$. Then, the pattern of window until **0** ... **0F** occurs is one of following:

Table 3.1: Average recovery rate of secret exponents in each rule

w	Rule 0 (%)	Rule 1 (%)	Rule 2 (%)	Rule 3 (%)	all (%)
3 (Experimental)	25.01	8.06	2.93	24.96	60.97
3 (Theoretical)	25.00	8.04	2.92	25.00	60.95
4 (Experimental)	20.00	8.41	1.38	19.99	49.78
4 (Theoretical)	20.00	8.42	1.39	20.00	49.81
5 (Experimental)	16.67	7.94	0.63	16.67	41.90
5 (Theoretical)	16.67	7.95	0.63	16.67	41.92
6 (Experimental)	14.29	7.24	0.28	14.29	36.09
6 (Theoretical)	14.29	7.24	0.28	14.29	36.09
7 (Experimental)	12.50	6.49	0.13	12.53	31.65
7 (Theoretical)	12.50	6.52	0.13	12.50	31.65

1. **0** ... **0L** does not occur.
2. **0** ... **0L** occurs r ($r \geq 1$) times.

In pattern 1, $Y_i = z$ and the probability of the bit sequence is $\left(\frac{1}{2}\right)^z \frac{1}{4}$. In pattern 2, when we set the number of zeros in each **0** ... **0L** as $z_1, \ldots, z_r$ ($z_i \geq 0$), $Y_i = z + \sum_{i=1}^{r} z_i$ and the probability of bit sequence is $\left(\frac{1}{4}\right)^{r+1} \left(\frac{1}{2}\right)^z \prod_{i=1}^{r} \left(\frac{1}{2}\right)^{z_i}$. Therefore, $E[Y_i] = 2$.

In conclusion, the average recovery rate of the secret exponent in Rule 3 is

$$\frac{E[Y_i]}{E[X_i]} = \frac{1}{w+1}.$$

This proves Lemma 4. □

3.3.2 Numerical Experiment: Calculating the Exact Rate of Recovered Bits of the Secret Exponent

To check our analysis's validity, we randomly generated 10000 bits, converted them into square-and-multiply sequences, and applied the optimal recovery rules for bits of a secret exponent given in Section 3.2.2. We performed 1000 time experiments. We calculated the average rate of the recovered bits by each rule. Table 3.1 shows the results of this experiment. Table 3.1 shows that our analysis matches with the experimental result.

3.3.3 Numerical Experiment: Applying to the CRT-RSA Scheme

We applied the optimal recovery rules on the CRT-RSA secret keys d_p and d_q. This experiment checked if the recovery rates are similar between the CRT-RSA secret keys and random bits. We generated secret keys on the 2048-bit CRT-RSA scheme and generated square-and-multiply sequences. In each CRT-RSA secret key, we generated square-and-multiply sequences on (d_p, d_q), and therefore, obtained two square-and-multiply sequences. We applied the optimal recovery rules on these two square-and-multiply sequences and calculated the rates of recovered bits of CRT-RSA secret keys. We repeated the above for 100 CRT-RSA secret keys generated randomly. Then, we averaged all knowable bit rates over 100 times results. Moreover, we generated 1024 bits randomly 200 times, generated

Table 3.2: Rates of the recovered bits in the 2048-bit CRT-RSA scheme

w	3	4	5	6	7
CRT-RSA (%)	60.80	49.96	41.84	36.19	31.76
Random Bits (%)	60.97	49.80	41.89	35.98	31.65

square-and-multiply sequences, applied the optimal bit recovery rules, and averaged the rates of recovered bits of CRT-RSA secret keys over 200 times results.

Table 3.2 shows our experimental results. The recovery rates of bits of the CRT-RSA secret keys and random bits are almost the same. Therefore, the rates of recovered bits are similar between the CRT-RSA secret keys and random bits.

3.4 Obtaining More Information on Non-Recovered Bits

The previous section gave the exact rate of recovered bits on the CRT-RSA secret keys. However, when we only use recovered bits, the CRT-RSA secret keys are not recovered in polynomial time when $w = 4$ because the exact rate of recovered bits of the CRT-RSA secret keys is less than 50% [87]. This result contradicts the previous work that the CRT-RSA secret keys are recovered in polynomial time when $w = 4$ [10].

We now focus on non-recovered bits. The additional information is embedded in non-recovered bits, as dictated in [10, 115]. We capture this additional information by calculating the likelihood of each bit value **0** or **1** in non-recovered bits. For example, we consider **SSMSSM** in $w = 2$. Bit sequences **0101**, **1101**, and **1111** do not contradict with **SSMSSM**. Common bits are the second and fourth ones; thus, the recovered bits are **x1x1**. If there is no information on the non-recovered bits, namely the first and third bits, we have $2^2 = 4$ candidates. However, there are 3 candidates. The likelihoods of bit values in each non-recovered bit are not the same. Therefore, the additional information on non-recovered bits is embedded in the likelihood of each bit value.

This section proposes a method for obtaining the likelihood of each bit value in each non-recovered bit. For this purpose, we adopt a Monte-Carlo approach. First, we choose many bit sequences uniformly that do not contradict a given square-and-multiply sequence. However, the method for uniformly choosing the bit sequences is not trivial. To construct this random sampling method, we build our method based on Van Vredendaal's method [115] to calculate the number of candidates of bit sequence corresponding to the given square-and-multiply sequence. Van Vredendaal's method is a straightforward dynamic programming approach that calculates the number of candidates of a bit sequence in each window. We use this information, the number of candidates of a bit sequence, in our method for obtaining the likelihood of each bit value in each non-recovered bit.

Section 3.4.1 gives a random sampling method to choose a bit sequence uniformly that does not contradict a given square-and-multiply sequence. Section 3.4.2 shows the result of the numerical experiments performed for calculating the likelihood of each bit value.

3.4.1 Random Sampling Method Based on a Given Square-and-Multiply Sequence

We propose a random sampling method for a bit sequence that does not contradict a given square-and-multiply sequence. Our random sampling method is given as Algorithm 5. When we sample many outputs corresponding to the input, we run Step 1 once and Step 2 many times.

Algorithm 5 Method for random sampling of a bit sequence corresponding a square-and-multiply sequence

Input: The window size w, a square-and-multiply sequence
Output: An input candidate that does not contradict the given square-and-multiply sequence
Step 1: Calculate the Number of Candidates
from The lowest-order window **to** the highest-order window
for all possible windows
1. Sum of the number of candidates of the neighboring lower-order windows that does not have common bits with the current window (A).
2. Calculate the number of candidates of the current window (B).
3. Calculate A times B and store it in the current candidate window.

end for
Step 2: Sampling a Bit Sequence
from The highest-order window **to** the lowest-order window
1. Define $X_1, X_2, \ldots, X_k$ as the number of candidates for all possible current windows that do not overlap with the neighboring higher-order window.
2. Choose a window with probability $X_i / \left(\sum_{j=1}^{k} X_j\right)$.
3. Set 0 between the current window and the neighboring higher-order window.
4. In the current window,
   a. Set the MSB bit as **1**.
   b. Set lower-order bits of $\underline{\mathbf{1}}$ as **0**.
   c. Set nondetermined bits as **0** or **1** randomly.

Step 1 of Algorithm 5 corresponds to Van Vredendaal's method [115] for calculating the number of candidates of a bit sequence corresponding to the given square-and-multiply sequence. Step 1 calculates the number of candidates of a bit sequence in each possible position of windows from low-order windows. The value A summarizes the information of low-order windows. The value B calculates the number of candidates of the current window. We calculate 2^b, where b is the number of non-determined bits in the current window. By storing A times B in each window, we preserve the number of candidates in each possible position of windows, including the information of low-order windows. When we finish Step 1, we obtain the number of candidates of the bit sequence corresponding to the given square-and-multiply sequence.

In Step 2, we sample a bit sequence uniformly corresponding to the given square-and-multiply sequence. We determine the position of windows from higher windows. When some windows' position is determined, we choose the neighboring low-order windows based on the number of candidates of each window in Step 1. This selection method realizes sampling uniformly corresponding to the given square-and-multiply sequence. When the position of a window is determined, we set **0** between the current window and the neighboring higher-order window (leading zeros), set the MSB bit as **1** (leading one), and the lower-order bits of $\underline{\mathbf{1}}$ as **0** (trailing zeros). Moreover, we set the non-determined bits as **0** or **1** randomly because these bits are flat when the current window position is determined. When we finish Step 2, we sample a bit sequence that does not contradict the square-and-multiply sequence uniformly.

Table 3.3: Distribution of likelihood of **1** in 2048-bit CRT-RSA scheme

w	3	4	5	6	7
All 0	0.328	0.285	0.248	0.212	0.189
All 1	0.280	0.214	0.174	0.147	0.127
0-10%	0	0	0	0	0
10-20%	3.38×10^{-4}	3.91×10^{-5}	4.89×10^{-6}	0	0
20-30%	1.02×10^{-2}	5.17×10^{-3}	2.52×10^{-3}	1.41×10^{-3}	9.00×10^{-4}
30-40%	3.27×10^{-2}	2.45×10^{-2}	1.93×10^{-2}	1.55×10^{-2}	1.30×10^{-2}
40-50%	7.99×10^{-2}	0.104	0.133	0.168	0.199
50-60%	0.143	0.230	0.295	0.338	0.363
60-70%	5.90×10^{-2}	6.42×10^{-2}	5.86×10^{-2}	5.23×10^{-2}	4.74×10^{-2}
70-80%	3.20×10^{-2}	3.13×10^{-2}	2.73×10^{-2}	2.55×10^{-2}	2.23×10^{-2}
80-90%	2.31×10^{-2}	2.39×10^{-2}	2.29×10^{-2}	2.02×10^{-2}	1.84×10^{-2}
90-100%	1.22×10^{-2}	1.80×10^{-2}	2.00×10^{-2}	2.04×10^{-2}	1.90×10^{-2}
non-recovered	400	512	591	655	699
$\log_2$(#Cand.)	350	448	534	608	660

3.4.2 Numerical Experiment: Calculating the Likelihood in Each Non-Recovered Bit

We calculated the likelihood of each bit in CRT-RSA secret keys using Algorithm 5. We performed numerical experiments on $3 \leq w \leq 7$. In each w, we generated the 2048-bit CRT-RSA secret keys 100 times randomly. In each CRT-RSA secret key, we generated square-and-multiply sequences on (d_p, d_q), and therefore, obtained two square-and-multiply sequences. Therefore, we generated 200 square-and-multiply sequences corresponding to a 1024-bit number. In each square-and-multiply sequence, we obtained 1000 samples using Algorithm 5 and calculated the likelihood of **1**. Moreover, we calculated the number of non-recovered bits and the number of candidates that are not the same in 1000 samples. Finally, we averaged the above information over 200 square-and-multiply sequences and calculated the likelihood of **1**, the average of the number of non-recovered bits, and $\log_2$(the average of #Candidate). Table 3.3 shows the experimental results. In Table 3.3 and the following discussion, we represent the average of the number of non-recovered bits as non-recovered and $\log_2$(the average of #Candidate) as $\log_2$(#Cand.). Moreover, we drop fractions in these two values.

Table 3.3 describes that non-recovered and $\log_2$(#Cand.) are larger in larger w. This result agrees with our intuition that it is more difficult to recover the CRT-RSA secret keys in a larger w. Moreover, the likelihood of **1** gathers at 40-60% in a larger w, which is why it is more difficult to recover the CRT-RSA secret keys in a larger w.

When $w = 4$, non-recovered is 512 and $\log_2$(#Cand.) is 448. Non-recovered is almost the same as $1024/2 = 512$ and $\log_2$(#Cand.) is smaller than $1024/2 = 512$. Therefore, this corresponds to the fact that we can recover the CRT-RSA secret keys when $w = 4$ in the original analysis [10].

When $w = 5$, non-recovered is 591 and $\log_2$(#Cand.) is 534. Both non-recovered and $\log_2$(#Cand.) are larger than $1024/2 = 512$. Therefore, it is not easy to recover the CRT-RSA secret keys when $w = 5$ in polynomial time. However, there are about 2% bits, which are **1** with high probability. This information corresponds to about 20 bits. In the next section, we use this information for recovering the CRT-RSA secret keys.

3.5 New Recovering Method for the CRT-RSA Secret Keys

We now propose a new method for recovering CRT-RSA secret keys. Our method is a combination of the original method [10] and Kunihiro et al.'s method [59]. We propose our new method in Section 3.5.1, and we show the results of our experiment in Section 3.5.2.

3.5.1 Our Proposed Method

We now explain our proposed method. In our method, we give inputs as follows:

- The square-and-multiply sequences of d_p and d_q
- r: The number of iteration of the random sampling algorithm proposed in Section 3.4
- l: The maximum number of leaves we search
- ε: The parameter for recovering additional bits
- t and c: The parameter for running Kunihiro et al.'s method

Then, we recover the CRT-RSA secret keys.

First, we extract information of d_p and d_q from square-and-multiply sequences. Specifically, we perform the followings:

- We recover bits partially using the optimal recovery rules for bits of the CRT-RSA secret keys.
- We run our random sampling algorithm r times on each d_p and d_q. In each d_p and d_q we recover bits as $\mathbf{0}$ when the likelihood of $\mathbf{1}$ is less than ε, and $\mathbf{1}$ when the likelihood of $\mathbf{1}$ is more than $1-\varepsilon$.
- We record the position of $\mathbf{S}$s which are next to a $\mathbf{M}$ or at the beginning of w $\mathbf{S}$s in the square-and-multiply sequences from the final operation as Section 3.2.3. Then we connect these $\mathbf{S}$s with bits.

We label each bit by matching the above conditions. Specifically, we define R as the position of the recovered bits. Next, we define P as the position of the additional recovered bits. Finally, we define S as the bits' position connected with $\mathbf{S}$s.

We now recover the CRT-RSA secret keys. We perform the branch and bound strategy on bits at the position in P. First, we calculate t-revealed bits of d_p and d_q at the position in P. Second, we discard a leaf if there are more than c mismatches between the calculated and given t bits. Additional to this strategy, we adopt exception handling on the bits at R or S. First, if we calculate a bit in R, we discard a leaf if there is a mismatch with the recovered bits. Next, if we calculate a bit in S, we convert calculated bits hitting S to a square-and-multiply sequence, and we discard a leaf if there are mismatches with the given square-and-multiply sequence. By doing these, we recover the CRT-RSA secret keys from square-and-multiply sequences.

We now compare our proposed method to the original method [10] which only used the information of bits in S. Our method performs more pruning using the additional information than the original method. Thus, our proposed method generates fewer leaves while there is a possibility of discarding the correct leaves. Therefore, our method may recover more CRT-RSA secret keys because fewer leaves are generated than the original method.

Table 3.4: Results of our proposed method in the 2048-bit CRT-RSA scheme

	ε	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
	t	1	100	50	33	25	20	16	14	12	11	10
$l = 1,000,000$	Time (s)	8.37	7.71	8.27	7.12	7.23	16.6	8.70	11.3	19.1	14.2	32.9
	Success Rate (%)	11	12	11	17	12	14	12	16	9	11	8
	Too Many (%)	89	88	89	83	88	86	83	78	86	74	75
	Pruning (%)	0	0	0	0	0	0	5	6	5	15	17
$l = 2,000,000$	Time (s)	12.1	11.2	15.9	17.0	14.0	27.8	8.50	12.6	20.4	17.5	12.6
	Success Rate (%)	19	16	14	16	19	17	20	21	11	15	10
	Too Many (%)	81	84	86	84	80	83	79	74	78	76	73
	Pruning (%)	0	0	0	0	1	0	1	5	11	9	17

3.5.2 Numerical Experiments of Our Proposed Method

We now recover the CRT-RSA secret keys using our proposed method. We ran our algorithm on $w = 5$ and set the number of random sampling as $r = 1000$. Moreover, we set (ε, t, c). If $\varepsilon = 0$, we set $t = 1$ and $c = 0$, which corresponds to the combination of two methods proposed in the original method [10]. In other ε, we set $c = 1$ and $t = \lfloor 1/\varepsilon \rfloor$. In our experiment, we implemented our algorithm in depth-first search and aborted if we search l leaves similar to [10]. In each parameter, we generated secret keys of the 2048-bit CRT-RSA scheme 100 times randomly. We measured average time in successful trials and success rates when the correct k_p and k_q were given.

Moreover, we recorded why our proposed method failed. There are two reasons for failure:

- Generate more than l leaves
- Prune the correct leaf

In the original method, the reason for failure is only the former. However, stated above, our proposed method may discard the correct leave, and then, our method fails. Table 3.4 gives the result of the experiment. "Too Many" means the failure because of searching more than l leaves, and "Pruning" means the failure because of pruning the correct leaf.

When $l = 1,000,000$, our method recovers 17% of the CRT-RSA secret keys when $(\varepsilon, t, c) = (0.03, 33, 1)$, while the original method [10] recovers 8.6% of the CRT-RSA secret keys. When $l = 2,000,000$, our method recovers 21% of the CRT-RSA secret keys when $(\varepsilon, t, c) = (0.07, 14, 1)$, while the original method recovers 13% of the CRT-RSA secret keys. In almost all parameters, our method recovers more CRT-RSA secret keys than the original method. Therefore, our method recovers more secret keys compared to the original method when $w = 5$.

We now focus on the reason for failure. The probability of too many leaves is 70–90%, which is much higher than the probability of pruning the correct leaf. This result occurs because we have a small chance to hit the correct leaf because of too many leaves originally. In larger ε, the failure probability of too many leaves decreases, especially $\varepsilon \geq 0.09$ when $l = 1,000,000$, and $\varepsilon \geq 0.06$ when $l = 2,000,000$. This result matches our intuition that our proposed method generates fewer leaves in larger ε. Moreover, in larger ε, the failure probability of pruning the correct leaf increases, especially $\varepsilon \geq 0.06$ in both l. This result matches our intuition that our proposed method tends to discard the correct leaf in a larger ε. When we set $\varepsilon \leq 0.05$, there is almost no failure because of pruning the correct leaf. However, when we set $\varepsilon \geq 0.08$, our method prunes the correct leaf with high probability. Therefore, the appropriate parameter of our method exists in $0.05 < \varepsilon < 0.08$.

3.6 Conclusion and Future Work

We improved the recovery method for the CRT-RSA secret key from the correct square-and-multiply sequences. First, we calculated the exact rate of recovered bits of the secret exponent only from a square-and-multiply sequence. Next, we extracted the information embedded in non-recovered bits by calculating the likelihood of each bit value. Finally, we proposed a new method for recovering the CRT-RSA secret key and improved the original method [10] when $w = 5$. In the future, we should determine the appropriate parameters in our proposed method.

Chapter 4

Recovering CRT-RSA Secret Keys from Noisy Square-and-Multiply Sequences

4.1 Introduction

4.1.1 Background

This chapter focuses on the CRT-RSA encryption or signature scheme [76] when noisy square-and-multiply sequences are extracted. It is proved that the CRT-RSA secret keys are recovered in polynomial time from the correct square-and-multiply sequences when $w \leq 4$ [10]. It is also proved that the CRT-RSA secret keys are recovered in polynomial time from square-and-multiply sequences with errors when $w = 1$ [84]. These methods are based on Heninger-Shacham's method [43] for searching candidates of the CRT-RSA secret keys. Naturally, it may be possible to recover the CRT-RSA secret keys in polynomial time from square-and-multiply sequences with errors when $w \leq 4$.

Based on these studies, Oonishi and Kunihiro proposed a method for recovering the CRT-RSA secret keys in a general w [83]. However, this method for recovering the CRT-RSA secret keys adopts the slower method than Heninger-Shacham's method for searching candidates of the CRT-RSA secret keys. The previous method is slow because square-and-multiply sequences are generated from the MSB side, whereas Heninger-Shacham's method recovers the CRT-RSA secret keys from the LSB side. With Heninger-Shacham's method, even if we obtain the partially correct LSB bits of the CRT-RSA secret keys, these bits are not always converted into the correct square-and-multiply sequences. Thus, Oonishi-Kunihiro's method [83] recovers the CRT-RSA secret keys from MSB sides, which requires additional time. Moreover, it is proved that the CRT-RSA secret keys are recovered in polynomial time from square-and-multiply sequences with errors only when $w = 2$. Thus, we tackle the following two questions:

1. How do we recover the CRT-RSA secret keys from square-and-multiply sequences with errors based on Heninger-Shacham's method?
2. How many errors are tolerable when we recover the CRT-RSA secret keys?

4.1.2 Our Contribution

This study discusses how to recover the CRT-RSA secret keys from square-and-multiply sequences with errors in the sliding window method by answering these two questions. This chapter is written based on our study published at the 25th Australasian Conference on Information Security and Privacy (ACISP2020) [85].

First, we propose a method for recovering the CRT-RSA secret keys from square-and-

Table 4.1: Tolerable error rate of our method

w	1	2	3	4
Tolerable Error Rate	0.108	0.067	0.034	0.008

multiply sequences with errors based on Heninger-Shacham's method, which answers the first question. Our method is an extension of Oonishi-Kunihiro's method [83, 84]. Specifically, we focus on obtaining the correct square-and-multiply sequences from the LSBs of the CRT-RSA secret keys.

Second, we analyze our method, which is the answer to the second question. To analyze our method, we assume that **S** flips into **M** with probability δ, and **M** flips into **S** with probability δ. We show that our method recovers the CRT-RSA secret keys when the error rate δ is less than the values in Table 4.1. There are no ranges of the error rate δ for which our method can recover the CRT-RSA secret keys when $w \geq 5$. Table 4.1 shows that our method recovers the CRT-RSA secret keys from square-and-multiply sequences with a higher error rate than previous research. When $w = 1$, our method recovers the CRT-RSA secret keys when $\delta = 0.108$, compared with the maximum error rate of 0.058 reported in [84]. Moreover, when $w = 2$, our method recovers the CRT-RSA secret keys when $\delta = 0.067$, as compared with the maximum error rate of 0.017 reported in [83]. These error rate values of δ may seem small, but the error rate reported in [10] is $\delta = 0.011$. Therefore, our method recovers the CRT-RSA secret keys from actual side-channel information.

Finally, we conduct numerical experiments using the proposed method. We first perform numerical experiments with a various error rate of δ values in square-and-multiply sequences. Through these experiments, our method recovers the CRT-RSA secret keys when δ takes the values given in Table 4.1. Next, we consider the real error $\delta = 0.011$ in the square-and-multiply sequences, as shown in [10]. From Table 4.1, it is impossible to recover the CRT-RSA secret keys when $w = 4$ and $\delta = 0.011$. However, our experiments show that our method recovers the CRT-RSA secret keys when $w = 4$ and $\delta = 0.011$. Therefore, our method recovers CRT-RSA secret keys from actual side-channel information in the sliding window method.

4.1.3 Organization of this Chapter

This chapter has six sections. In Section 4.2, we describe previous results on recovering the CRT-RSA secret keys from square-and-multiply sequences. In Section 4.3, we propose a recovery method from noisy square-and-multiply sequences. In Section 4.4, we analyze our proposed recovery method. In Section 4.5, we provide the results of numerical experiments. In Section 4.6, we conclude this chapter.

4.2 Previous Methods for Recovering the CRT-RSA Secret Keys

This section describes previous studies. First, we describe the method for recovering the CRT-RSA secret keys directly from the correct square-and-multiply sequences [10]. This method is detailed in Section 3.2.3, and we provide only an analysis in Section 4.2.1, which is extended to our approach. Next, we describe the method for recovering the CRT-RSA secret keys from the noisy square-and-multiply sequences [83, 84] in Section 4.2.2. Finally, we discuss these methods for constructing an efficient method for recovering the CRT-RSA secret keys from the noisy square-and-multiply sequences in Section 4.2.3.

Table 4.2: Collision entropy rate H for each w [13]

w	1	2	3	4	5	6	7
H	1.000	0.786	0.652	0.545	0.460	0.395	0.345

4.2.1 Method for Recovering the CRT-RSA Secret Keys from the Correct Square-and-Multiply Sequences [10]

In theory, the CRT-RSA secret keys are recovered in polynomial time in n when the window size w is less than or equal to 4. In the proof, the number of leaves that do not contradict the given square-and-multiply sequences is evaluated by calculating the collision entropy rate H. Let us define the collision entropy rate H. Now, s_t is defined as the set of all candidate square-and-multiply sequences from t bits. Then, the collision entropy rate H is defined as

$$H = \lim_{t\to\infty} \left(-\frac{1}{t} \log \sum_{s\in s_t} \Pr[s]^2 \right).$$

Table 4.2 shows the values of H for each w. Here, 2^{-tH} represents the average probability of the calculated sequence not contradicting a given square-and-multiply sequence generated from t bits. This value is less than $2^{-t/2}$ when $w \leq 4$. Therefore, when we guess incorrectly at the 1-st depth, the number of leaves generated from this leaf is

$$\sum_{t=0}^{n/2-1} 2^t \left(2^{-tH}\right)^2 = \sum_{t=0}^{n/2-1} 2^{t(1-2H)} = \frac{1-2^{n(1-2H)/2}}{1-2^{1-2H}},$$

and the expected total number of leaves in the candidate tree of the CRT-RSA secret keys is less than

$$\frac{n}{2}\left(1+\frac{1-2^{n(1-2H)/2}}{1-2^{1-2H}}\right).$$

In conclusion, it is possible to recover the CRT-RSA secret keys from the correct square-and-multiply sequences in polynomial time in n, when window size w is less than or equal to 4.

A method for calculating collision entropy H is given as follows [14]. First, a Markov transition matrix M is generated in a square-and-multiply sequence. Matrix M is calculated by regarding the last w sequential operations as one state. In particular, the position of **M** is mainly considered; **S** is skipped when a window is determined to construct a Markov transition matrix M for each bit. Therefore, the information on the total length of a square-and-multiply sequence is discarded. Second, the given sequence and the calculated sequence are combined into one state and compose a transition matrix. This transition matrix is given as the tensor product of M. Third, we erase the row and column whose input sequence and calculated sequence contradict each other to obtain a calculated sequence that completely matches the given sequence. This final transition matrix M' denotes the hidden Markov model of the square-and-multiply sequence. Finally, H is calculated as follows:

$$H = -\log \lambda_{M'}, \tag{4.1}$$

where $\lambda_{M'}$ is the largest absolute value of the eigenvalues of M'.

4.2.2 Method for Recovering the CRT-RSA Secret Keys from the Noisy Square-and-Multiply Sequences [83, 84]

We described the method for recovering the CRT-RSA secret keys, focusing on the branch-and-bound step. We now detail the method for recovering the CRT-RSA secret keys in the binary method, namely, when $w = 1$ [84]. To calculate the bits of the CRT-RSA secret keys, we repeat the following until recovering all CRT-RSA secret keys: First, we calculate t bits of d_p and d_q using Heninger-Shacham's method [43]. Second, we convert these bits into square-and-multiply sequences in d_p and d_q; **0** is converted into **S** and **1** into **SM** because only the binary method is considered. The square-and-multiply sequences obtained from $2t$ new bits and the given square-and-multiply sequences are compared, and the disagreement rate is calculated. Finally, the leaves, whose disagreement rate is larger than Y, are discarded. In theory, the CRT-RSA secret keys are recovered in polynomial time in n when the error rate δ satisfies $\delta \leq 0.058$. The analysis was conducted through a mathematical induction, and this proof is a different method from that described in [10].

We now explain the method for recovering the CRT-RSA secret keys in the general w [83]. This method recovers the CRT-RSA secret keys from the MSB side and does not use Heninger-Shacham's method, which recovers the CRT-RSA secret keys from the LSB side. To calculate a bit in each CRT-RSA secret key, we repeat the following in order until recovering all CRT-RSA secret keys:

1. Generate all candidates of bits of the CRT-RSA secret keys
2. Discard the candidates that do not satisfy $N = pq$
3. Calculate the disagreement rate and remaining L leaves with a low disagreement rate

These calculations require $O\left(n^2\right)$-time, whereas the corresponding calculations in Heninger-Shacham's method require $O(n)$-time. This difference occurs that we require additions, subtractions, multiplications, and divisions of two n-bit numbers in [83], while we only conduct additions and subtractions of two n-bit numbers with Heninger-Shacham's method. Thus, the calculation of bits of the CRT-RSA secret keys requires more time than Heninger-Shacham's method. We now review the analysis. The CRT-RSA secret keys are recovered in polynomial time in n only when $w \leq 2$. Moreover, when $w = 2$, the error rate δ satisfies $\delta \leq 0.017$. This analysis was performed using a mathematical induction similar to the analysis in the case of the binary method [84]. However, no analysis is given when $w \geq 3$.

4.2.3 Discussion on Previous Methods

The method in [10] depends on the correctness of the square-and-multiply sequences used when the CRT-RSA secret keys are recovered. When there are errors in the given square-and-multiply sequences, it is impossible to determine the pruning position. The method in [84] depends on the one-to-one correspondence between bits and a square-and-multiply sequence with the binary method. In the sliding window method, if a calculated bit sequence is converted into a square-and-multiply sequence naively, there is a difference from the given square-and-multiply sequence even if there are no errors in the calculated bits. Therefore, we cannot straightforwardly extend these methods to reach our goal. For an extension to the sliding window method, we use the slower method [83]. We then want to recover the CRT-RSA secret keys more efficiently by using Heninger-Shacham's method.

4.3 Our Proposed Method for Recovering the CRT-RSA Secret Keys

We now propose a new method for recovering the CRT-RSA secret keys from square-and-multiply sequences with errors. Our method adopts a branch-and-bound strategy based on the Heninger-Shacham's method [43] at each i-th depth of the candidate tree of the CRT-RSA secret keys. In our proposed method, we input the followings:

- Noisy square-and-multiply sequences
- L: The maximum number of leaves we store.

First, we calculate each bit in p, q, d_p, and d_q using Eqs. (3.1)–(3.3). Second, we calculate the distance between the given sequences and the calculated sequences. Finally, we prune the leaves whose rank is higher than L, similar to [58, 87]. The difference between our method and [58, 87] is that our method calculates distance based on square-and-multiply sequences. We figure out this distance to recover the CRT-RSA secret keys based on Heninger-Shacham's method.

We define distance D as the disagreement rate between the given sequences and the calculated sequences on the LSBs of d_p and d_q. First, we define $D_{p,t}$ as the disagreement rate between the given sequences and the calculated sequences generated from the t LSBs of d_p. Similarly, we define $D_{q,t}$ as the disagreement rate between the given sequences and the calculated sequences generated from the t LSBs of d_q. Then, D is defined as

$$D = \max\left(\min_{0\le j\le w-1} D_{p,i+\tau(k_p)+1-j}, \min_{0\le j\le w-1} D_{q,i+\tau(k_q)+1-j}\right).$$

By defining D as the above, we always consider the correct square-and-multiply sequences. We define D as the above because a square-and-multiply sequence is divided into partial sequences generated from at most w-bits. Thus, there are always the starting bits within the sequential w-bits where we can convert bits into the correct square-and-multiply sequence.

We now show an example of $w = 3$ as in Table 4.3. We consider the situation that the square-and-multiply sequence of d_p is given as **SMSSMMS** and the square-and-multiply sequence of d_q is given as **SSMSMSM**. Moreover, we calculate 5 LSBs of d_p as **10110** and 5 LSBs of d_q as **01101**. After converting bits to a square-and-multiply sequence, we calculate the disagreement rate by comparing it from the final operations for both sequences. The value of D is 0.25 as shown in Table 4.3.

Remark 1 Intuitively, we calculate D by converting all t calculated bits into square-and-squaring sequences. However, we can calculate D more efficiently. At the i-th depth of the candidate tree, we store followings:

- The number of operations in the calculated sequences generated from the $i+\tau(k_p)+1-j$ LSBs of d_p as $l_{p,i-j}$ in each j satisfying $0 \le j \le w-1$
- The number of mismatches between the given sequences and the calculated sequences generated from the $i+\tau(k_p)+1-j$ LSBs of d_p as $e_{p,i-j}$ in each j satisfying $0 \le j \le w-1$
- The number of operations in the calculated sequences generated from the $i+\tau(k_q)+1-j$ LSBs of d_q as $l_{q,i-j}$ in each j satisfying $0 \le j \le w-1$
- The number of mismatches between the given sequences and the calculated sequences generated from the $i+\tau(k_q)+1-j$ LSBs of d_q as $e_{q,i-j}$ in each j satisfying

Table 4.3: Example of calculating distance D when $w = 3$

	j	0	1	2
d_p	Bits	**10110**	**0110**	**110**
	Calculated Sequence	**SSSMSMS**	**SSSMS**	**SSMS**
	Given Sequence	**SMSSMMS**	**SSMMS**	**SMMS**
	Disagreement Rate	$3/7 = 0.429$	$1/5 = 0.2$	$1/4 = 0.25$
$\min_t D_{p,t}$	0.2			
d_q	Bits	**01101**	**1101**	**101**
	Calculated Sequence	**SSSMSSM**	**SSMSSM**	**SSSM**
	Given Sequence	**SSMSMSM**	**SMSMSM**	**SMSM**
	Disagreement Rate	$3/7 = 0.429$	$3/6 = 0.5$	$1/4 = 0.25$
$\min_t D_{q,t}$	0.25			
D	0.25			

$$0 \leq j \leq w - 1$$

These values can be renewed based on the values of the $(i + \tau(k_p) + 2)$-th bit of d_p and the $(i + \tau(k_q) + 2)$-th bit of d_q. Then, we can calculate D as

$$D = \max\left(\min_{0 \leq j \leq w-1} \frac{e_{p,i-j}}{l_{p,i-j}}, \min_{0 \leq j \leq w-1} \frac{e_{q,i-j}}{l_{q,i-j}}\right).$$

Remark 2 There are obvious errors in the given sequences, such as subsequent **M**'s. When we heuristically correct one of the subsequent **M**'s, there is a possibility that we change a correct **M** into an **S**, because we do not know which **M** is wrong. When we change the correct **M** into an **S**, our method discards the correct candidate with a higher probability. Therefore, our method uses the given sequences just as they are.

4.4 Analysis of Our Proposed Method

We will now analyze our method. The main result is given as the following Theorem 3.

Theorem 3 We use the values of Y_w shown in Table 4.1 for each $w \leq 4$. Moreover, we assume that the error rate δ satisfies $\delta < Y_w$. If we store L leaves at each level of the candidate tree of the CRT-RSA secret keys, our method will correctly recover n-bit CRT-RSA secret keys in $\max\left(O\left(n^2 L\right), O\left(nL \log L\right)\right)$ time with probability

$$1 - \left(\frac{w^2 2^{2w-2\alpha_\varepsilon-4}}{1 - 2^{-2\alpha_\varepsilon}} \frac{n}{L} + \frac{2\exp\left(2(w-1)\varepsilon^2\right)}{1 - \exp\left(-2\varepsilon^2\right)} L^{-2\varepsilon^2 \log e}\right)$$

for some positive real number ε satisfying $\varepsilon < Y_w - \delta$ and some positive real number α_ε depending on δ and ε.

In the previous result [83], time complexity is given as $O\left(n^2 L^2\right)$. However, by our recalculation, the time complexity is given as $O\left(n^3 L^2\right)$ which is different from the previous result [83]. This time complexity is larger than our result. Moreover, the previous result's failure probability is $O\left(n^2/L\right)$, which is larger than ours. Thus, our method is better than the previous one in both the time complexity and the failure probability.

In Theorem 3, the success probability converges to 1 when $L \to \infty$. It intuitively corresponds to the fact that the correct leaf is not pruned at a larger L. In particular, the value of L for which our method runs in polynomial time in n is given in Corollary 1.

Corollary 1 Let $L = n^{1+\gamma}$ $(\gamma > 0)$ in our proposed method. Then, as $n \to \infty$, the success probability of our method converges to 1, and the time complexity is given as $O\left(n^{3+\gamma} \log n\right)$, which is polynomial time in n.

To prove Theorem 3, we adopt assumptions similar to the original research [10] and more specific assumptions [13]. The given square-and-multiply sequences are generated from random bits. Moreover, it is assumed that calculated bits from incorrect leaves are random on **0** and **1** independently. In addition to these assumptions, assumptions on the calculated sequences based on the collision entropy H are adopted. These assumptions are used in the analysis of recovering the CRT-RSA secret keys from the correct square-and-multiply sequences. However, we consider errors in the analysis of our proposed method. Therefore, based on assumptions [10, 13], we use following assumptions where d'_p and d'_q are the incorrect value:

1. The given square-and-multiply sequences are generated from random bits.
2. The calculated bits of d_p and d_q from incorrect leaves are independent.
3. When $(i + \tau(k_p) + 1)$ LSBs of d'_p are the same with d_p and the next bit is different with d_p,
$$\Pr\left[D_{p,t+\tau(k_p)+1} \leq Y\right] \leq 2^{-(t-i)H}$$
with some constant $H \in [0, 1]$.
4. When $(i + \tau(k_q) + 1)$ LSBs of d'_q are the same with d_q and the next bit is different with d_q,
$$\Pr\left[D_{q,t+\tau(k_q)+1} \leq Y\right] \leq 2^{-(t-i)H}$$
with some constant $H \in [0, 1]$.

We now present our strategy for proving Theorem 3. First, we analyze the time complexity of our method in Section 4.4.1. Next, we analyze our method's failure probability briefly in Section 4.4.2, and we give the detailed proof in Section 4.4.3. Our method fails when there are more than L leaves whose distance is smaller than the correct leaf. To analyze the failure probability, we calculate the value of H. Especially, we obtain the condition of δ and Y satisfying $H > 1/2$ in Lemma 5, similar as the original result [10]. Then, we analyze the failure probability similar to Kunihiro-Honda's method [58].

4.4.1 Analysis of Time Complexity

We now evaluate the time complexity of our method. Our method comprises three steps: generating the root, generating the tree, and searching for the CRT-RSA secret keys. Two of these steps (generating the root and searching for the CRT-RSA secret keys) can be analyzed as [43] with small modifications. The time complexity of generating the root is $O((\log e)\, n)$. Moreover, searching for the CRT-RSA secret keys is $O(L)$. We now evaluate the time complexity of generating the tree.

First, we evaluate the time complexity of generating leaves at each depth. We will focus on one leaf. For one leaf, the simultaneous Eqs. (3.1)–(3.3) are solved in $O(n)$. We convert the calculated bits into square-and-multiply sequences and calculate the distance in $O(w)$ for each of the generated leaves. Thus, new leaves are generated from one leaf in $O(n)$. There are at most L leaves at each depth. Therefore, all leaves are generated in $O(nL)$.

Then, we evaluate the time complexity of pruning leaves at each depth. There are at most $2L$ leaves at each depth. $2L$ leaves are sorted in $O(L \log L)$, and L leaves are discarded in $O(L)$; therefore, leaves are pruned in $O(L \log L)$.

Thus, leaves are generated and pruned at each depth in $\max(O(nL), O(L \log L))$. Because leaves are generated until $n/2$-th depth, the time complexity of generating the tree is $\max\left(O\left(n^2 L\right), O(nL \log L)\right)$. In conclusion, the time complexity of our method is $\max\left(O\left(n^2 L\right), O(nL \log L)\right)$.

4.4.2 Evaluation of Failure Probability

We now evaluate the failure probability of the proposed method. Our analysis is divided into two parts:

1. Obtain the condition of δ and Y satisfying $H > 1/2$
2. Evaluate the failure probability in a similar way to [58]

We now otain the condition of δ and Y satisfying $H > 1/2$ by is given by proving Lemma 5.

Lemma 5 Let the value of Y_w be as in Table 4.1 and the value of δ be $\delta < Y_w$ for $w \leq 4$. We define x_t as the randomly chosen t bits and define C_{x_t} as the square-and-multiply sequence generated from x_t. Moreover, we define O as the given square-and-multiply sequence generated from random bits and has an added error rate of δ. We define $d_{C_{x_t},O}$ as the disagreement rate between C_{x_t} and the corresponding operations in O. Then, some positive real values ε and α_ε exist such that

$$\Pr\left[d_{C_{x_t},O} \leq \delta + \varepsilon\right] = 2^{-t(1/2+\alpha_\varepsilon)}.$$

From Lemma 5, we can estimate the number of leaves at depth t whose distance is less than $\delta + \varepsilon$. As given in the later discussion, the expected number of leaves is less than $w^2 2^{2w-2\alpha_\varepsilon-3} / \left(1 - 2^{-2\alpha_\varepsilon}\right)$, and this is almost constant because we consider only small w values, such as $w \leq 4$. Therefore, the rough success probability is

$$\left(1 - \frac{w^2 2^{2w-2\alpha_\varepsilon-3}}{L\left(1 - 2^{-2\alpha_\varepsilon}\right)}\right)^{n/2} \sim 1 - \frac{w^2 2^{2w-2\alpha_\varepsilon-4}}{1 - 2^{-2\alpha_\varepsilon}} \frac{n}{L}.$$

Thus, our method recover CRT-RSA secret keys with high probability when we use $L = n^{1+\gamma}$ ($\gamma > 0$) as Corollary 1. Now we give the sketch proof of Lemma 5, and the detailed proof is shown in Section 4.4.3.

The Sketch Proof of Lemma 5

First, we evaluate the probability p_Y of the calculated sequence having a disagreement rate of less than Y when compared with the given square-and-multiply sequence O. We define random variable l_{x_t} as the length of C_{x_t}. Then, we define $O_{l_{x_t}}$ as the corresponding operations in O. We also define random variable e_{x_t} as the number of errors found between $O_{l_{x_t}}$ and C_{x_t}. Moreover, we define random variable Z_{x_t} as $Z_{x_t} = l_{x_t} Y - e_{x_t}$. It should be noted that e_{x_t} depends on δ, and Z_{x_t} depends on δ and Y, but we omit this in their notation.

The probability p_Y of the calculated sequence having a disagreement rate less than Y is calculated as

$$p_Y = \Pr\left[\frac{e_{x_t}}{l_{x_t}} \leq Y\right] = \Pr\left[Z_{x_t} \geq 0\right]$$

Table 4.4: Values of H_s

w	1	2	3	4
$Y = \delta$	0.108	0.067	0.034	0.008
s	1.7	1.9	2.1	3.1
H_s	0.5001	0.5008	0.5002	0.5026

$$\leq \inf_{s>0} \Pr\left[\exp\left(sZ_{x_t}\right) \geq 1\right] \leq \inf_{s>0} E\left[\exp\left(sZ_{x_t}\right)\right]. \tag{4.2}$$

In particular, the final inequality obeys Markov's inequality.

Now, we can write $E\left[\exp\left(sZ_{x_t}\right)\right]$ as

$$E\left[\exp\left(sZ_{x_t}\right)\right] = \boldsymbol{v} M_s^t \left[11\ldots1\right]^T,$$

where $\boldsymbol{v}$ is some constant-valued vector, and T means the transpose. The construction of M_s is shown in Section 4.4.3.

We now calculate the value of $E\left[\exp\left(sZ_{x_t}\right)\right]$. We define J_s as the Jordan form of M_s. Then, a regular matrix Q exists such that $M_s = QJ_sQ^{-1}$. Thus, $M_s^t = QJ_s^tQ^{-1}$. Therefore,

$$\begin{aligned} E\left[\exp\left(sZ_{x_t}\right)\right] &= \boldsymbol{v} M_s^t \left[11\ldots1\right]^T \\ &= \left(\boldsymbol{v}Q\right) J_s^t \left(Q^{-1}\left[11\ldots1\right]^T\right). \end{aligned}$$

We define λ_{M_s} as the largest absolute value of the eigenvalues of M_s. Then, we have

$$\inf_{s>0} E\left[\exp\left(sZ_{x_t}\right)\right] \sim \left(\inf_{s>0} \lambda_{M_s}\right)^t.$$

We now go back to our original goal, evaluating the probability p_Y. From equation (4.2), it holds that $p_Y \leq \left(\inf_{s>0} \lambda_{M_s}\right)^t$, and we prove Lemma 5 by calculating λ_{M_s}.

However, we cannot analytically calculate the value of λ_{M_s}. Thus, we numerically calculate the value of λ_{M_s} using Matlab. We use the implemented function for calculating eigenvalues.

We now explain our numerical analysis method. First, we generate matrix M_s by setting the values of δ, Y, and s. From this matrix M_s, we calculate $H_s = -\log \lambda_{M_s}$.

We now show that if we set $\delta < Y_w$ and $Y = \delta$, then $H_s > 1/2$. First, we set $Y = \delta$ and set s as shown in Table 4.4. Thus, the value of H_s is given as in Table 4.4. For larger δ, we cannot find any s satisfying $H_s > 1/2$. We focus only on the bounds of δ, and we should consider smaller δ. We show these in Section 4.4.3.

We now prove Lemma 5. If $\delta < Y_w$ and $Y = \delta$, there is an s such that $H_s > 1/2$, as shown in Table 4.4. Moreover, because $E\left[\exp\left(sZ_{x_t}\right)\right]$ is a continuous function of Y, even if we take the larger Y, H_s is larger than $1/2$. Then, if we set $Y = \delta + \varepsilon < Y_w$ with a positive real-valued ε, there is a positive real-valued α_ε such that $H_s \geq 1/2 + \alpha_\varepsilon$. Therefore, when $\delta < Y_w$, there are positive real-valued ε such that $p_{\delta+\varepsilon} \leq 2^{-t(1/2+\alpha_\varepsilon)}$. This proves Lemma 5. □

We now evaluate the failure probability of the proposed method using Lemma 5, in a similar way to that in [58]. First, we evaluate P_t, which is the probability that a leaf containing correct information is discarded at the t-th depth of the candidate tree. The number of leaves generated is 2^t at the t-th depth. Then, we define X_1 as a leaf containing

correct information, and we define the other leaves as X_i $(2 \leq i \leq 2^t)$. Moreover, we define C_i as the distance of X_i from the given square-and-multiply sequences. Now,

$$\begin{aligned}
P_t &\leq \Pr\left[\sum_{i=2}^{2^t} \mathbf{1}\left[C_i \leq C_1\right] \geq L\right] \\
&\leq \Pr\left[\left(\sum_{i=2}^{2^t} \mathbf{1}\left[C_i \leq \delta+\varepsilon\right] \geq L\right) \cup \left(C_1 \geq \delta+\varepsilon\right)\right] \\
&\leq \Pr\left[\sum_{i=2}^{2^t} \mathbf{1}\left[C_i \leq \delta+\varepsilon\right] \geq L\right] + \Pr\left[C_1 \geq \delta+\varepsilon\right] \\
&\leq \frac{1}{L} E\left[\sum_{i=2}^{2^t} \mathbf{1}\left[C_i \leq \delta+\varepsilon\right]\right] + \Pr\left[C_1 \geq \delta+\varepsilon\right] \\
&\leq \frac{1}{L} \sum_{i=2}^{2^t} \Pr\left[C_i \leq \delta+\varepsilon\right] + \Pr\left[C_1 \geq \delta+\varepsilon\right]. \qquad (4.3)
\end{aligned}$$

We now calculate the upper bounds of these two terms.

First, let us focus on the first term of Eq. (4.3). When there are common $(i+\tau(k_p)+1)$ LSBs with the correct d_p and the next bit is different with d_p,

$$\Pr\left[\min_{0 \leq j \leq w-1} D_{p,\tau(t+k_p)+1-j} \leq \delta+\varepsilon\right] \leq w 2^{w-1} 2^{-(t-i)(1/2+\alpha_\varepsilon)}$$

because of Lemma 5 and our assumption. Similarly, when there are common $(i+\tau(k_q)+1)$ LSBs with the correct d_q and the next bit is different with d_q,

$$\Pr\left[\min_{0 \leq j \leq w-1} D_{q,\tau(t+k_q)+1-j} \leq \delta+\varepsilon\right] \leq w 2^{w-1} 2^{-(t-i)(1/2+\alpha_\varepsilon)}.$$

Now, there are 2^{t-i-1} candidates at depth t satisfying both of followings:

- There are common $(i+\tau(k_p)+1)$ LSBs with the correct d_p.
- There are common $(i+\tau(k_q)+1)$ LSBs with the correct d_q.

Thus,

$$\begin{aligned}
\frac{1}{L} \sum_{i=2}^{2^t} \Pr\left[C_i \leq \delta+\varepsilon\right] &\leq \frac{1}{L} \sum_{i=0}^{t-1} 2^{t-i-1} \left(w 2^{w-1} 2^{-(t-i)(1/2+\alpha_\varepsilon)}\right)^2 \\
&= \frac{w^2 2^{2w-3}}{L} \sum_{i=0}^{t-1} 2^{-2(t-i)\alpha_\varepsilon} \leq \frac{w^2 2^{2w-2\alpha_\varepsilon-3}}{L\left(1-2^{-2\alpha_\varepsilon}\right)} \qquad (4.4)
\end{aligned}$$

with some positive real value $\alpha_\varepsilon > 0$.

Next, we evaluate the second term. For this purpose, we introduce Hoeffding's theorem [46].

Theorem 4 [46] We define $X_1, \ldots, X_n$ as i.i.d. random variables with a Bernoulli distribution with parameter p. If we define random variable X as $X = \sum_{i=1}^{n} X_i$,

$$\Pr[X \geq n(p+\gamma)] \leq \exp\left(-2n\gamma^2\right).$$

The length of one square-and-multiply sequence corresponding to a leaf at the t-th depth is more than $t - w + 1$. Thus, in each square-and-multiply sequence, the probability that the disagreement rate is higher than $\delta + \varepsilon$ is at most $\exp\left(-2(t - w + 1)\varepsilon^2\right)$. Therefore,

$$\Pr\left[C_1 \geq \delta + \varepsilon\right] \leq 2\exp\left(-2(t - w + 1)\varepsilon^2\right). \tag{4.5}$$

We now evaluate Eq. (4.3) by combining Eqs. (4.4) and (4.5):

$$P_t \leq \frac{w^2 2^{2w-2\alpha_\varepsilon-3}}{L\left(1 - 2^{-2\alpha_\varepsilon}\right)} + 2\exp\left(-2(t - w + 1)\varepsilon^2\right).$$

Pruning is performed on $(\lfloor \log L \rfloor + 1) \leq t \leq n/2$. Thus, the failure probability P is

$$\begin{aligned}
P &= \sum_{t=\lfloor \log L \rfloor + 1}^{n/2} P_t \\
&\leq \sum_{t=\lfloor \log L \rfloor + 1}^{n/2} \left(\frac{w^2 2^{2w-2\alpha_\varepsilon-3}}{L\left(1 - 2^{-2\alpha_\varepsilon}\right)} + 2\exp\left(-2(t - w + 1)\varepsilon^2\right) \right) \\
&\leq \frac{w^2 2^{2w-2\alpha_\varepsilon-4}}{1 - 2^{-2\alpha_\varepsilon}} \frac{n}{L} + \frac{2\exp\left(2(w-1)\varepsilon^2\right)\exp\left(-2\left(\lfloor \log L \rfloor + 1\right)\varepsilon^2\right)}{1 - \exp\left(-2\varepsilon^2\right)} \\
&\leq \frac{w^2 2^{2w-2\alpha_\varepsilon-4}}{1 - 2^{-2\alpha_\varepsilon}} \frac{n}{L} + \frac{2\exp\left(2(w-1)\varepsilon^2\right)\exp\left(-2\varepsilon^2 \log L\right)}{1 - \exp\left(-2\varepsilon^2\right)} \\
&\leq \frac{w^2 2^{2w-2\alpha_\varepsilon-4}}{1 - 2^{-2\alpha_\varepsilon}} \frac{n}{L} + \frac{2\exp\left(2(w-1)\varepsilon^2\right)}{1 - \exp\left(-2\varepsilon^2\right)} L^{-2\varepsilon^2 \log e}.
\end{aligned}$$

In the above discussion, we give proof of the time complexity and failure probability of our method. Therefore, Theorem 3 is proven. □

4.4.3 The Remaining Parts of the Proof of Lemma 5

In the above proof of Lemma 5, we do not explain followings:

- The method for constructing the matrix M_s in proving Lemma 5
- The value of H_s in other parameters

Thus, we explain these.

The Method for Constructing the Matrix M_s in Proving Lemma 5

First, $E\left[\exp\left(sZ_{x_t}\right)\right]$ can be written as

$$E\left[\exp\left(sZ_{x_t}\right)\right] = \sum_{x_t \in \{0,1\}^t} \sum_{O_{l_{x_t}} \in \{\mathbf{S},\mathbf{M}\}^{l_{x_t}}} \left(\Pr\left[x_t\right] \Pr\left[O_{l_{x_t}}\right] \exp\left(sZ_{x_t}\right)\right), \tag{4.6}$$

and we transform Eq. (4.6).

We will now construct a recursive expression on the term in the summation: $\Pr\left[x_t\right] \Pr\left[O_{l_{x_t}}\right] \exp\left(sZ_{x_t}\right)$. To construct this recursive expression, we define Δx_t as the t-th bits of x_t and $\Delta O_{l_{x_t}}$ as the difference between $O_{l_{x_{t-1}}}$ and $O_{l_{x_t}}$. Moreover, we define Δl_{x_t} and Δe_{x_t} as $\Delta l_{x_t} = l_{x_t} - l_{x_{t-1}}$ and $\Delta e_{x_t} = e_{x_t} - e_{x_{t-1}}$. $\Delta O_{l_{x_t}}, \Delta l_{x_t}$, and Δe_{x_t} as the additional information based on the additional determined operations on the calculated square-and-multiply sequences. Then,

$$\Pr\left[x_t\right] \Pr\left[O_{l_{x_t}}\right] \exp\left(sZ_{x_t}\right)$$

$$
\begin{aligned}
&= \Pr[x_t] \Pr\left[O_{l_{x_t}}\right] \exp(sYl_{x_t}) \exp(-se_{x_t}) \\
&= \Pr[x_{t-1} \circ \Delta x_t] \Pr\left[O_{l_{x_{t-1}}} \circ \Delta O_{l_{x_t}}\right] \exp\left(sYl_{x_{t-1}}\right) \exp(sY\Delta l_{x_t}) \\
&\quad \exp\left(-se_{x_{t-1}}\right) \exp(-s\Delta e_{x_t}) \\
&= \Pr[x_{t-1}] \Pr\left[O_{l_{x_{t-1}}}\right] \exp\left(sZ_{x_{t-1}}\right) \Pr[\Delta x_t | x_{t-1}] \Pr\left[\Delta O_{l_{x_t}} | x_{t-1}\right] \\
&\quad \exp(sY\Delta l_{x_t}) \exp(-s\Delta e_{x_t}). \qquad (4.7)
\end{aligned}
$$

In the above equations, $\Pr[x_{t-1}] \Pr\left[O_{l_{x_{t-1}}}\right] \exp\left(sZ_{x_{t-1}}\right)$ can be regarded as a value calculated from x_{t-1} and $O_{l_{x_{t-1}}}$, and the remaining parts can be regarded as the changing term depending on x_{t-1}. Thus, we calculate this value by transitioning from the $(t-1)$-th bit to the t-th bit. To calculate this, we define the finite simultaneous states of x_t and $O_{l_{x_t}}$. Then, we can write a recursive expression for the value of $E[\exp(sZ_{x_t})]$ with a finite number of states. We can construct a Markov chain on the correct square-and-multiply sequence [14]. However, we cannot use their construction directly because it discards the total length of the square-and-multiply sequence. Therefore, we construct a different Markov chain on the square-and-multiply sequences.

First, let us consider the state of the calculated sequence. If we recover each bit, all possible states are $\mathbf{0}$, $\mathbf{1x}^i$ $(0 \le i \le w-1)$, where $\mathbf{x}$ are unknown bits. Thus, the number of all possible states is 2^w. $\mathbf{0}$ and $\mathbf{1x}^{w-1}$ can be converted into square-and-multiply sequences, and the rest are intermediate cases. Therefore, the renewal of the value of l_{x_t}, e_{x_t} occurs when we hit $\mathbf{0}$ and $\mathbf{1x}^{w-1}$. The transition probabilities are given as follows:

- **From $\mathbf{0}$**: $\Pr[\mathbf{0} \to \mathbf{0}] = \frac{1}{2}, \Pr[\mathbf{0} \to \mathbf{1}] = \frac{1}{2}$,
- **From $\mathbf{1x}^i$** $(0 \le i \le w-2)$: $\Pr\left[\mathbf{1x}^i \to \mathbf{1x}^i\mathbf{0}\right] = \frac{1}{2}$, $\Pr\left[\mathbf{1x}^i \to \mathbf{1x}^i\mathbf{1}\right] = \frac{1}{2}$,
- **From $\mathbf{1x}^{w-1}$**: $\Pr\left[\mathbf{1x}^{w-1} \to \mathbf{0}\right] = \frac{1}{2}, \Pr\left[\mathbf{1x}^{w-1} \to \mathbf{1}\right] = \frac{1}{2}$,

and the rest are zero.

Next, we calculate the transition probabilities of the given sequence without errors. For a window size of w, the following are all the possible states of the operations in the given sequence:

- $\mathbf{S}_0$: An $\mathbf{S}$ out of the window
- $\mathbf{S}_{\mathrm{bef},i}$ $(1 \le i \le w)$: The i-th $\mathbf{S}$ in the window and before $\mathbf{M}$
- $\mathbf{S}_{\mathrm{aft},i}$ $(2 \le i \le w)$: The i-th $\mathbf{S}$ in the window and after $\mathbf{M}$
- $\mathbf{M}_i$ $(1 \le i \le w)$: The i-th $\mathbf{M}$ in the window

The first three, $\mathbf{S}_0$, $\mathbf{S}_{\mathrm{bef},i}$ $(1 \le i \le w)$, and $\mathbf{S}_{\mathrm{aft},i}$ $(2 \le i \le w)$, are $\mathbf{S}$, and $\mathbf{M}_i$ $(1 \le i \le w)$ is $\mathbf{M}$ in the square-and-multiply sequences. Thus, the number of all possible states is

$$1 + w + (w-1) + w = 3w.$$

The transition probabilities are given as

- **From $\mathbf{S}_0$**: $\Pr[\mathbf{S}_0 \to \mathbf{S}_0] = \frac{1}{2}, \Pr[\mathbf{S}_0 \to \mathbf{S}_{\mathrm{bef},1}] = \frac{1}{2}$,
- **From $\mathbf{S}_{\mathrm{bef},1}$**: $\Pr[\mathbf{S}_{\mathrm{bef},1} \to \mathbf{M}_1] = \frac{1}{2^{w-1}}$, $\Pr[\mathbf{S}_{\mathrm{bef},1} \to \mathbf{S}_{\mathrm{bef},2}] = \frac{2^{w-1}-1}{2^{w-1}}$,
- **From $\mathbf{S}_{\mathrm{bef},i}$** $(2 \le i \le w)$:
  $\Pr[\mathbf{S}_{\mathrm{bef},i} \to \mathbf{M}_i] = \frac{2^{i-2}}{2^{w-1}-2^{i-2}}$, $\Pr[\mathbf{S}_{\mathrm{bef},i} \to \mathbf{S}_{\mathrm{bef},i+1}] = \frac{2^{w-1}-2^{i-1}}{2^{w-1}-2^{i-2}}$,
- **From $\mathbf{S}_{\mathrm{aft},i}$** $(2 \le i \le w-1)$: $\Pr[\mathbf{S}_{\mathrm{aft},i} \to \mathbf{S}_{\mathrm{aft},i+1}] = 1$,

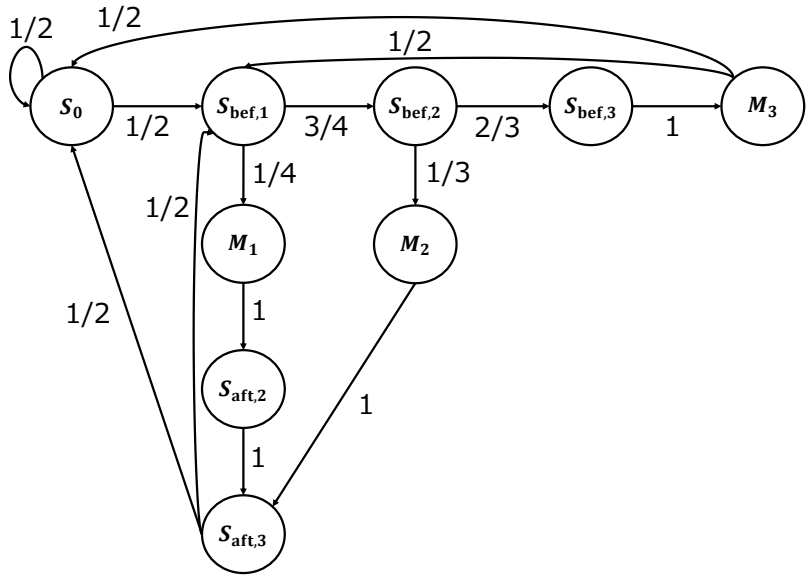

Fig. 4.1: Example of transitions when $w = 3$

- **From** $\mathbf{S}_{\text{aft},w}$: $\Pr[\mathbf{S}_{\text{aft},w} \to \mathbf{S}_0] = \frac{1}{2}, \Pr[\mathbf{S}_{\text{aft},w} \to \mathbf{S}_{\text{bef},1}] = \frac{1}{2}$,
- **From** $\mathbf{M}_i$ $(1 \le i \le w-1)$: $\Pr[\mathbf{M}_i \to \mathbf{S}_{\text{aft},i+1}] = 1$,
- **From** $\mathbf{M}_w$: $\Pr[\mathbf{M}_w \to \mathbf{S}_0] = \frac{1}{2}, \Pr[\mathbf{M}_w \to \mathbf{S}_{\text{bef},1}] = \frac{1}{2}$,

and the rest are zero. An example of the transitions when $w = 3$ is presented in Fig. 4.1.

We can simulate square-and-multiply sequences in the sliding window method by defining the transition probability as done above. We mainly focus on reading bits from the MSB side. First, when we hit a **0** bit, then $\Pr[\mathbf{S}_0] = \frac{1}{2}$, and this corresponds to the edges of $\mathbf{S}_0$ having a probability of $1/2$. Next, when we hit a **1** bit, then there are 2^{w-1} random window candidates, and there is one $\mathbf{M}$ in the square-and-multiply sequences. Then, the probabilities of the square-and-multiply sequences for each position of $\mathbf{M}$ are given as follows:

- $\Pr[\mathbf{M}_1] = \frac{1}{2^w}$.
- $\Pr[\mathbf{M}_i] = \frac{1}{2^{w+2-i}}$ $(2 \le i \le w)$.

We now trace our transition matrix:

$$\Pr[\mathbf{M}_1] = \Pr[\mathbf{S}_0 \to \mathbf{S}_{\text{bef},1}] \Pr[\mathbf{S}_{\text{bef},1} \to \mathbf{M}_1] \Pr[\mathbf{M}_1 \to \mathbf{S}_{\text{aft},2}] \prod_{j=2}^{w-1} \Pr[\mathbf{S}_{\text{aft},j} \to \mathbf{S}_{\text{aft},j+1}]$$
$$= \frac{1}{2} \frac{1}{2^{w-1}} = \frac{1}{2^w},$$

$$\Pr[\mathbf{M}_i] = \Pr[\mathbf{S}_0 \to \mathbf{S}_{\text{bef},1}] \left(\prod_{j=1}^{i-1} \Pr[\mathbf{S}_{\text{bef},j} \to \mathbf{S}_{\text{bef},j+1}] \right) \Pr[\mathbf{S}_{\text{bef},i} \to \mathbf{M}_i] \Pr[\mathbf{M}_i \to \mathbf{S}_{\text{aft},i+1}]$$

$$\left(\prod_{k=i+1}^{w-1} \Pr\left[\mathbf{S}_{\mathrm{aft},k} \rightarrow \mathbf{S}_{\mathrm{aft},k+1}\right]\right)$$
$$= \frac{1}{2}\frac{2^{w-1}-1}{2^{w-1}}\left(\prod_{j=2}^{i-1}\left(\frac{2^{w-1}-2^{j-1}}{2^{w-1}-2^{j-2}}\right)\right)\frac{2^{i-2}}{2^{w-1}-2^{i-2}} = \frac{1}{2^{w+2-i}} \quad (2 \leq i \leq w).$$

It should be noted that there is no term after $\Pr\left[\mathbf{S}_{\mathrm{bef},i} \rightarrow \mathbf{M}_i\right]$ when $i = w$. Therefore, our transition matrix simulates square-and-multiply sequences in the sliding window method.

These transition probabilities are calculated based on considering all 2^{w-1} patterns of square-and-multiply sequences whose lengths are $w+1$. $\mathbf{S}_0$ and $\mathbf{S}_{\mathrm{aft},w}$ have the following common properties:

- They are distinguished as $\mathbf{S}$ in the square-and-multiply sequences.
- Their distributions of the next operation are the same.
- Their window information is reset in the next operation.

Thus, we summarize them in one operation, $\mathbf{S}_0$. Therefore, the transition matrix of the given sequence without error is given as a square matrix of size $3w - \mathbf{1}(w \geq 2)$, where $\mathbf{1}(x)$ is an indicator function.

Using the above information, we can calculate M_s on all the possible pairs of the states in the given and calculated square-and-multiply sequences as $(S_{\mathrm{cal}}, S_{\mathrm{giv}})$. That is, we can calculate the transition probability between $(S_{\mathrm{cal}}, S_{\mathrm{giv}})$ before the transition and $\left(S'_{\mathrm{cal}}, S'_{\mathrm{giv}}\right)$ after the transition. It should be noted that the number of states in M_s is $2^w\,(3w - \mathbf{1}(w \geq 2))$, namely 6 when $w = 1$, 20 when $w = 2$, 64 when $w = 3$, and 176 when $w = 4$. Hereafter, we focus on the transition part in Eq. (4.7),

$$\Pr\left[\Delta x_t | x_{t-1}\right] \Pr\left[\Delta O_{l_{x_t}} | x_{t-1}\right] \exp\left(sY\Delta l_{x_t}\right) \exp\left(-s\Delta e_{x_t}\right),$$

based on the state of S'_{cal}, which is used for changing $\Delta O_{l_{x_t}}, \Delta l_{x_t}$, and Δe_{x_t}. First, $\Pr\left[\Delta x_t | x_{t-1}\right] = \Pr\left[S_{\mathrm{cal}} \rightarrow S'_{\mathrm{cal}}\right]$ on all states of S'_{cal}. Next, we consider the remaining parts. When S'_{cal} is not $\mathbf{0}$ or $\mathbf{1x}^{w-1}$, there is no new operation in the calculated square-and-multiply sequences. Therefore, $S_{\mathrm{giv}} = S'_{\mathrm{giv}}$, the values of Δl_{x_t} and Δe_{x_t} are zero, and

$$\Pr\left[\Delta O_{l_{x_t}} | x_{t-1}\right] \exp\left(sY\Delta l_{x_t}\right) \exp\left(-s\Delta e_{x_t}\right) = 1.$$

When S'_{cal} is $\mathbf{0}$ or $\mathbf{1x}^{w-1}$, there are new operations in the calculated square-and-multiply sequences, and the values of Δl_{x_t} and Δe_{x_t} are not zero. We now define $G_{x_t,j}$ and $G'_{x_t,j}$ as the $l_{x_{t-1}} + j \;\; (0 \leq j \leq \Delta l_{x_t})$-th operation in the given square-and-multiply sequence before and after considering errors. Similarly, we define $C_{x_t, l_{x_{t-1}}+j}$ as the $l_{x_{t-1}} + j \;\; (0 \leq j \leq \Delta l_{x_t})$-th operation in the calculated sequence. Then,

$$\Pr\left[\Delta O_{l_{x_t}} | x_{t-1}\right] \exp\left(sY\Delta l_{x_t}\right) \exp\left(-s\Delta e_{x_t}\right)$$
$$= \exp\left(sY\Delta l_{x_t}\right) \prod_{j=1}^{\Delta l_{x_t}} \left(\Pr\left[G_{l_{x_{t-1}}-1+j} \rightarrow G_{l_{x_{t-1}}+j}\right]\right.$$
$$\left.\sum_{G'_{l_{x_{t-1}}+j} = \{\mathbf{S},\mathbf{M}\}} \left(\Pr\left[G'_{l_{x_{t-1}}+j} | G_{l_{x_{t-1}}+j}\right] \exp\left(-s\mathbf{1}\left(G'_{l_{x_{t-1}}+j} \neq C_{x_t, l_{x_{t-1}}+j}\right)\right)\right)\right),$$

where $\mathbf{1}(x)$ is the indicator function, and the inner states of the summation are considered

only on $\{\mathbf{S}, \mathbf{M}\}$. The value of Δl_{x_t} is

$$\begin{cases} 1 & \text{if } S'_{\text{cal}} = \mathbf{0}, \\ w+1 & \text{if } S'_{\text{cal}} = \mathbf{1}\mathbf{x}^{w-1}. \end{cases}$$

Now,

$$\sum_{G'_{l_{x_{t-1}}+j}=\{\mathbf{S},\mathbf{M}\}} \left(\Pr\left[G'_{l_{x_{t-1}}+j} | G_{l_{x_{t-1}}+j}\right] \exp\left(-s\mathbf{1}\left(G'_{l_{x_{t-1}}+j} \neq C_{x_t, l_{x_{t-1}}+j}\right)\right)\right)$$

is

$$\begin{cases} 1-\delta+\delta\exp(-s) & G_{l_{x_{t-1}}+j} = C_{x_t, l_{x_{t-1}}+j}, \\ (1-\delta)\exp(-s)+\delta & G_{l_{x_{t-1}}+j} \neq C_{x_t, l_{x_{t-1}}+j}, \end{cases}$$

and we write these two values as $T = 1-\delta+\delta\exp(-s), F = (1-\delta)\exp(-s)+\delta$. Based on the above discussion, we calculate the matrix M_s.

We now present a simple example for $w = 1$. First, the transition probabilities of the calculated sequence are given as

- **From 0**, $\Pr[\mathbf{0} \to \mathbf{0}] = \frac{1}{2}, \Pr[\mathbf{0} \to \mathbf{1}] = \frac{1}{2}$,
- **From 1**, $\Pr[\mathbf{1} \to \mathbf{0}] = \frac{1}{2}, \Pr[\mathbf{1} \to \mathbf{1}] = \frac{1}{2}$,

and the rest are zero. Second, the transition probabilities of the given sequence are given as

- **From** $\mathbf{S}_0$, $\Pr[\mathbf{S}_0 \to \mathbf{S}_0] = \frac{1}{2}, \Pr[\mathbf{S}_0 \to \mathbf{S}_{\text{bef},1}] = \frac{1}{2}$,
- **From** $\mathbf{S}_{\text{bef},1}$, $\Pr[\mathbf{S}_{\text{bef},1} \to \mathbf{M}_1] = 1$,
- **From** $\mathbf{M}_1$, $\Pr[\mathbf{M}_1 \to \mathbf{S}_0] = \frac{1}{2}, \Pr[\mathbf{M}_1 \to \mathbf{S}_{\text{bef},1}] = \frac{1}{2}$,

and the rest are zero. We now consider 6 states, $(\mathbf{0}, \mathbf{S}_0)$, $(\mathbf{0}, \mathbf{S}_{\text{bef},1})$, $(\mathbf{0}, \mathbf{M}_1)$, $(\mathbf{1}, \mathbf{S}_0)$, $(\mathbf{1}, \mathbf{S}_{\text{bef},1})$, and $(\mathbf{1}, \mathbf{M}_1)$. If we consider these states in this order, the matrix M_s is given as

$$\begin{bmatrix} \frac{T}{4}\exp(sY) & \frac{T}{4}\exp(sY) & 0 & \frac{TF}{8}\exp(2sY) & \frac{TF}{8}\exp(2sY) & \frac{T^2}{4}\exp(2sY) \\ 0 & 0 & \frac{F}{2}\exp(sY) & \frac{F^2}{4}\exp(2sY) & \frac{F^2}{4}\exp(2sY) & 0 \\ \frac{T}{4}\exp(sY) & \frac{T}{4}\exp(sY) & 0 & \frac{TF}{8}\exp(2sY) & \frac{TF}{8}\exp(2sY) & \frac{T^2}{4}\exp(2sY) \\ \frac{T}{4}\exp(sY) & \frac{T}{4}\exp(sY) & 0 & \frac{TF}{8}\exp(2sY) & \frac{TF}{8}\exp(2sY) & \frac{T^2}{4}\exp(2sY) \\ 0 & 0 & \frac{F}{2}\exp(sY) & \frac{F^2}{4}\exp(2sY) & \frac{F^2}{4}\exp(2sY) & 0 \\ \frac{T}{4}\exp(sY) & \frac{T}{4}\exp(sY) & 0 & \frac{TF}{8}\exp(2sY) & \frac{TF}{8}\exp(2sY) & \frac{T^2}{4}\exp(2sY) \end{bmatrix}.$$

For example, let us explain the element at $(2, 5)$, which corresponds to the transition from $(\mathbf{0}, \mathbf{S}_{\text{bef},1})$ to $(\mathbf{1}, \mathbf{S}_{\text{bef},1})$. First,

$$\Pr[\Delta x_t | x_{t-1}] = \Pr[\mathbf{1}|\mathbf{0}] = \frac{1}{2},$$

and the calculated sequence is **SM**. The length of the calculated sequence is 2, and thus $\Delta l_{x_t} = 2$. Next, the only possible transition from $\mathbf{S}_{\text{bef},1}$ to $\mathbf{S}_{\text{bef},1}$ in two steps is

Table 4.5: H_s when $w = 1$ and $s = 1.7$

$Y = \delta$	0	0.02	0.04	0.06	0.08	0.1	0.108	0.109
H_s	0.8080	0.7511	0.6942	0.6373	0.5802	0.5231	0.5001	0.4973

Table 4.6: H_s when $w = 2$ and $s = 1.9$

$Y = \delta$	0	0.02	0.04	0.06	0.062	0.064	0.066	0.067	0.068
H_s	0.6845	0.6299	0.5751	0.5201	0.5146	0.5091	0.5036	0.5008	0.4981

Table 4.7: H_s when $w = 3$ and $s = 2.1$

$Y = \delta$	0	0.01	0.02	0.03	0.031	0.032	0.033	0.034	0.035
H_s	0.5942	0.5667	0.5390	0.5113	0.5085	0.5058	0.5030	0.5002	0.4974

Table 4.8: H_s when $w = 4$ and $s = 3.1$

$Y = \delta$	0	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009
H_s	0.5346	0.5306	0.5266	0.5226	0.5186	0.5146	0.5106	0.5066	0.5026	0.4985

$\mathbf{S}_{\mathrm{bef},1} \to \mathbf{M}_1 \to \mathbf{S}_{\mathrm{bef},1}$. Therefore, $\Pr\left[G_{l_{x_{t-1}}} \to G_{l_{x_{t-1}}+1}\right] = \Pr\left[\mathbf{S}_{\mathrm{bef},1} \to \mathbf{M}_1\right] = 1$, and $\Pr\left[G_{l_{x_{t-1}}+1} \to G_{l_{x_{t-1}}+2}\right] = \Pr\left[\mathbf{M}_1 \to \mathbf{S}_{\mathrm{bef},1}\right] = \dfrac{1}{2}$. Moreover, both operations are different between the given and calculated sequences. Therefore,

$$\begin{aligned}&\Pr\left[\Delta x_t | x_{t-1}\right] \Pr\left[\Delta O_{l_{x_t}} | x_{t-1}\right] \exp\left(sY\Delta l_{x_t}\right) \exp\left(-s\Delta e_{x_t}\right) \\ &= \frac{1}{2}\exp(2sY) \cdot 1 \cdot \frac{1}{2} \cdot F^2 = \frac{F^2}{4}\exp(2sY).\end{aligned}$$

The Value of H_s in Other Parameters

In the sketch proof, we only focused on the bounds of δ and Y that have an s satisfying $H_s > 1/2$. To prove Lemma 5, we set a smaller δ and Y. Tables 4.5–4.8 show the results.

Tables 4.5–4.8 describes that the value of H_s is a monotonic decreasing function of δ and Y. Moreover, the value of H_s decreases linearly with δ and Y. For example, in Table 4.5, when δ and Y increase by 0.02, the value of H_s decreases by approximately 0.0569. For larger δ and Y, when δ and Y increase by 0.002, the value of H_s decreases by approximately 0.0057, and this value is approximately 1/10 of 0.0569. This property is satisfied in the other values of w. From these results, the value of H_s is an almost linearly decreasing function of δ when $Y = \delta$ and s is fixed.

Remark 3 It should be noted that setting δ and Y as 0 corresponds to no errors, as in [10], $Z_{x_t} = -e_{x_t}$ and $E\left[\exp\left(sZ_{x_t}\right)\right]$ is a monotonic decreasing function of s. Because $P_0 \leq \inf_{s>0} E\left[\exp\left(sZ_{x_t}\right)\right]$, it follows that

$$P_0 \leq \lim_{s\to\infty} E\left[\exp\left(sZ_{x_t}\right)\right] \sim \lim_{s\to\infty} 2^{-tH_s},$$

and this equation is similar to that in [10]. If we set s as a sufficiently large number, such as 10000, the values of H_s are the same as those shown in Table 4.2.

Table 4.9: Experimental results for $w = 1$

	δ	0	0.02	0.04	0.06	0.08	0.09	0.1	0.11	0.12	0.13
$n = L = 1024$	Success Rate (%)	100	99	88	59	31	19	7	3	1	0
	Time (s)	8.76	9.73	8.11	8.63	9.24	10.7	9.13	15.4	10.8	–
$n = L = 2048$	Success Rate (%)	100	100	77	35	10	6	0			
	Time (s)	49.7	49.8	53.2	54.2	54.0	66.8	–			

4.5 Numerical Experiments

This section shows the results of numerical experiments performed on the proposed method. In this experiment, we try to recover the CRT-RSA secret keys from square-and-multiply sequences with the error rate δ values shown in Table 4.1. We performed these numerical experiments using the NTL library 11.3.2 on C++. Moreover, we assumed that we know the values of k_p and k_q, and the actual time may be 2^{15} times as long as these experimental results.

We ran the proposed method on the 1024-bit CRT-RSA and the 2048-bit CRT-RSA, corresponding to $n = 1024$ and 2048, respectively. For each n, we set parameters (w, δ, L). For $n = 1024$, we set $L = 2^{10}$ from $L = n^{1+\gamma}$ $(\gamma > 0)$ based on Corollary 1. Similarly, for $n = 2048$, we set $L = 2^{11}$. Moreover, we set (w, δ). For each (n, w, δ, L), we generated 100 CRT-RSA secret keys and measured the success rate and average implementation times for each successful trial. Tables 4.9–4.12 show the results.

These results describe that our proposed method recovers the CRT-RSA secret keys when δ is less than that shown in Table 4.1. Moreover, a few CRT-RSA secret keys are recovered when δ is slightly more than that shown in Table 4.1. Therefore, our analysis matches these experimental results. However, when $w = 4$, our method recovered 29% of the CRT-RSA secret keys for $n = 1024$ and 9% for $n = 2048$ when $\delta = 0.01$, although the theoretical bound is $\delta = 0.008$. Thus, we should perform a more rigorous analysis of this theoretical bound.

By comparing the results obtained for $n = 1024$ and $n = 2048$, we found that the implementation time was approximately 5 or 6 times longer for $n = 2048$ than for $n = 1024$. The obtained implementation times matched our analysis because the time complexity is $O\left(n^3\right)$ when $L = n$.

Next, we consider the actual errors $\delta = 0.011$ reported in [10]. We now discuss the results obtained when using an error rate of $\delta = 0.011$. Tables 4.9–4.11 show that our method will recover the CRT-RSA secret keys with higher probability when $w = 1, 2$, and 3. Moreover, Tables 4.12 shows that our method will recover a few CRT-RSA secret keys when $w = 4$. Our method will also recover the CRT-RSA secret keys for smaller L values when $w = 1$ and 2. Thus, when $w = 1$ and 2, we search for the smallest L whose success rate is 100% in the range of $L = 2^{11}$. In addition, we performed experiments for $w = 3$ and 4 with $(n, L) = \left(1024, 2^{10}\right)$ and $\left(2048, 2^{11}\right)$. Table 4.13 shows the experimental results.

Table 4.13 describes that our method recovers almost all CRT-RSA secret keys when $w = 1$ and 2. When $w = 3$, our method recovers more than 50% of the CRT-RSA secret keys. Our analysis concluded that our method could recover the CRT-RSA secret keys with a high probability for these values of w. These results match our analysis. Moreover, our method recovers a few CRT-RSA secret keys when $w = 4$. Our method does not guarantee to recover the CRT-RSA secret keys when $w = 4$ and $\delta = 0.011$ but recovers the CRT-RSA secret keys using these parameters. Therefore, our method works on actual errors.

Table 4.10: Experimental results for $w = 2$

	δ	0	0.02	0.04	0.06	0.07	0.08	0.09
$n = L = 1024$	Success Rate (%)	100	89	45	16	6	1	0
	Time (s)	10.9	11.2	10.1	9.85	18.5	8.06	–
$n = L = 2048$	Success Rate (%)	100	73	26	2	1	0	
	Time (s)	58.8	61.6	55.3	62.8	46.3	–	

Table 4.11: Experimental results for $w = 3$

	δ	0	0.01	0.02	0.03	0.035	0.04	0.045	0.05
$n = L = 1024$	Success Rate (%)	100	84	42	23	15	7	7	0
	Time (s)	9.23	9.10	10.0	9.54	19.1	13.0	17.8	–
$n = L = 2048$	Success Rate (%)	100	61	25	3	1	1	0	
	Time (s)	62.6	59.4	60.5	57.8	55.9	54.6	–	

Table 4.12: Experimental results for $w = 4$

	δ	0	0.005	0.01	0.015	0.02	0.025	0.03	0.035
$n = L = 1024$	Success Rate (%)	100	58	29	7	5	1	2	0
	Time (s)	11.5	11.3	11.8	9.92	15.6	8.11	8.10	–
$n = L = 2048$	Success Rate (%)	100	41	9	1	1	0		
	Time (s)	71.2	66.0	54.2	55.4	58.5	–		

Table 4.13: Experimental results for $\delta = 0.011$

w	1	1	2	2	3	3	4	4
n	1024	2048	1024	2048	1024	2048	1024	2048
L	2^5	2^8	2^{10}	2^{11}	2^{10}	2^{11}	2^{10}	2^{11}
Success Rate (%)	100	100	100	96	72	59	18	5
Time (s)	0.359	7.02	10.7	65.0	11.5	51.2	14.9	56.9

4.6 Conclusion and Future Work

We discussed how to recover the CRT-RSA secret keys from square-and-multiply sequences with errors in the left-to-right sliding window method. First, we proposed a method for recovering the CRT-RSA secret keys from square-and-multiply sequences with errors. Second, we analyzed our method and calculated the upper bounds of the error rates δ in the square-and-multiply sequences, as in Table 4.1. Finally, we performed numerical experiments using the proposed method.

In the future, we should give a more rigorous analysis when $w = 4$. Our method does not guarantee to recover the CRT-RSA secret keys when $w = 4$ and $\delta = 0.011$, but our method recovered the CRT-RSA secret keys. We should then give a more rigorous analysis of the recovery when $w = 4$ in more detail.

In theory, the CRT-RSA secret keys are not recovered even if there is no error in square-and-multiply sequences when $w = 5$, and our method does not work when $w = 5$ even if $\delta = 0$. However, in practice, the CRT-RSA secret keys may be recovered as the previous studies [10] and the content of Chapter 3. Thus, we should consider methods for recovering the CRT-RSA secret keys from noisy square-and-multiply sequences when $w = 5$.

Chapter 5

Exact Security Analysis of the Ring-LWE and Module-LWE Based Schemes against NTT Leakage

5.1 Introduction

5.1.1 Background

This chapter focuses on the security of Ring-LWE [66] and Module-LWE [12] based schemes against side-channel attacks on a Number Theoretic Transform (NTT). We focus on the Lyubashevsky et al.'s cryptosystem (LPR cryptosystem) [67], which is the most basic scheme of both Ring-LWE and Module-LWE based schemes. We evaluate the LPR cryptosystem against NTT leakage.

In previous research, Primas et al. [95] proposed a method for extracting the information from addition, subtraction, and multiplication in an NTT. The authors proposed a method for recovering the secret keys by using an iterative method, namely the loopy belief propagation. Moreover, they evaluated security through numerical experiments.

However, the convergence of loopy belief propagation is not guaranteed, mentioned by Primas et al. The loopy belief propagation is employed to calculate the marginal distribution of secret polynomial. Unfortunately, the loopy belief propagation is not guaranteed to calculate the accurate marginal distribution. Thus, we should provide the exact security analysis to clarify the threats of side-channel attacks on NTTs.

5.1.2 Our Contribution

In this chapter, we analyze the exact security of the LPR cryptosystem [67] against side-channel attacks on an NTT. We focus on the LPR cryptosystem using the Cooley-Tukey NTT and the Gentleman-Sande INTT. To enable us to evaluate the exact security, we adopt the erasure model on the multiplication. We assume that the input of the integer multiplication is extracted with probability δ. Based on this leakage model, we evaluate the exact security of the LPR cryptosystem.

First, we propose an algorithm for recovering the LPR secret keys via recovering secret polynomial partially. We mainly focus on how to recover secret polynomial partially. We then analyze our proposed algorithm based on the number of recovered coefficients of the secret polynomial. We propose a method for calculating the number of recovered coefficients. To calculate the number of recovered coefficients, we adopt an iterative method, which always converges. Moreover, we analyze our algorithm for recovering the

full LPR secret keys. We show that our method recovers the secret keys when $\delta \leq 0.78$ at the current computational power. Finally, we perform numerical experiments, and our method recovers the secret keys when $\delta \leq 0.78$. We then present a novel solution for analyzing the threats to the NTTs.

5.1.3 Organization of this Chapter

This chapter has six sections. In Section 5.2, we provide the preliminaries used in this chapter. In Section 5.3, we describe our leakage model in NTTs and propose an algorithm for recovering coefficients from the leakage information. In Section 5.4, we give an exact analysis of the recovery rate of coefficients of a secret polynomial. In Section 5.5, we propose a method for recovering the full LPR secret keys. In Section 5.6, we conclude this chapter.

5.2 Preliminaries

In this section, we provide the notations in NTTs described in Section 5.2.1. We then summarize previous research on recovering LPR secret keys [95] in Section 5.2.2.

5.2.1 Notations in NTTs

We now focus on the values in the following two aspects:

- **Global values**: $N(\log N + 1)$ values in NTTs
- **Local values**: four values in each butterfly

Here, we introduce notations of the global and local values. Significantly, the Cooley-Tukey NTT and Gentleman-Sande INTT have the same structure, and we deal with these NTTs simultaneously in later sections. We define the global values and local values by considering the structure of these NTTs.

We first define $M_{i,j}$ as the global values. In the Cooley-Tukey NTT, we define $M_{0,k}$ $(0 \leq k \leq N-1)$ as the inputs $a[k]$. Moreover, we define $M_{i+1,k}$ as $a[k]$ after the operations of the index i in Algorithm 2, where $0 \leq i \leq n-1$ and $0 \leq k \leq N-1$. Using $M_{i,k}$, the inputs and outputs of $(N \log N)/2$ butterfly operations are represented as follows:

- **Input**: $M_{i,k}$ and $M_{i,k+2^{n-1-i}}$
- **Output**: $M_{i+1,k}$ and $M_{i+1,k+2^{n-1-i}}$

In the above representation, $\left\lfloor k/2^{n-1-i} \right\rfloor$ is even. We then renew $a[k]$ from $M_{i,k}$ to $M_{i+1,k}$ and $a[k+t]$ from $M_{i,k+2^{n-1-i}}$ to $M_{i+1,k+2^{n-1-i}}$. Similar to the Cooley-Tukey NTT, we define the global values in the Gentleman-Sande INTT. In the Gentleman-Sande INTT, we define $M_{\log N,k}$ $(0 \leq k \leq N-1)$ as the inputs $a[k]$. Moreover, we define $M_{i,k}$ as $a[k]$ after the operations of the index i in Algorithm 3, where $0 \leq i \leq n-1$ and $0 \leq k \leq N-1$. Using $M_{i,k}$, we can represent the inputs and outputs of $(N \log N)/2$ butterfly operations as follows:

- **Input**: $M_{i+1,k}$ and $M_{i+1,k+2^{n-1-i}}$
- **Output**: $M_{i,k}$ and $M_{i,k+2^{n-1-i}}$

In the above representation, $\left\lfloor k/2^{n-1-i} \right\rfloor$ is even. We then renew $a[k]$ from $M_{i+1,k}$ to $M_{i,k}$ and $a[k+t]$ from $M_{i+1,k+2^{n-1-i}}$ to $M_{i,k+2^{n-1-i}}$.

Next, we define m_i $(1 \leq i \leq 4)$ as the local values. In the following discussion, we write

the multiplier in each butterfly as λ. In the Cooley-Tukey NTT, we renew $a[k]$ and $a[k+t]$ as

$$a[k] = a[k] + \lambda a[k+t] \bmod q$$
$$a[k+t] = a[k] - \lambda a[k+t] \bmod q$$

in each butterfly. We then define m_1 and m_2 as the $a[k]$ and $a[k+t]$ before renewal, respectively. Moreover, we define m_3 and m_4 as $a[k]$ and $a[k+t]$ after renewal, respectively. Similar to the Cooley-Tukey NTT, we define the local values in the Gentleman-Sande INTT. In the Gentleman-Sande INTT, we renew $a[k]$ and $a[k+t]$ as

$$a[k] = a[k] + a[k+t] \bmod q$$
$$a[k+t] = \lambda\left(a[k] - a[k+t]\right) \bmod q$$

in each butterfly. Then, we define m_1 and m_2 as $a[k]$ and $a[k+t]$ after renewal, respectively. Moreover, we define m_3 and m_4 as $a[k]$ and $a[k+t]$ before renewal, respectively.

In the above discussion, we defined the global and local values. Then, in both of the Cooley-Tukey NTT and the Gentleman-Sande INTT, each butterfly is constructed by four values $M_{i,k}$, $M_{i,k+2^{n-1-i}}$, $M_{i+1,k}$, and $M_{i+1,k+2^{n-1-i}}$ in the global values. Moreover, these values correspond to m_1, m_2, m_3, and m_4, respectively. Then, to define the global index on each butterfly, we define set I as

$$I := \left\{(i,k) \,|\, (0 \le i \le \log N - 1) \cap \left(\lfloor k/2^{n-1-i} \rfloor \text{ is even.}\right)\right\},$$

which is the set of indexes of $M_{i,k}$ corresponding to m_1.

5.2.2 Previous Side-Channel Attacks on an NTT [95]

Primas et al. [95] attack an LPR cryptosystem [67] using an NTT with $N = 256$, $q = 7681$, and $\sigma = 4.51$. Primas et al. extract side-channel leakage from

$$c_1 s + c_2 = \mathrm{INTT}\left(\mathrm{NTT}\left(c_1\right) \cdot \mathrm{NTT}(s) + \mathrm{NTT}\left(c_2\right)\right)$$

in the decryption step. This INTT is calculated similarly as a Cooley-Tukey NTT using different values as multipliers. Significantly, the Cooley-Tukey NTT merges the power of ω and γ, whereas the NTT in Primas et al.'s study only considers the power of ω and considers γ separately. Moreover, the INTT in Primas et al.'s study is realized using $\omega^{-1} \bmod q$ instead of ω and has the same structure as the Cooley-Tukey NTT. Thus, by considering the Cooley-Tukey NTT, we also deal with the INTT in Primas et al.'s study.

Their algorithm recovers the LPR secret keys in the following order:

1. Extracting information from the INTT
2. Recovering $M_{i,k}$'s using loopy belief propagation
3. Recovering the LPR secret key s through a lattice reduction algorithm

The second step is a heuristic procedure owing to the use of the loopy belief propagation. We analyze this part exactly in our contribution. We now review the three steps in more detail.

Extracting Information from the INTT

We can extract information from $(N \log N)/2$ butterflies through template attacks [16]. The information is extracted from additions, subtractions, and multiplications of two integers.

First, we explain the information of additions and subtractions. We now define r_{ADD} and r_{SUB} as follows:

$$r_{\mathrm{ADD}} = \begin{cases} 1 & m_1 + \lambda m_2 \geq q \\ 0 & \text{Otherwise} \end{cases}, r_{\mathrm{SUB}} = \begin{cases} 1 & m_1 - \lambda m_2 < 0 \\ 0 & \text{Otherwise} \end{cases}.$$

We then extract the value of r_{ADD} and r_{SUB} as the information of an addition and subtraction. The information on the additions and subtractions is mostly extracted correctly. Thus, Primas et al. assumed the correctness of this information.

The multiplication information is extracted from a calculation of $\lambda m_2 \bmod q$. We then extract the probability distribution of m_2 depending on the values of m_2 and $\lambda m_2 \bmod q$. Primas et al. gave a simpler model, i.e., the noisy Hamming weight leakage model. In this model, attackers extract the Hamming weight of the input and output of multiplication with noise. To describe this model, we denote $\mathrm{HW}(x)$ as the Hamming weight function inputting x. Moreover, we denote $\mathcal{N}\left(0, \sigma_l^2\right)$ as a Gaussian distribution with a mean of 0 and standard deviation of $\sigma_l > 0$. We extract the information as $\mathrm{HW}\left(m_2\right) + e$ and $\mathrm{HW}\left(\lambda m_2 \bmod q\right) + e$, where $e \sim \mathcal{N}\left(0, \sigma_l^2\right)$. We then extract the probability distribution of m_2.

Recovery of Values of $M_{i,k}$'s

We now describe the recovery of the $M_{i,k}$ values. We define the probability functions in each butterfly based on the information. We then recover $M_{i,k}$ by maximizing the product of all probability functions.

First, we define the probability functions in each butterfly. We define $f_{\mathrm{ADD}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_3\right)$, $f_{\mathrm{SUB}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_4\right)$, and $f_{\mathrm{MUL}}\left(\widetilde{m}_2\right)$ as the probability functions representing the addition, subtraction, and multiplication information, respectively. In addition, $f_{\mathrm{ADD}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_3\right)$ takes a 0 or 1. Moreover, $f_{\mathrm{ADD}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_3\right) = 1$ if and only if

- $\widetilde{m}_1 + \lambda \widetilde{m}_2 \equiv \widetilde{m}_3 \bmod q$, and
- $0 \leq \widetilde{m}_1 + \left(\lambda \widetilde{m}_2 \bmod q\right) - q r_{\mathrm{ADD}} < q$.

Similarly, $f_{\mathrm{SUB}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_4\right)$ takes a 0 or 1. In addition, $f_{\mathrm{SUB}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_4\right) = 1$ if and only if

- $\widetilde{m}_1 - \lambda \widetilde{m}_2 \equiv \widetilde{m}_4 \bmod q$, and
- $0 \leq \widetilde{m}_1 - \left(\lambda \widetilde{m}_2 \bmod q\right) + q r_{\mathrm{SUB}} < q$.

Finally, $f_{\mathrm{MUL}}\left(\widetilde{m}_2\right)$ is defined as $f_{\mathrm{MUL}}\left(\widetilde{m}_2\right) = \Pr\left[m_2 = \widetilde{m}_2\right]$.

We recover $M_{i,k}$ by using the probability functions $f_{\mathrm{ADD}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_3\right)$, $f_{\mathrm{SUB}}\left(\widetilde{m}_1, \widetilde{m}_2, \widetilde{m}_4\right)$, and $f_{\mathrm{MUL}}\left(\widetilde{m}_2\right)$. We define Z as the product of all probability functions on all $N\left(\log N + 1\right)$ values in the NTT, namely,

$$\begin{aligned} Z = \prod_{(i,k) \in I} \Big(& f_{\mathrm{ADD}}\left(\widetilde{M}_{i,k}, \widetilde{M}_{i,k+2^{n-1-i}}, \widetilde{M}_{i+1,k}\right) \\ & \times f_{\mathrm{SUB}}\left(\widetilde{M}_{i,k}, \widetilde{M}_{i,k+2^{n-1-i}}, \widetilde{M}_{i+1,k+2^{n-1-i}}\right) \times f_{\mathrm{MUL}}\left(\widetilde{M}_{i,k+2^{n-1-i}}\right)\Big) \end{aligned}$$

We then calculate the marginal distribution $Z_{i,k}$ of value $M_{i,k}$ through a loopy belief propagation. Next, we search the low entropy $M_{i,k}$, such as the peak in a single value, and recover such information. Primas et al. do not apply loopy belief propagation to all $M_{i,k}$ and obtain a lower entropy of $M_{i,k}$. Primas et al.'s algorithm recovers 192 values on $M_{5,k}$ $(32 \leq k \leq 127$ and $160 \leq k \leq 255)$.

Recovering the Secret Key s using Lattice Reduction Algorithm

Next, the secret key s is recovered. In this description, polynomial $a \in \mathcal{R}_q$ is represented as the vector $\boldsymbol{a} = [a[0], a[1], \ldots, a[255]]^T \in \mathbb{Z}_q^{256}$, where $a[i]$ is the coefficient of x^i in a.

In the previous step, 192 values on $M_{5,k}$ are recovered, and $\boldsymbol{r} \in \mathbb{Z}_q^{192}$ is defined as the vector containing these values. We now express $\boldsymbol{r}$ using $\boldsymbol{s}$. From Algorithm 2, $\boldsymbol{r}$ has 192 values out of $a[k]$s after $i = 4$, and we obtain $c_1 s + c_2$ when we run $i = 5$, 6, and 7. The secret value is only s in $c_1 s + c_2$. Now, $c_1 s + c_2$ can be written as $C_1 \boldsymbol{s} + \boldsymbol{c}_2$ in the vector, where C_1 is the matrix in $\mathbb{Z}_q^{256\times 256}$. Each element of C_1 is given as follows:

$$C_{1,j,k} = \begin{cases} c_1[k-j] & \text{if } k-j \geq 0 \\ -c_1[k-j+256] & \text{Otherwise,} \end{cases}$$

and the matrix C_1 is known. Moreover, the inverse transformations of $i = 5$, 6, and 7 are linear mappings. Thus, when we focus only on the 192 values of $M_{5,k}$, we can represent $\boldsymbol{r}$ as $\boldsymbol{r} = U_1 \boldsymbol{s} + \boldsymbol{u_2}$ using the matrix $U_1 \in \mathbb{Z}_q^{192\times 256}$ and the vector $\boldsymbol{u_2} \in \mathbb{Z}_q^{192}$. Thus, $\boldsymbol{r} - \boldsymbol{u_2} = U_1 \boldsymbol{s}$, and only $\boldsymbol{s}$ is unknown. Moreover, we set $b = e - as$ in the key generation, and this equation is rewritten as $\boldsymbol{b} = \boldsymbol{e} - A\boldsymbol{s}$ using the matrix $A \in \mathbb{Z}_q^{256\times 256}$, the elements of which are given as

$$A_{j,k} = \begin{cases} a[k-j] & \text{if } k-j \geq 0 \\ -a[k-j+256] & \text{Otherwise,} \end{cases}$$

and the vector is $\boldsymbol{b}, \boldsymbol{e} \in \mathbb{Z}_q^{256}$. The unknown vectors are $\boldsymbol{s}$ and $\boldsymbol{e}$. By substituting $\boldsymbol{r} - \boldsymbol{u_2} = U_1 \boldsymbol{s}$ into $\boldsymbol{b} = \boldsymbol{e} - A\boldsymbol{s}$ we obtain $\boldsymbol{b'} = \boldsymbol{e} - A'\boldsymbol{s'}$, where $\boldsymbol{s'}$ has $64 (= 256 - 192)$ dimensions. If we obtain $\boldsymbol{s'}$, we recover $\boldsymbol{s}$ and the secret key s. Now, the norm of $\boldsymbol{e}$ is small. Thus, Primas et al. used the BKZ algorithm [4] to calculate this short vector $\boldsymbol{e}$ and recover the LPR secret keys s.

Experimental Results of the Previous Method

By using the above method, we can recover all secret keys from the real trace. Moreover, using the Hamming weight noisy model, we can recover the secret keys when $\sigma_l \leq 0.5$. Thus, the previous method works on the LPR cryptosystem. This attack is extended to CRYSTALS-KYBER [90].

Primas et al.'s method partially recover coefficients of the secret polynomial, and the full LPR secret keys are recovered by using a lattice reduction algorithm. Their method recovers 192 coefficients of the secret polynomial in the above discussion before applying a lattice reduction algorithm. Primas et al. also experimented on cases in which fewer values are recovered. From their experiments, the secret keys are practically recovered when 160 coefficients are recovered. Their method also recovers the secret keys when 150 coefficients are recovered, but it takes a long time. Their method does recover the secret keys when 146 or fewer coefficients are recovered. Thus, the LPR secret keys are recovered when more than or equal to 150 coefficients are recovered before applying a lattice reduction algorithm.

5.3 Our Proposed Method for Recovering Secret Polynomial from NTT leakage

In this section, we propose our method for recovering the secret polynomial transformed by the NTT. This recovery corresponds to recovering $M_{i,k}$'s in Primas et al.'s method [95].

We consider a simpler model than Primas et al.'s approach and propose a new method for recovering the secret polynomial based on our model. First, we describe our leakage model in Section 5.3.1. Next, we propose two recovery methods in Sections 5.3.2 and 5.3.3, respectively. The method proposed in Section 5.3.2 is more straightforward than the other, whereas the method proposed in Section 5.3.3 may be more efficient than the other. For each method, we show the results of the numerical experiments.

5.3.1 Our Leakage Model

In this study, we assume that the input of multiplication is extracted with probability $1-\delta$. If $\delta = 0$, we extract all candidate values are extracted, and if $\delta = 1$, we extract no information. We focus only on multiplication and do not consider addition or subtraction. We now consider the leakage model in Cooley-Tukey NTT and Gentleman-Sande INTT in more detail.

Our Leakage Model in the Cooley-Tukey NTT

In Cooley-Tukey NTT, we extract m_2 in each butterfly with probability $1-\delta$. When we consider the entire NTT, we extract $M_{i,k+2^{n-i-1}}$, where $(i,k) \in I$ with probability $1-\delta$. We do not extract any other information.

Our Leakage Model in the Gentleman-Sande INTT

In the Gentleman-Sande INTT, we can extract m_2 because we renew m_2 by $m_2 = \lambda(m_3 - m_4) \bmod q$. Moreover, we can extract $M_{0,k}$'s from the final multiplication of $N^{-1} \bmod q$. Thus, we extract the information in the following manner:

- We extract $M_{0,k}$ $(N/2 \leq k \leq N-1)$ twice from the butterfly and the final multiplication of $N^{-1} \bmod q$.
- We extract $M_{0,k}$ $(0 \leq k \leq N/2-1)$ once from the final multiplication of $N^{-1} \bmod q$.
- We extract $M_{i,k+2^{n-i-1}}$ $((i,k) \in I$ and $i \neq 0)$ once from the butterfly.
- We do not extract any information in the other variables.

Now, we fail to extract the input of multiplication with probability δ. When we obtain information twice, we fail to extract information with probability δ^2. When we obtain information once, we fail to extract information with probability δ. Therefore, we extract the information in the following manner:

- We extract $M_{0,k}$ $(N/2 \leq k \leq N-1)$ with probability $1-\delta^2$.
- We extract $M_{0,k}$ $(0 \leq k \leq N/2-1)$ with probability $1-\delta$.
- We extract $M_{i,k+2^{n-i-1}}$ $((i,k) \in I$ and $i \neq 0)$ with probability $1-\delta$.
- We do not extract any information in the other variables.

Figure 5.1 shows the example of leakage in Gentleman-Sande INTT for $N = 4$.

5.3.2 Recovering Coefficients of the Secret Polynomial

We now propose the recovery algorithm on the Cooley-Tukey NTT and Gentleman-Sande INTT. Our recovery algorithm is given in Algorithm 6. Now, we briefly describe our recovery algorithm.

In our recovery algorithm, we input the extracted m_2 and parameters on the NTT. Moreover, we input the index of the NTT, namely, Inv, as 0 in the Cooley-Tukey NTT and 1 in the Gentleman-Sande INTT. We then output the inputs of the Cooley-Tukey NTT or the outputs of the Gentleman-Sande INTT, which correspond to $M_{0,k}$ $(0 \leq k \leq N-1)$.

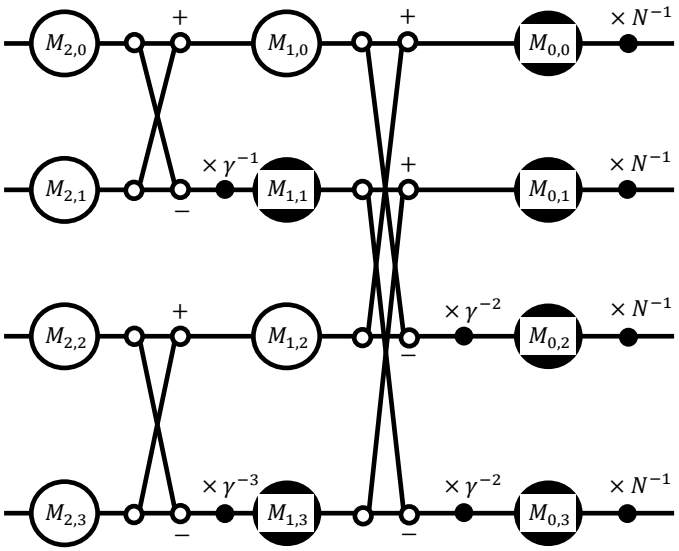

Fig. 5.1: Our leakage model in Gentleman-Sande INTT when $N = 4$. The inputs of all of the multiplications leak out. Thus, $M_{0,0}$, $M_{0,1}$, $M_{1,1}$, and $M_{1,3}$ are extracted with probability $1-\delta$. Moreover, $M_{0,2}$ and $M_{0,3}$ are extracted with probability $1-\delta^2$ because these values are extracted twice. In Cooley-Tukey NTT, multiplications using N^{-1} are not performed, and information is not leaked from these multiplications N^{-1}.

Significantly, the information is the same in both the Cooley-Tukey NTT and Gentleman-Sande INTT. Thus, we can apply our recovery algorithm to both the Cooley-Tukey NTT and Gentleman-Sande INTT.

To recover $M_{0,k}$, we propagate all butterfly information. Specifically, in each butterfly, we calculate the local values m_1, m_2, m_3, and m_4 from the following:

$$\begin{cases} m_1 + \lambda m_2 \equiv 2^{\mathrm{Inv}} m_3 \bmod q \\ m_1 - \lambda m_2 \equiv 2^{\mathrm{Inv}} m_4 \bmod q \end{cases} . \tag{5.1}$$

From Eq. (5.1), we calculate all values if we know two out of four values. We then apply the new calculated values to the other butterflies. We repeat the calculation of $M_{i,k}$ until we can no longer calculate a new $M_{i,k}$.

Our algorithm terminates when no value is recovered. Thus, at least 1 value is recovered in each iteration. In our algorithm, there are at most $N(\log N + 1)$ values. If we conduct at most $N(\log N+1)+1$ iterations, all values are recovered, and our algorithm terminates. Therefore, our algorithms terminate within at most $N(\log N + 1) + 1$ iterations.

Results of Numerical Experiments

We now show the results of numerical experiments conducted on the proposed method. We applied these numerical experiments using the NTL library 11.3.2 on C++.

We ran the proposed method on the parameters from Göttert et al.'s study [38]. In particular, we used $(N, q) = (256, 7681)$ and $(512, 12289)$, and set the erasure rate δ. For each (N, q, δ), we randomly generated 100 polynomials and measured the average of the following:

- Running time

Algorithm 6 Our algorithm for recovering the polynomial on the NTT

Input: N, q, γ, Inv, and extracted m_2
Output: Recovered $M_{0,k}$'s, which correspond to the coefficients of the non-NTT form of the secret polynomial
$C = 1$
While $C > 0$
  $C = 0$
  $e = 1$, $t = N/2$
  for $i = 0$ **to** $\log N - 1$
    $s = 0$
    for $j = 0$ **to** $e - 1$
      Calculate e', the bit-reversed value of $e + j$
      for $k = s$ **to** $s + t - 1$
        Calculation of Butterfly: We calculate the values using the following:

$$\begin{cases} M_{i,k} + \gamma^{(-1)^{\mathrm{Inv}} e'} M_{i,k+2^{n-1-i}} \equiv 2^{\mathrm{Inv}} M_{i+1,k} \bmod q \\ M_{i,k} - \gamma^{(-1)^{\mathrm{Inv}} e'} M_{i,k+2^{n-1-i}} \equiv 2^{\mathrm{Inv}} M_{i+1,k+2^{n-1-i}} \bmod q \end{cases}$$

        The number of calculated values is added to C
      end for
      $s = s + 2t$
    end for
    $e = 2e$, $t = t/2$
  end for
end while
Return Recovered $M_{0,k}$

- The number of iterations
- The number of recovered $M_{0,k}$'s

Tables 5.1 and 5.2 show the experimental result of the number of recovered $M_{0,k}$'s in the Cooley-Tukey NTT and Gentleman-Sande INTT, respectively.

Tables 5.1 and 5.2 describe that the number of recoveries of $M_{0,k}$ decreases when δ increases. In particular, the number of recoveries of $M_{0,k}$ decreases by approximately $N/2$ from $\delta = 0.7$ to 0.8 in the Cooley-Tukey NTT. Moreover, the number of recoveries of $M_{0,k}$ decreases by approximately $N/2$ from $\delta = 0.75$ to 0.8 in the Gentleman-Sande INTT. The number of iteration is small compared to the worst case for both the Cooley-Tukey NTT and the Gentleman-Sande INTT.

5.3.3 Variant Algorithm of Recovery of Coefficients of the Secret Polynomial

This subsection presents a variant algorithm for the recovery of the coefficients, which is given as Algorithm 7. In this algorithm, we focus on a butterfly in two directions, i.e., a forward recovery step and a backward recovery step, as indicated in Algorithm 7. By recovering in two directions, we may propagate the recovered values more efficiently. Now, we regard the pair of the forward and backward recovery steps as a single iteration. We repeat the iterations until there is no renewal in the recovery result.

Algorithm 7 Variants of our algorithm for recovering a secret polynomial for the NTT

Input: N, q, γ, Inv, and extracted m_2
Output: Recovered $M_{0,k}$'s, which correspond to the coefficients of the non-NTT form of a secret polynomial
$C = 1$
While $C > 0$
  $C = 0$
  Forward recovery step
  $e = 1$, $t = N/2$
  for $i = 0$ **to** $\log N - 1$
    $s = 0$
    for $j = 0$ **to** $e - 1$
      Calculate e', the bit-reversed value of $e + j$
      for $k = s$ **to** $s + t - 1$
        Calculation in Butterfly: We calculate the values using the following:

$$\begin{cases} M_{i,k} + \gamma^{(-1)^{\mathrm{Inv}} e'} M_{i,k+2^{n-1-i}} \equiv 2^{\mathrm{Inv}} M_{i+1,k} \bmod q \\ M_{i,k} - \gamma^{(-1)^{\mathrm{Inv}} e'} M_{i,k+2^{n-1-i}} \equiv 2^{\mathrm{Inv}} M_{i+1,k+2^{n-1-i}} \bmod q \end{cases}$$

        The number of calculated values is added to C
      end for
      $s = s + 2t$
    end for
    $e = 2e$, $t = t/2$
  end for
  Backward recovery step
  $e = N/2$, $t = 1$
  for $i = \log N - 1$ **down to** 0
    $s = 0$
    for $j = 0$ **to** $e - 1$
      Calculate e', the bit-reversed value of $e + j$
      for $k = s$ **to** $s + t - 1$
        Calculation of butterfly: We calculate the values using the following:

$$\begin{cases} M_{i,k} + \gamma^{(-1)^{\mathrm{Inv}} e'} M_{i,k+2^{n-1-i}} \equiv 2^{\mathrm{Inv}} M_{i+1,k} \bmod q \\ M_{i,k} - \gamma^{(-1)^{\mathrm{Inv}} e'} M_{i,k+2^{n-1-i}} \equiv 2^{\mathrm{Inv}} M_{i+1,k+2^{n-1-i}} \bmod q \end{cases}$$

        The number of calculated values is added to C
      end for
      $s = s + 2t$
    end for
    $e = e/2$, $t = 2t$
  end for
end while
Return Recovered $M_{0,k}$

Table 5.1: Experimental result on recovering the coefficients of the secret polynomial on the Cooley-Tukey NTT using Algorithm 6. "count" means the number of recovered $M_{0,k}$'s.

N / δ	256			512		
	time [ms]	#(iterations)	count	time [ms]	#(iterations)	count
0	18.7	8	255	36.7	9	511
0.2	16.9	8	254.5	22.8	9	510.6
0.4	17.0	8.04	252.3	26.4	9	509.3
0.6	19.9	11.7	237.9	50.7	11.8	495.1
0.7	32.8	19.0	172.7	55.4	21.3	424.8
0.8	16.2	7.89	45.3	38.6	10.9	89.8
0.9	6.55	3.48	16.7	16.5	4.06	31.7
1	1.42	1	0	3.90	1	0

Table 5.2: Experimental results on recovering the coefficients of the secret polynomial on the Gentleman-Sande INTT using Algorithm 6. "count" means the number of recovered $M_{0,k}$'s.

N / δ	256			512		
	time [ms]	#(iterations)	count	time [ms]	#(iterations)	count
0	7.44	2	256	13.8	2	512
0.2	14.3	6.61	255.7	22.5	7.60	511.8
0.4	16.2	7.51	255.1	26.9	8.56	511.0
0.6	15.0	9.08	251.6	42.5	9.63	507.6
0.7	28.5	14.8	236.2	39.7	15.3	491.8
0.75	37.3	19.7	192.1	90.0	25.8	438.3
0.8	18.2	10.9	109.6	57.7	16.0	234.0
0.9	7.26	3.79	43.0	18.5	4.50	85.8
1	2.32	1	0	4.08	1	0

Results of Numerical Experiments

We now describe the results of numerical experiments conducted on the variant of our proposed method. We ran the variant of our proposed method similar to that described in Section 5.3.2. Tables 5.3 and 5.4 show the experimental results for the number of recoveries of the secret polynomial in the Cooley-Tukey NTT and Gentleman-Sande INTT, respectively. One iteration in Tables 5.3 and 5.4 corresponds to two iterations in Tables 5.1 and 5.2. Thus, we include "2×#(iterations)" in Tables 5.3 and 5.4 for comparison with Tables 5.1 and 5.2.

The number of recovered coefficients and the running time is almost the same as those in Tables 5.1 and 5.2. However, the number of iterations is different between Tables 5.1 and 5.3. Moreover, the number of iterations is also different between Tables 5.2 and 5.4. We explain these in more detail.

First, we compare the results of the Cooley-Tukey NTT in Tables 5.1 and 5.3. Tables 5.1 and 5.3 show that the number of iterations of Algorithm 6 is larger when $\delta \leq 0.7$. In particular, the number of iterations in Table 5.3 is approximately 3/4 times that in Table 5.1 when $\delta \leq 0.7$. The number of iterations of Algorithm 7 is almost the same as the number of iterations of Algorithm 6 when $\delta \geq 0.8$. These results are as follows: When $\delta \leq 0.7$, our method recovers many $M_{i,k}$'s from the small number of initially recovered

Table 5.3: Experimental results on recovering the coefficients of the secret polynomial on the Cooley-Tukey NTT using Algorithm 7. "count" means the number of recovered $M_{0,k}$'s. "2×#(iterations)" corresponds to one iteration in Table 5.1.

N / δ	256				512			
	time [ms]	#(iterations)	2×#(iterations)	count	time [ms]	#(iterations)	2×#(iterations)	count
0	7.31	2	4	255	20.0	2	4	511
0.2	13.9	3	6	254.7	26.1	3	6	510.5
0.4	14.2	3.11	6.22	252.6	19.4	3.17	6.34	508.6
0.6	19.9	4.81	9.62	238.6	32.8	4.86	9.72	495.7
0.7	29.4	6.82	13.64	175.9	40.2	7.47	14.94	420.8
0.8	15.8	4.21	8.42	43.6	44.1	6.42	12.84	114.9
0.9	8.42	2.49	4.98	16.1	23.9	2.73	5.46	32.9
1	7.63	1	2	0	6.15	1	2	0

Table 5.4: Experimental results on recovering the coefficients of the secret polynomial on the Gentleman-Sande INTT using Algorithm 7. "count" means the number of recovered $M_{0,k}$'s. "2×#(iterations)" corresponds to one iteration in Table 5.2.

N / δ	256				512			
	time [ms]	iterations	2×#(iterations)	count	time [ms]	iterations	2×#(iterations)	count
0	9.41	2	4	256	24.2	2	4	512
0.2	10.5	2.57	5.14	255.7	26.9	2.69	5.38	511.7
0.4	13.8	3.01	6.02	255.3	18.9	3.01	6.02	511.1
0.6	16.3	4.06	8.12	251.5	27.6	4.05	8.10	507.6
0.7	25.3	5.73	11.46	236.2	33.6	5.95	11.90	489.8
0.75	23.6	7.43	14.86	193.6	59.8	8.97	17.94	438.7
0.8	23.8	5.66	11.32	108.3	67.4	9.82	19.64	311.9
0.9	8.85	2.61	5.22	43.6	26.8	2.91	5.82	84.5
1	2.35	1	2	0	6.27	1	2	0

$M_{i,k}$'s. Thus, our method heavily propagates the recovery values of $M_{i,k}$'s. Therefore, two-direction propagation realizes faster convergence, and fewer iterations are required in Algorithm 7. When $\delta \geq 0.8$, our method recovers few $M_{i,k}$'s, and incurs a small propagation. Thus, when $\delta \geq 0.8$, the number of iterations of Algorithm 7 is almost the same as the number of iterations of Algorithm 6.

Next, we compare the results of the Gentleman-Sande INTT in Tables 5.2 and 5.4. Tables 5.2 and 5.4 show that the numbers of iterations of Algorithm 6 are almost the same except when $\delta = 0.7$ and 0.75. In the Gentleman-Sande INTT, we obtain some values of $M_{0,k}$'s before applying our method, which causes a small difference in Tables 5.2 and 5.4 except when $\delta = 0.7$ and 0.75. When $\delta = 0.7$ and 0.75, we obtain a fewer number of $M_{0,k}$'s before applying our method, which heavily propagates the recovery values of the $M_{i,k}$'s. Thus, our method incurs more propagations when $\delta = 0.7$ and 0.75.

5.3.4 Connection to Recovering LPR Secret Keys

We now focus on the number of non-recovered values in the Gentleman-Sande INTT. When $\delta = 0.75$, our algorithm does not recover about 64 values when $N = 256$ and 74 values when $N = 512$. Based on Primas et al.'s experiment results, $106(= 256 - 150)$ non-recovered values are recovered using a lattice reduction algorithm. Therefore, the LPR secret keys may be recovered when at least $\delta \leq 0.75$. In the later sections, we give the bound of δ in more detail.

5.4 Analysis of Recovery Rate of Coefficients of the Secret Polynomial

We now analyze the recovery rate of coefficients using our proposed Algorithms 6 and 7. We analyze our proposed algorithm by renewing the probability of recovery. The difference in the analysis of Algorithms 6 and 7 is the method for constructing the renewal equations of the probability of recovery. First, we describe the notations used in this analysis in Section 5.4.1. Next, we provide an overview of our analysis in Section 5.4.2, followed by an explanation of the initial probabilities, which are common in the analysis of Algorithms 6 and 7 in Section 5.4.3. We then give an analysis of Algorithms 6 and 7 in Section 5.4.4 and 5.4.5, respectively.

5.4.1 Notations used in Our Analysis

First, we describe the variables representing the probabilities used in our analysis. Our analysis propagates the probabilities between butterflies by applying the global values $M_{i,k}$. However, in the following explanation, we mainly focus on the renewal equations of the local values m_i. Thus, we define the notations of the probabilities based on both the global values $M_{i,k}$ and the local values m_i.

We now define the probabilities of the state of recovery in each butterfly, $B_{i,k}^{\boldsymbol{l}}$ or $b^{\boldsymbol{l}}$, which correspond to the global values of $M_{i,k}$ or the local values of m_i, respectively. The indexes i and k in $B_{i,k}^{\boldsymbol{l}}$ are the global indexes of a butterfly as the elements applied in set I. The index $\boldsymbol{l}$ shows the state of the recovery of the butterfly, and we explain the state of recovery of a butterfly in more detail to define $B_{i,k}^{\boldsymbol{l}}$ or $b^{\boldsymbol{l}}$. Each butterfly is composed of four values, and each has two states, non-recovered or recovered. Thus, the number of states of a butterfly is $2^4 = 16$. Next, we define the index $\boldsymbol{l}$ by four bits. In this definition, we count bits from the left to the right side. Then, if the j-th bit of $\boldsymbol{l}$ is 1, the local value m_j is recovered. For example, if $\boldsymbol{l} = 0010$, we have already recovered m_3, and the other values are not recovered.

Next, we define the probabilities of recovery in each value, $P_{i,k}^{(l)}$ or $p_i^{(l)}$, which correspond to the global values of $M_{i,k}$ or the local values of m_i, respectively. The indexes i and k in $P_{i,k}^{(l)}$ are the indexes of the global values of $M_{i,k}$. The index l means the state of recovery, and we describe the state of recovery in more detail to define $P_{i,k}^{(l)}$ or $p_i^{(l)}$. We define three states for each value as follows:

- **State 1**: $M_{i,k}$ is not recovered.
- **State 2**: $M_{i,k}$ is newly recovered and not propagated in another butterfly.
- **State 3**: $M_{i,k}$ has already been recovered and used in another butterfly.

Here, l corresponds to the index of states 1, 2, or 3. In particular, **State 2** is a temporary state. Moreover, each value $M_{i,k}$ changes in a one-way direction as **State 1** $\rightarrow$ **State 2** $\rightarrow$ **State 3**.

In the following discussion, we call $B_{i,k}^{\boldsymbol{l}}$ and $P_{i,k}^{(l)}$ the global variables, and we call $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ the local variables.

5.4.2 Overview of Our Analysis

In our analysis, we renew probabilities $B_{i,k}^{\boldsymbol{l}}$ and $P_{i,k}^{(l)}$ based on Algorithm 6. From the above discussion, the transition of the state is one-way. Thus, $B_{i,k}^{\boldsymbol{l}}$ and $P_{i,k}^{(l)}$ converges to a certain value. We calculate the convergence values of $B_{i,k}^{\boldsymbol{l}}$ and $P_{i,k}^{(l)}$. To calculate the convergence values, we calculate the total amount of change in $P_{i,k}^{(l)}$ during each iteration. Specifically, we calculate $\sum_{i=0}^{\log N}\sum_{k=0}^{N-1}\sum_{l=1}^{3}\left|P_{i,k}^{(l)\prime}-P_{i,k}^{(l)}\right|$, where $P_{i,k}^{(l)\prime}$ is $P_{i,k}^{(l)}$ after the iteration. If this value is smaller than the threshold, let $10^{-7}N\left(\log N+1\right)$ in this thesis, we stop our analysis and output the sum of $P_{0,k}^{(3)}$ as the recovery rate. We perform these calculations using Matlab.

5.4.3 Initial Values

We now give the initial values in the analysis. We focus on the global variables in this subsection.

In the Cooley-Tukey NTT, we set the following:

$$P_{i,k}^{(l)}=\begin{cases}1 & \text{if } 0\le i\le \log N-1, \lfloor k/2^{n-1-i}\rfloor \text{ is even, and } l=1\\ \delta & \text{if } 0\le i\le \log N-1, \lfloor k/2^{n-1-i}\rfloor \text{ is odd, and } l=1\\ 1-\delta & \text{if } 0\le i\le \log N-1, \lfloor k/2^{n-1-i}\rfloor \text{ is odd, and } l=2\\ 0 & \text{Otherwise.}\end{cases}$$

The first line corresponds to the local value m_1, where we do not extract information. The second and third lines correspond to the local value m_2.

In the Gentleman-Sande INTT, we set the following:

$$P_{i,k}^{(l)}=\begin{cases}\delta & \text{if } i=0, 0\le k\le N/2-1 \text{ and } l=1\\ 1-\delta & \text{if } i=0, 0\le k\le N/2-1 \text{ and } l=2\\ \delta^2 & \text{if } i=0, N/2\le k\le N-1 \text{ and } l=1\\ 1-\delta^2 & \text{if } i=0, N/2\le k\le N-1 \text{ and } l=2\\ 1 & \text{if } 1\le i\le \log N-1, \lfloor k/2^{n-1-i}\rfloor \text{ is even, and } l=1\\ \delta & \text{if } 1\le i\le \log N-1, \lfloor k/2^{n-1-i}\rfloor \text{ is odd, and } l=1\\ 1-\delta & \text{if } 1\le i\le \log N-1, \lfloor k/2^{n-1-i}\rfloor \text{ is odd, and } l=2\\ 0 & \text{Otherwise.}\end{cases}$$

The first to fourth lines correspond to the global value $M_{0,k}$, where we extract more information than the other values. The remaining lines are the same as the Cooley-Tukey NTT.

5.4.4 Analysis of Algorithm 6

We explain the renewal probabilities equations in our analysis based on Algorithm 6. Specifically, when we refer to a butterfly in Algorithm 6, we renew the global variables included in the butterfly. We then realize the propagation of the probability.

Equations of Renewal Probabilities

We now focus on the renewal equation in each butterfly by using the local variables, $b^{\boldsymbol{l}}$ and $p_i^{(l)}$. First, we give a brief strategy of the renewal of $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ in each butterfly. We construct the equations of the renewal probabilities based on the transition of states. The state transition occurs by the following:

- The propagation of the recovery in other butterflies
- The recovery from Eq. (5.1)

Before the recovery using Eq. (5.1), the following are satisfied in m_i based on the propagation of the recovery in other butterflies:

- m_i is not recovered with probability $p_i^{(1)}$. (**State 1** $\rightarrow$ **State 1**)
- m_i is newly recovered with probability $p_i^{(2)}$. (**State 1** $\rightarrow$ **State 2**)
- m_i has been recovered with the probability $p_i^{(3)}$. (**State 3** $\rightarrow$ **State 3**)

Then, for m_i in **State 1**, the followings occur in the current butterfly:

- m_i is not recovered with the probability $p_i^{(1)} / \left(p_i^{(1)} + p_i^{(2)}\right)$.
- m_i is newly recovered with the probability $p_i^{(2)} / \left(p_i^{(1)} + p_i^{(2)}\right)$.

These are propagations from other butterflies.

Based on these propagations, we construct the renewal equation of $b^{\boldsymbol{l}}$ and $p_i^{(l)}$. Specifically, we renew $b^{\boldsymbol{l}}$ by considering the change in state of butterfly $\boldsymbol{l}$. For example, if $\boldsymbol{l} = \mathbf{0010}$, we have already recovered m_3. Then, we have the following:

- If we do not recover any values, namely, m_1, m_2, and m_4 are all in **State 1**, the state of the butterfly is still **0010**.
- Otherwise, we recover all values, and the state of the butterfly changes into **1111**.

Therefore, we apply the following:

- We add $\dfrac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \dfrac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \dfrac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} b^{\mathbf{0010}}$ to $b^{\mathbf{0010}}$.
- We add $\left(1 - \dfrac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \dfrac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \dfrac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right) b^{\mathbf{0010}}$ to $b^{\mathbf{1111}}$.

It should be noted that $b^{\mathbf{1100}}$, $b^{\mathbf{1010}}$, $b^{\mathbf{0110}}$, $b^{\mathbf{1110}}$, $b^{\mathbf{1001}}$, $b^{\mathbf{0101}}$, $b^{\mathbf{1101}}$, $b^{\mathbf{0011}}$, $b^{\mathbf{1011}}$, and $b^{\mathbf{0111}}$ are 0. Moreover, we renew $p_i^{(l)}$. In the above example, m_3 has already been recovered and used, namely, in **State 3**, and we add $b^{\mathbf{0010}}$ to $p_3^{(3)}$. In the other m_i's, we calculate the transition before and after the renewal. We now have the following:

- If we do not recover any values, m_1, m_2, and m_4 are all in **State 1**.
- Otherwise,
  - **State 1** $\rightarrow$ **State 1** $\rightarrow$ **State 2** or
  - **State 1** $\rightarrow$ **State 2** $\rightarrow$ **State 3**,

  where the first transition is the propagation of the recovery in other butterflies, and the second transition is recovered using Eq. (5.1).

We then apply the following:

- We add $\dfrac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \dfrac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \dfrac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} b^{\mathbf{0010}}$ to $p_i^{(1)}$, where $i = 1, 2$, and 4.

- We add $\left(\dfrac{p_i^{(1)}}{p_i^{(1)}+p_i^{(2)}} - \dfrac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\dfrac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\dfrac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right) b^{\mathbf{0010}}$ to $p_i^{(2)}$, where $i = 1, 2$, and 4.
- We add $\dfrac{p_i^{(2)}}{p_i^{(1)}+p_i^{(2)}} b^{\mathbf{0010}}$ to $p_i^{(3)}$, where $i = 1, 2$, and 4.

We propagate $p_i^{(l)}$, namely, $P_{i,k}^{(l)}$ in the global variables, to the other butterflies.

We now give the renewal equation of $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ in more detail. The renewal equation is different in the first iteration and the other iterations. Before the first iteration, we give the initial values only for $p_i^{(l)}$'s. Thus, using $p_i^{(l)}$'s, we calculate the values of $b^{\boldsymbol{l}}$. Specifically, we renew $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ by

$$
\begin{aligned}
b^{\mathbf{0000}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{1000}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{0100}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{0010}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{0001}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{1111}} &= 1 - \left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
&\quad \left. + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_1^{(1)} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad \left. + \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_1^{(2)} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad \left. + \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_1^{(3)} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\\
p_2^{(1)} &= \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad \left. + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)
\end{aligned}
$$

$$
\begin{aligned}
p_2^{(2)} &= \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_2^{(3)} &= \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\\
p_3^{(1)} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_3^{(2)} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_4^{(1)} &= \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\right)\\
p_4^{(2)} &= \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\right)
\end{aligned}
$$

and the other values are all 0.

In the other iterations, we renew $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ by

$$
\begin{aligned}
b^{\mathbf{0000}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}}\\
b^{\mathbf{1000}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}}+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{1000}}\\
b^{\mathbf{0100}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0100}}\\
b^{\mathbf{0010}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}}\\
b^{\mathbf{0001}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}}\\
b^{\mathbf{1111}} &= \left(1-\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\right.\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
&\quad\left.\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\right)b^{\mathbf{0000}}\\
&\quad+\left(1-\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{1000}}+\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0100}}
\end{aligned}
$$

$$
\begin{aligned}
&+\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0010}}+\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\right)b^{\mathbf{0001}}\\
p_1^{(1)} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0100}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}}\\
p_1^{(2)} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(1-\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0100}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(1-\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0010}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(1-\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\right)b^{\mathbf{0001}}\\
p_1^{(3)} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\left(b^{\mathbf{0000}}+b^{\mathbf{0100}}+b^{\mathbf{0010}}+b^{\mathbf{0001}}\right)+b^{\mathbf{1000}}+b^{\mathbf{1111}}\\
p_2^{(1)} &= \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}}\\
&\quad+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{1000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}}\\
p_2^{(2)} &= \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}}\\
&\quad+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(1-\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{1000}}+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0010}}\\
&\quad+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\right)b^{\mathbf{0001}}\\
p_2^{(3)} &= \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\left(b^{\mathbf{0000}}+b^{\mathbf{1000}}+b^{\mathbf{0010}}+b^{\mathbf{0001}}\right)+b^{\mathbf{0100}}+b^{\mathbf{1111}}\\
p_3^{(1)} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}}\\
&\quad+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{1000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0100}}
\end{aligned}
$$

$$
\begin{aligned}
&+ \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} b^{\mathbf{0001}} \\
p_3^{(2)} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}} \right. \\
&\quad \left. + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}} \right) b^{\mathbf{0000}} \\
&\quad + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \left(1 - \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \right) b^{\mathbf{1000}} + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \left(1 - \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \right) b^{\mathbf{0100}} \\
&\quad + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \left(1 - \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \right) b^{\mathbf{0001}} \\
p_3^{(3)} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}} \left(b^{\mathbf{0000}} + b^{\mathbf{1000}} + b^{\mathbf{0100}} + b^{\mathbf{0001}} \right) + b^{\mathbf{0010}} + b^{\mathbf{1111}} \\
p_4^{(1)} &= \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \right. \\
&\quad \left. + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}} \right) b^{\mathbf{0000}} \\
&\quad + \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} b^{\mathbf{1000}} + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} b^{\mathbf{0100}} \\
&\quad + \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} b^{\mathbf{0010}} \\
p_4^{(2)} &= \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}} + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}} \right. \\
&\quad \left. + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} + \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}} \right) b^{\mathbf{0000}} \\
&\quad + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \left(1 - \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \right) b^{\mathbf{1000}} + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \left(1 - \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \right) b^{\mathbf{0100}} \\
&\quad + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}} \left(1 - \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \right) b^{\mathbf{0010}} \\
p_4^{(3)} &= \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}} \left(b^{\mathbf{0000}} + b^{\mathbf{1000}} + b^{\mathbf{0100}} + b^{\mathbf{0010}} \right) + b^{\mathbf{0001}} + b^{\mathbf{1111}}
\end{aligned}
$$

and the other values are all 0. After an iteration, we apply an additional renewal as follows in $0 \leq i \leq N-1$:

$$
\begin{aligned}
P^{(2)}_{\log N,i} &= 0, \\
P^{(3)}_{\log N,i} &= P^{(2)}_{\log N,i} + P^{(3)}_{\log N,i}.
\end{aligned}
$$

Result of Theoretical Analysis of Algorithm 6

This subsection shows a theoretical analysis of the recovery rate $M_{0,k}$. First, we show the results of our analysis. Tables 5.5 and 5.6 gives the theoretical recovery rate of $M_{0,k}$. From these tables, the recovery rate decreases suddenly. The value of δ at which the recovery rate decreases is between $\delta = 0.7$ and 0.8. Thus, our analysis matches the experimental results, although the recovery rate changes more gradually in the experimental results.

We now connect our theoretical results to previous research [95]. In our analysis, $N = 256$ and $q = 7681$ for the Cooley-Tukey NTT are the same as those in the previous study. Primas et al. calculated the average entropy in each σ_l. The success rate of Primas et al.'s

Table 5.5: Theoretical value of average recovery rate of coefficients of the secret polynomial for the Cooley-Tukey NTT

N / δ	256		512		1024		2048	
	count	ratio	count	ratio	count	ratio	count	ratio
0	255	0.9961	511	0.9980	1023	0.9990	2047	0.9995
0.2	254.6	0.9943	510.6	0.9972	1022.6	0.9986	2046.6	0.9993
0.4	252.9	0.9879	508.9	0.9939	1020.9	0.9970	2044.9	0.9985
0.6	243.9	0.9526	499.9	0.9763	1011.9	0.9882	2035.9	0.9941
0.7	216.5	0.8456	472.5	0.9228	984.5	0.9614	2008.5	0.9807
0.71	210.9	0.8240	466.9	0.9120	978.9	0.9560	2002.9	0.9780
0.72	206.2	0.8056	462.2	0.9027	974.2	0.9514	1998.2	0.9757
0.73	200.9	0.7847	456.8	0.8922	968.8	0.9461	1992.8	0.9730
0.74	162.1	0.6332	418.0	0.8164	930.0	0.9082	1954.0	0.9541
0.75	157.0	0.6131	412.8	0.8062	924.8	0.9031	1948.8	0.9515
0.76	57.6	0.2252	313.3	0.6120	825.3	0.8060	1849.3	0.9030
0.77	51.9	0.2029	104.9	0.2049	212.9	0.2079	1236.9	0.6040
0.78	47.5	0.1854	95.3	0.1861	191.1	0.1866	382.9	0.1870
0.79	43.6	0.1702	87.3	0.1705	174.8	0.1707	349.8	0.1708
0.8	40.1	0.1567	80.3	0.1568	160.7	0.1569	321.4	0.1569
0.9	15.7	0.0612	31.3	0.0612	62.6	0.0612	125.3	0.0612
1	0	0	0	0	0	0	0	0

attack drastically decreases when $\sigma_l = 0.5$ compared to $\sigma_l = 0.6$, and no successful cases are achieved at $\sigma_l = 0.7$. When $\sigma_l = 0.5$, 0.6, and 0.7, the average entropy is almost 9, 9.5, and 10, respectively. Table 5.5 shows that the recovery ratio of the input drastically decreases for $\delta = 0.75$ to 0.76. Specifically, we recover more than 150 coefficients when $\delta = 0.75$, whereas we recover only 57.6 coefficients when $\delta = 0.76$. Now, the average entropy is $0.75 \log 7681 = 9.68$ and $0.76 \log 7681 = 9.81$, respectively. Thus, the decrease is almost the same as that in the previous model and our proposed approach. Previous research applies the average entropy to the real traces. Therefore, our analysis can be applied to a real trace.

5.4.5 Analysis of Algorithm 7

We now describe the equations of the renewal probabilities of Algorithm 7. The results of the analysis are given as the same as Tables 5.5 and 5.6.

We now focus on the renewal equation in each butterfly by using the local variables, $b^{\boldsymbol{l}}$ and $p_i^{(l)}$. First, we give a brief strategy of the renewal of $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ in each butterfly. We describe the renewal in the forward recovery step, and the renewal in the backward recovery step is similarly given. In the forward recovery step, we have renewed $p_1^{(l)}$ and $p_2^{(l)}$ in previous butterflies. By using $p_1^{(l)}$ and $p_2^{(l)}$, we renew $p_3^{(l)}$ and $p_4^{(l)}$. However, we do not renew $p_1^{(l)}$ or $p_2^{(l)}$ because we do not use newly recovered m_1 or m_2 immediately in the forward recovery step, For example, when we consider the renewal $\boldsymbol{l} = \mathbf{1010}$, we have already recovered m_1 and m_3. Then, we have the following:

- If we do not recover m_2, we recover m_4 from m_1 and m_3, and the state of the butterfly changes into **1011**.
- Otherwise, we newly recover m_2 and m_4, and the state of butterfly changes into **1111**.

Table 5.6: Theoretical value of average recovery rate of coefficients of the secret polynomial for the Gentleman-Sande INTT

N / δ	256		512		1024		2048	
	count	ratio	count	ratio	count	ratio	count	ratio
0	256	1	512	1	1024	1	2048	1
0.2	255.7	0.9990	511.7	0.9995	1023.7	0.9997	2047.7	0.9999
0.4	255.2	0.9968	511.2	0.9984	1023.2	0.9992	2047.2	0.9996
0.6	252.9	0.9877	508.9	0.9939	1020.9	0.9969	2044.9	0.9985
0.7	246.9	0.9646	502.9	0.9823	1014.9	0.9911	2038.9	0.9956
0.71	245.6	0.9592	501.6	0.9796	1013.6	0.9898	2037.6	0.9949
0.72	243.9	0.9526	499.9	0.9763	1011.9	0.9882	2035.9	0.9941
0.73	241.9	0.9448	497.9	0.9724	1009.9	0.9862	2033.9	0.9931
0.74	239.5	0.9357	495.5	0.9678	1007.5	0.9839	2031.5	0.9920
0.75	236.6	0.9241	492.6	0.9620	1004.6	0.9810	2028.7	0.9905
0.76	226.8	0.8859	482.8	0.9429	994.8	0.9715	2018.8	0.9857
0.77	221.5	0.8654	477.5	0.9327	989.5	0.9663	2013.5	0.9832
0.78	185.8	0.7256	441.8	0.8628	953.8	0.9314	1977.8	0.9657
0.79	106.6	0.4162	214.2	0.4184	430.2	0.4201	1454.2	0.7100
0.8	98.1	0.3834	196.6	0.3839	393.5	0.3842	787.3	0.3844
0.9	42.3	0.1652	84.6	0.1652	169.2	0.1652	338.3	0.1652
1	0	0	0	0	0	0	0	0

Therefore, we conduct the following:

- We add $\dfrac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} b^{\mathbf{1010}}$ to $b^{\mathbf{1011}}$.
- We add $\dfrac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}} b^{\mathbf{1010}}$ to $b^{\mathbf{1111}}$.

Moreover, we renew $p_i^{(l)}$. In the above example, m_3 has already been recovered and used, namely, in **State 3**, and we add $b^{\mathbf{1011}}$ to $p_3^{(3)}$. We consistently newly recover m_4, namely, in **State 2**, and we add $b^{\mathbf{1011}}$ to $p_4^{(2)}$. We propagate $p_i^{(l)}$, namely, $P_{i,k}^{(l)}$ in the global variables, to the other butterflies. We calculate the total change of $P_{i,k}^{(l)}$ during each iteration. If this value is smaller than the initially determined threshold, we stop our analysis and output the sum of $P_{0,k,3}$ as the recovery rate.

We now give the renewal equation of $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ in more detail. As the difference from the analysis in Algorithm 6, we do not renew $p_1^{(l)}$ or $p_2^{(l)}$ in the forward recovery step or $p_3^{(l)}$ or $p_4^{(l)}$ in the backward recovery step.

Renewal Equation in Forward Recovery Step

The renewal equation in the forward recovery step is different in the first or other iterations. During the first iteration, we renew $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ using the following:

$$b^{\mathbf{0000}} = \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}$$

$$b^{\mathbf{1000}} = \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}} \frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}} \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}} \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}$$

$$
\begin{aligned}
b^{\mathbf{0100}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{0010}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{0001}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{0011}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\\
b^{\mathbf{1011}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(1-\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
b^{\mathbf{0111}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\left(1-\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
b^{\mathbf{1111}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\\
p_3^{(1)} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_3^{(2)} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)\\
p_4^{(1)} &= \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\right)\\
p_4^{(2)} &= \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}+\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\left(\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\right.\\
&\quad\left.+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\right),
\end{aligned}
$$

and the other values are all 0. In the other iteration, we renew $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ by applying

$$
\begin{aligned}
b^{\mathbf{0000}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0000}}\\
b^{\mathbf{1000}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0000}}+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{1000}}\\
b^{\mathbf{0100}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}b^{\mathbf{0100}}\\
b^{\mathbf{0010}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0010}}\\
b^{\mathbf{0001}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0001}}
\end{aligned}
$$

$$
\begin{aligned}
b^{\mathbf{0011}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0011}}\\
b^{\mathbf{1011}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(b^{\mathbf{0010}}+b^{\mathbf{0001}}+b^{\mathbf{0011}}\right)+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\left(b^{\mathbf{1010}}+b^{\mathbf{1001}}+b^{\mathbf{1011}}\right)\\
b^{\mathbf{0111}} &= \frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\left(b^{\mathbf{0010}}+b^{\mathbf{0001}}+b^{\mathbf{0011}}\right)+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\left(b^{\mathbf{0110}}+b^{\mathbf{0101}}+b^{\mathbf{0111}}\right)\\
b^{\mathbf{1111}} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\left(b^{\mathbf{0000}}+b^{\mathbf{0010}}+b^{\mathbf{0001}}+b^{\mathbf{0011}}\right)\\
&\quad+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\left(b^{\mathbf{1000}}+b^{\mathbf{1010}}+b^{\mathbf{1001}}+b^{\mathbf{1011}}\right)\\
&\quad+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\left(b^{\mathbf{0100}}+b^{\mathbf{0110}}+b^{\mathbf{0101}}+b^{\mathbf{0111}}\right)+b^{\mathbf{1100}}+b^{\mathbf{1110}}+b^{\mathbf{1101}}+b^{\mathbf{1111}}\\
p_3^{(1)} &= \left(1-\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\right)b^{\mathbf{0000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0001}}+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{1000}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}b^{\mathbf{0100}}\\
p_3^{(2)} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0000}}+\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\right)b^{\mathbf{0001}}+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{1000}}\\
&\quad+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}b^{\mathbf{0100}}+b^{\mathbf{1100}}+b^{\mathbf{1001}}+b^{\mathbf{0101}}+b^{\mathbf{1101}}\\
p_3^{(3)} &= b^{\mathbf{0010}}+b^{\mathbf{1010}}+b^{\mathbf{0110}}+b^{\mathbf{1110}}+b^{\mathbf{0011}}+b^{\mathbf{1011}}+b^{\mathbf{0111}}+b^{\mathbf{1111}}\\
p_4^{(1)} &= \left(1-\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}\right)b^{\mathbf{0000}}+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0010}}+\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{1000}}\\
&\quad+\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}b^{\mathbf{0100}}\\
p_4^{(2)} &= \frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{0000}}+\left(1-\frac{p_1^{(1)}}{p_1^{(1)}+p_1^{(2)}}\frac{p_2^{(1)}}{p_2^{(1)}+p_2^{(2)}}\right)b^{\mathbf{0010}}+\frac{p_2^{(2)}}{p_2^{(1)}+p_2^{(2)}}b^{\mathbf{1000}}\\
&\quad+\frac{p_1^{(2)}}{p_1^{(1)}+p_1^{(2)}}b^{\mathbf{0100}}+b^{\mathbf{1100}}+b^{\mathbf{1010}}+b^{\mathbf{0110}}+b^{\mathbf{1110}}\\
p_4^{(3)} &= b^{\mathbf{0001}}+b^{\mathbf{1001}}+b^{\mathbf{0101}}+b^{\mathbf{1101}}+b^{\mathbf{0011}}+b^{\mathbf{1011}}+b^{\mathbf{0111}}+b^{\mathbf{1111}}
\end{aligned}
$$

and the other values are all 0. After each iteration, we apply the following additional renewal in $0 \le i \le N-1$:

$$
\begin{aligned}
P^{(2)}_{\log N,i} &= 0,\\
P^{(3)}_{\log N,i} &= P^{(2)}_{\log N,i} + P^{(3)}_{\log N,i}.
\end{aligned}
$$

Renewal Equation in Backward Recovery Step

In the backward recovery step, we input $p_{3,l}$ and $p_{4,l}$, and do not change these values. We renew $b^{\boldsymbol{l}}$ and $p_i^{(l)}$ by the following:

$$
b^{\mathbf{0000}} = \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}}
$$

$$
\begin{aligned}
b^{\mathbf{1000}} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{1000}} \\
b^{\mathbf{0100}} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0100}} \\
b^{\mathbf{1100}} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{1100}} \\
b^{\mathbf{0010}} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}} + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}} \\
b^{\mathbf{1110}} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\left(b^{\mathbf{1000}}+b^{\mathbf{0100}}+b^{\mathbf{1100}}\right) + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\left(b^{\mathbf{1010}}+b^{\mathbf{0110}}+b^{\mathbf{1110}}\right) \\
b^{\mathbf{0001}} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}} + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}} \\
b^{\mathbf{1101}} &= \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\left(b^{\mathbf{1000}}+b^{\mathbf{0100}}+b^{\mathbf{1100}}\right) + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\left(b^{\mathbf{1001}}+b^{\mathbf{0101}}+b^{\mathbf{1101}}\right) \\
b^{\mathbf{1111}} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\left(b^{\mathbf{0000}}+b^{\mathbf{1000}}+b^{\mathbf{0100}}+b^{\mathbf{1100}}\right) \\
&\quad + \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\left(b^{\mathbf{0010}}+b^{\mathbf{1010}}+b^{\mathbf{0110}}+b^{\mathbf{1110}}\right) \\
&\quad + \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\left(b^{\mathbf{0001}}+b^{\mathbf{1001}}+b^{\mathbf{0101}}+b^{\mathbf{1101}}\right) + b^{\mathbf{1100}}+b^{\mathbf{1110}}+b^{\mathbf{1101}}+b^{\mathbf{1111}} \\
p_1^{(1)} &= \left(1-\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}} + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0100}} + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}} \\
&\quad + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}} \\
p_1^{(2)} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}} + \left(1-\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0100}} + \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}} \\
&\quad + \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}} + b^{\mathbf{0011}} + b^{\mathbf{0110}} + b^{\mathbf{0101}} + b^{\mathbf{0111}} \\
p_1^{(3)} &= b^{\mathbf{1000}} + b^{\mathbf{1010}} + b^{\mathbf{1001}} + b^{\mathbf{1011}} + b^{\mathbf{1100}} + b^{\mathbf{1110}} + b^{\mathbf{1101}} + b^{\mathbf{1111}} \\
p_2^{(1)} &= \left(1-\frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{0000}} + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{1000}} + \frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}} \\
&\quad + \frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}} \\
p_2^{(2)} &= \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0000}} + \left(1-\frac{p_3^{(1)}}{p_3^{(1)}+p_3^{(2)}}\frac{p_4^{(1)}}{p_4^{(1)}+p_4^{(2)}}\right)b^{\mathbf{1000}} + \frac{p_4^{(2)}}{p_4^{(1)}+p_4^{(2)}}b^{\mathbf{0010}} \\
&\quad + \frac{p_3^{(2)}}{p_3^{(1)}+p_3^{(2)}}b^{\mathbf{0001}} + b^{\mathbf{0011}} + b^{\mathbf{1010}} + b^{\mathbf{1001}} + b^{\mathbf{1011}} \\
p_2^{(3)} &= b^{\mathbf{0100}} + b^{\mathbf{0110}} + b^{\mathbf{0101}} + b^{\mathbf{0111}} + b^{\mathbf{1100}} + b^{\mathbf{1110}} + b^{\mathbf{1101}} + b^{\mathbf{1111}}
\end{aligned}
$$

and the other values are all 0. After an iteration, we apply an additional renewal as follows in $0 \le i \le N-1$:

$$P_{0,i}^{(2)} = 0,$$

$$P_{0,i}^{(3)} = P_{0,i}^{(2)} + P_{0,i}^{3}.$$

5.5 Recovery of the Full LPR Secret Keys

In this section, we analyze the method for recovering the full LPR secret keys. We construct our method for recovering the full LPR secret keys by replacing the recovery of $M_{i,k}$'s in Primas et al.'s method [95] into our approach for recovering the secret polynomial proposed in Section 5.3. Then, we analyze the security of the LPR cryptosystem. First, we give a rough analysis of the recovery of the full LPR secret keys in Section 5.5.1. Next, we describe the exact analysis of the recovery of the full LPR secret keys in Section 5.5.2. Finally, we provide the results of our numerical experiments on recovering the full LPR secret keys in Section 5.5.3.

5.5.1 Rough Analysis of Recovery of the Full LPR Secret Keys

First, we consider how many values are obtained on our leakage model. Our leakage model focuses on $N \log N/2$ values from butterflies and N values from the final multiplication. However, redundancy exists in the leakage value. We count $M_{0,k}$ $(N/2 \leq k \leq N-1)$ twice, and the number of such values is $N/2$. Moreover, there are butterflies having redundancy. Specifically, butterfly operations are conducted as follows:

- **Input**: $M_{i+1,k}$ and $M_{i+1,k+2^{n-1-i}}$
- **Output**: $M_{i,k}$ and $M_{i,k+2^{n-1-i}}$

In the above, $\lfloor k/2^{n-1-i} \rfloor$ is always even. Because

$$\left\lfloor \frac{k+2^{n-1-i}}{2^{n-2-i}} \right\rfloor = \left\lfloor \frac{k}{2^{n-2-i}} \right\rfloor + 2,$$

the parities of $\lfloor k/2^{n-2-i} \rfloor$ and $\lfloor \left(k+2^{n-1-i}\right)/2^{n-2-i} \rfloor$ are the same. Thus, there are butterflies for which we try to obtain three values. Such butterflies satisfy $0 \leq i \leq \log N - 2$, $\lfloor k/2^{n-1-i} \rfloor$ is even, and $\lfloor k/2^{n-2-i} \rfloor$ is odd. The number of such butterflies is $N(\log N - 1)/4$, and there is a redundant value in each butterfly. As a result, the number of redundant values is $N(\log N - 1)/4 + N/2 = N(\log N + 1)/4$. Thus, we try to obtain

$$\frac{N \log N}{2} + N - \frac{N(\log N + 1)}{4} = \frac{N(\log N + 3)}{4}$$

values.

Next, we analyze the bound of δ for which the full LPR secret keys are recovered. Now, we obtain $N(\log N + 3)/4$ values with probability $1 - \delta$. From Primas et al.'s study, we require partial αN $(0 \leq \alpha \leq 1)$ coefficients out of N coefficients. Then, we can recover the full LPR secret keys when

$$\frac{N(1-\delta)(\log N + 3)}{4} > \alpha N,$$

and we obtain the rough bound of δ for recovering the LPR secret keys as

$$\delta < 1 - \frac{4\alpha}{\log N + 3}. \tag{5.2}$$

We now apply this rough analysis to the actual parameters. When $N = 256$, the full LPR secret keys are recovered when 150 coefficients are obtained. These values are

based on Primas et al.'s experiment. Thus, by substituting $N = 256$ and $\alpha = 150/256$ in Eq. (5.2), the rough bound of δ is given as $\delta < 0.79$. Similarly, we consider the case of $N = 512$. In $N = 256$, we may recover the remaining $106(= 256 - 150)$ coefficients. Therefore, when $N = 512$, the full LPR secret keys are recovered when $406(= 512 - 106)$ coefficients are obtained. Thus, by substituting $N = 512$ and $\alpha = 406/512$ in Eq. (5.2), the rough bound of δ is given as $\delta < 0.74$.

5.5.2 Exact Analysis of Recovery of the Full LPR Secret Keys Using the Result of Section 5.4

We now give the bound on recovering the full LPR secret keys. We give the theoretical bound based on Table 5.6. When $N = 256$, the full LPR secret keys are recovered when 150 coefficients are obtained. Thus, Table 5.6 describes that the full LPR secret keys are recovered when $\delta \leq 0.78$. When $N = 512$, the full LPR secret keys are recovered when 406 coefficients are obtained. Thus, the full LPR secret keys are recovered when $\delta \leq 0.78$, similar to $N = 256$.

This bound, $\delta \leq 0.78$, is satisfied in a larger N. When $N = 1024$, the LPR secret keys are recovered when $918(= 1024 - 106)$ coefficients are obtained. Thus, from Table 5.6, the LPR secret keys are recovered when $\delta \leq 0.78$. When $N = 2048$, the LPR secret keys are recovered when $1942(= 2048 - 106)$ coefficients are obtained. Thus, the LPR secret keys are recovered when $\delta \leq 0.78$.

Therefore, the LPR secret keys are recovered when $\delta \leq 0.78$ for a larger N. This analysis is based on the number of non-recovered values by the lattice reduction algorithm. Thus, if the lattice reduction algorithm recovers more coefficients than the current one, more secret keys will be recovered. Our analysis based on Table 5.6 can be applied even if the lattice reduction algorithm improves.

5.5.3 Numerical Experiments on Full Recovery of the LPR Secret Keys

We now show the experiment of recovering the secret keys on the LPR cryptosystem. We ran the proposed method on $(N, q, \sigma) = (256, 7681, 4.51)$ and $(512, 12289, 4.86)$, which were proposed by Göttert et al. [38]. In each parameter set, we set the fraction rate δ, and generated 100 secret keys for each (N, q, σ, δ).

In this experiment, we recovered the LPR secret keys, which we could recover within an hour. To realize this, we applied a lattice reduction when we recovered more than 160 coefficients at $N = 256$ and more than 430 coefficients at $N = 512$. We outputted failure in other cases. We then measured the success rate and average implementation times for each successful trial. Table 5.7 shows the experimental results.

Table 5.7 describes that our method recovers the LPR secret keys completely when $\delta \leq 0.7$. When $\delta = 0.75$, our method recovers more than half of the secret keys, which matches the result in Section 5.3. For a larger δ, the success rate decreases suddenly, and this result matches our analysis in Section 5.4.

In the above experiment, we recovered the LPR secret keys to recover within an hour. We calculated the potential recovery rate of the LPR secret keys. We generated 100 polynomials in each (N, q, δ), and counted the number of trials recovering more than 150 coefficients when $N = 256$ and more than 406 coefficients when $N = 512$. Table 5.8 shows the experimental results, and we now focus when $\delta = 0.78$. Table 5.8 describes that our method recovers approximately $1/3$ of the LPR secret keys when $N = 256$ and approximately $1/10$ of the LPR secret keys when $N = 512$. Therefore, this experiment validates our analysis.

Table 5.7: Recovery rate of the LPR secret keys

		δ							
N		0	0.7	0.75	0.76	0.77	0.78	0.79	0.8
256	success rate (%)	100	100	81	58	34	18	6	0
	time [sec]	7.54	35.8	168.0	221.5	334.4	310.8	437.3	–
512	success rate (%)	100	100	62	43	16	2	0	0
	time [sec]	58.1	199.0	447.7	516.0	595.7	696.8	–	–

Table 5.8: Potential recovery rate of the LPR secret keys

		δ					
N	#(recovered)	0.75	0.76	0.77	0.78	0.79	0.8
256	150–159	6	16	13	9	4	0
	160–256	83	59	33	21	8	2
	Total	89	75	46	30	12	2
512	406–429	17	24	9	5	0	0
	430–512	67	43	17	3	1	0
	Total	84	67	26	8	1	0

5.6 Conclusion and Future Work

In this study, we considered applying an erasure model to the inputs of a multiplication. We assumed that we extract the input of multiplication with the probability of $1 - \delta$. We then constructed a recovery algorithm for a secret polynomial that consistently stops. Next, we analyzed the recovery rate of the coefficients, which has correspondence with the previous results. Finally, we conducted a security analysis on the LPR cryptosystem and proved that our method recovers the LPR secret keys when $\delta \leq 0.78$ under the current computational power. Then, we applied a concrete analysis method to the threats to an LPR cryptosystem.

In a future study, we will apply our analysis to side-channel attacks on actual NIST candidates, i.e., CRYSTALS-KYBER [90]. Specifically, we must consider the difference in the implementation of the NTT. To realize an accurate computation of CRYSTALS-KYBER, we use a smaller q satisfying $q \equiv 1 \bmod N$, and apply an NTT from $i = 0$ to $\log N - 2$ in Algorithm 2. Thus, we must consider this difference in our approach. This difference will be solved with a small extension. Moreover, our analysis can be connected with Xu et al.'s attack [118], and we must also consider the security of CRYSTALS-KYBER by combining our result with such an attack.

We must also analyze the number of iterations in converging our recovery algorithm. As shown in Tables 5.1 and 5.2, our recovery algorithm terminates much fewer iterations than the upper-bound $N(\log N + 1) + 1$. By evaluating the number of iterations, we must clarify the time complexity of our recovery algorithm.

Moreover, we must also consider the time complexity of the lattice reduction algorithm used in recovering all LPR secret keys. Currently, we are analyzing our method by focusing on the number of non-recovered coefficients. If the lattice reduction algorithm recovers more non-recovered coefficients than with the current approach, our method can be applied to broader situations. Thus, in our case, we must analyze the time complexity of the lattice reduction algorithm.

Chapter 6

Security Evaluation of RSA Scheme under an Efficient Construction of a Quantum Controlled Modular Adder

6.1 Introduction

6.1.1 Background

A controlled modular addition is an essential operation for Shor's algorithm. The optimal construction of a controlled modular adder is not apparent in quantum computers. A controlled modular adder is constructed from simple adders [22, 28, 41, 100, 113, 116, 121], among which there are many types [21, 25, 26, 33, 108, 113, 116]. Although an approximate controlled modular adder has also been proposed [35, 121], we only focus on accurate controlled modular adders in this chapter. Now, the previous construction follows a similar overall structure but differs in detail. We need to determine which combination is the best.

To evaluate the efficiency of the circuit, we minimize the depth of a controlled modular adder. A candidate controlled modular adder with a small depth was proposed by Van Meter and Itoh [113]. Van Meter-Itoh's construction uses three carry-lookahead adders. Based on their construction, Van Meter et al. [114] and Jones et al. [52] analyzed the computational cost of Shor's algorithm on Fault-Tolerant Quantum (FTQ) computers. Then, Jones et al. showed that we must assign a large fraction of our total distillation resources.

Moreover, when the depth is minimized, KQ introduced in [107] is also minimized. KQ is defined as a product of the number of qubits and the depth of a circuit, and minimizing KQ benefits both FTQ computers [52] and Noisy Intermediate-Scale Quantum (NISQ) computers [101]. Mainly, calculation errors occur even in FTQ computers, and errors are fewer in a circuit with a small KQ.

However, Van Meter-Itoh's construction has room for further minimization of the number of T gates. Thapliyal et al. [111] proposed the means of minimizing the number of T gates in a carry-lookahead adder. This method replaces several Toffoli gates using Gidney's relative-phase Toffoli gates [33]. Thus, we may reduce the number of T gates by applying Gidney's relative-phase Toffoli gates on the construction developed by Van Meter and Itoh.

In the above discussion, although we consider executions on FTQ computers, it is also essential to consider an efficient circuit for NISQ computers because we are currently in the NISQ era. Currently, NISQ machines have higher error rates on CNOT gates than on

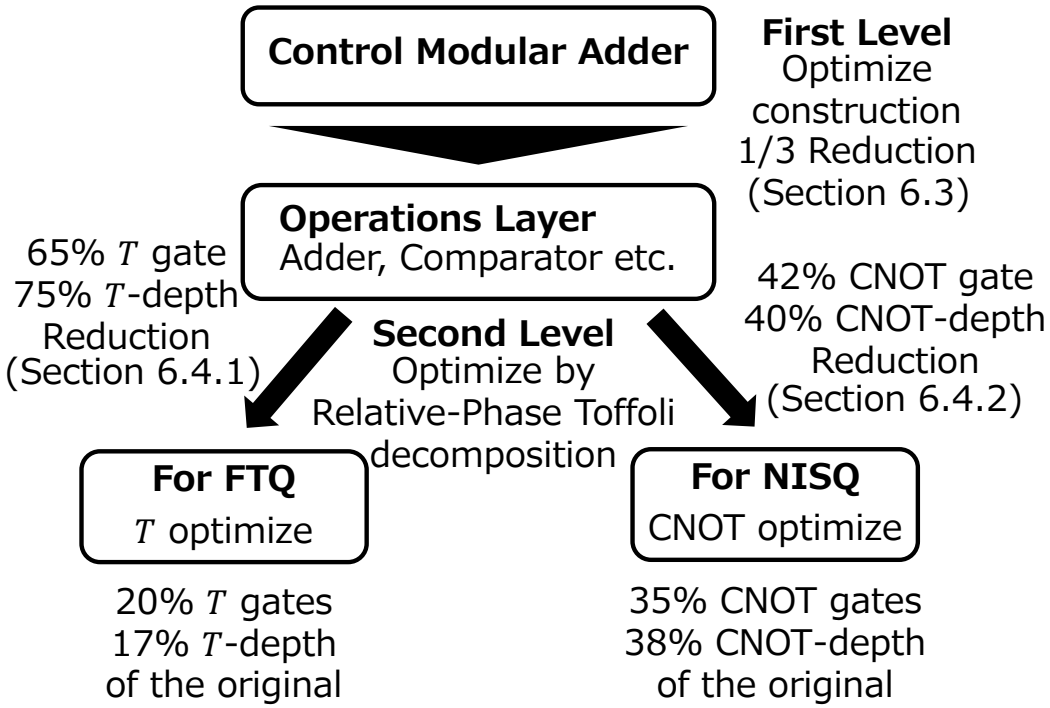

Fig. 6.1: Abstract of our results. This figure shows the two-level optimization of a controlled modular adder. In the first-level optimization, we optimize the construction of a controlled modular adder. In the second-level optimization, we minimize the depth for FTQ or NISQ computers by relative-phase Toffoli gates.

one-qubit gates [19, 48]. Thus, we must reduce the depth based on CNOT gates for NISQ computers. By using relative-phase Toffoli gates composed of a smaller number of CNOT gates [70] compared to a standard Toffoli gate, we may reduce the cost of a controlled modular adder for NISQ computing.

6.1.2 Our Contribution

This study proposes a method for optimizing a controlled modular adder based on a carry-lookahead adder. We apply two-level optimization on Van Meter-Itoh's original construction [113], as described in Figure 6.1. This chapter is written based on our paper submitted to arXiv preprint [86].

In the first-level optimization, we optimize the construction of a controlled modular adder. Specifically, we minimize the depth by focusing on the comparator in a carry-lookahead adder. Then, we reduce some of the controlled operations by taking advantage of the classicality of a and N.

In the second-level optimization, we minimize the depth for FTQ or NISQ computers using relative-phase Toffoli gates. This study assumes that all qubits are fully connected without considering the physical or logical topology [18, 28, 45]. We then reduce the T-depth in FTQ computing and CNOT-depth in NISQ computing. Instead of the standard Toffoli gates, we use Gidney's relative-phase Toffoli gates [33] for FTQ computing and Maslov's relative-phase Toffoli gates [70] for NISQ computing.

Our construction also minimizes the KQ. We clarify the definition of KQ in each device

because the cost of the gates is different between FTQ and NISQ computers. Specifically, we define KQ_T and $\mathrm{KQ}_{\mathrm{CX}}$, which is defined by the product of the number of qubits and the T-depth or CNOT-depth, respectively. We then minimize KQ_T and $\mathrm{KQ}_{\mathrm{CX}}$ for FTQ and NISQ computers. Moreover, we must consider the cost of distillation circuits for FTQ computers because distillation circuits require a high cost. When we consider distillation circuits, there is a trade-off between T-depth and the number of T gates running simultaneously. We show that we can achieve the smallest KQ_T when we run $\Theta\left(n/\sqrt{\log n}\right)$ T gates simultaneously.

Finally, we evaluate the security of the RSA scheme against FTQ computers. We evaluate the security based on Jones et al.'s evaluation [52]. We construct a quantum circuit of Shor's algorithm based on our controlled modular adder. Then, we estimate the required time for Shor's algorithm, based on IBM's plan for quantum computer development [49]. Our estimation shows that quantum computers will be able to break the 2048-bit RSA scheme 28.2 years later, which is 1.4 years earlier than Van Meter-Itoh's construction.

6.1.3 Organization of this Chapter

This chapter has five sections. Section 6.2 provides the preliminaries for this chapter. Section 6.3 describes the first-level optimization on a controlled modular adder. Section 6.4 details the second-level optimization, particularly in each FTQ or NISQ. Section 6.5 discusses the security of the RSA scheme. Section 6.6 concludes this chapter.

6.2 Preliminaries

This section gives preliminaries on this chapter. First, Section 6.2.1 explains the computational model of FTQ computers [52]. Next, Section 6.2.2 describes the relative-phase Toffoli gates, which are used to reduce the computational cost. Then, Section 6.2.3 shows the general construction of a controlled modular adder. Finally, Section 6.2.4 details the previous construction of a carry-lookahead adder, which is used in our construction.

6.2.1 The Computational Model of FTQ Computers [52]

As noted in Section 1.1.3, NISQ computers have high error rates in each operation, and research on FTQ is proceeding. In FTQ, operations are performed on qubits with error-correction, while operations are performed on physical qubits in NISQ. Jones et al. [52] considered a method for constructing FTQ computers as a layered architecture. Specifically, we perform the accurate computation on the Logical layer in FTQ computers, realized by many physical qubits with errors. Then, we explain Jones et al.'s computational model of the Logical layer in FTQ computers.

In this architecture, we adopt a fundamental gate set consisting of X, Y, Z, CNOT, and H gates. To run an S gate, we prepare an ancilla qubit $|Y\rangle = (|0\rangle + \mathrm{i}\,|1\rangle)/\sqrt{2}$ and run the circuit shown in Figure 6.2. An $S^\dagger$ gate can be realized by the reverse circuit of Figure 6.2.

Next, we prepare a T gate. To run a T gate, we prepare an ancilla qubit $|A\rangle$ given as $\left(|0\rangle + \mathrm{e}^{\mathrm{i}\pi/4}\,|1\rangle\right)/\sqrt{2}$ and run the circuit shown in Figure 6.3. To run a $T^\dagger$ gate, we apply a $S^\dagger$ gate instead of a S gate.

By Gottesman-Knill theorem [81], we can perform classical simulation on quantum circuits composed of quantum gate set except for T gates and measurement. However, to realize the universal quantum computation, we require a T gate that cannot be simulated classically. Thus, we require additional cost to realize T gates in FTQ computers. We

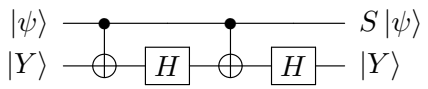

Fig. 6.2: Running a S gate [52]. $|Y\rangle$ in the second qubit is $(|0\rangle + \mathrm{i}\,|1\rangle)/\sqrt{2}$. This second qubit preserves before and after this computation.

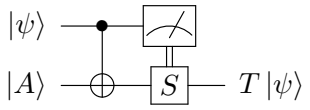

Fig. 6.3: Running a T gate [52]. $|A\rangle$ in the second qubit is $\left(|0\rangle + \mathrm{e}^{\mathrm{i}\pi/4}\,|1\rangle\right)/\sqrt{2}$.

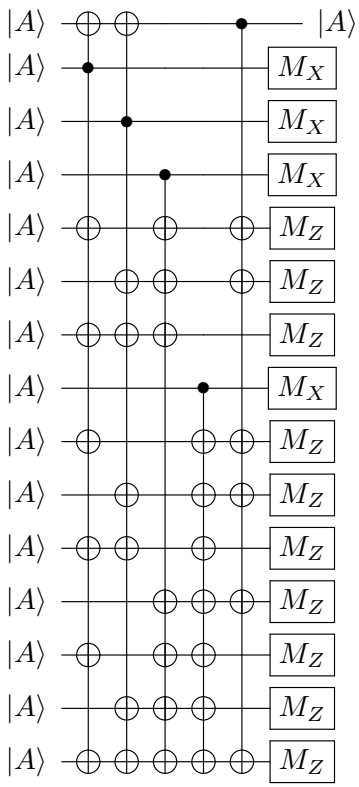

Fig. 6.4: A distillation circuit of $|A\rangle$ [29]. By this distillation circuit, we reduce the error rate of $|A\rangle$ from p to $35p^3$. In M_Z, we perform the measurement. In M_X, we measure after applying a H gate. By using these results of measurement, we perform error-correction on $|A\rangle$.

realize T gates by distillation [29], which requires a lot of logical qubits and time steps additionally, and research on optimization of distillation is carried out [36, 37]. Specifically, preparing $|A\rangle$ is done by a distillation circuit. One of the distillation circuits is shown in Figure 6.4. This distillation circuit requires 15 qubits and 6 time steps, even assuming all CNOT gates can be implemented concurrently, but this is not easy to realize. Thus, a T gate has the highest cost in FTQ computers, and we must reduce the number of T gates.

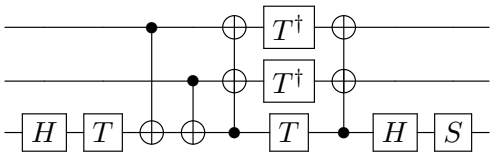

Fig. 6.5: Gidney's relative-phase Toffoli gate [33] given by the unitary matrix (6.1). We call this decomposition GRT. The control bits are the first and second qubits, and the target bit is the third qubit. This calculation preserves the phase only when we input $|0\rangle$ at the target qubit.

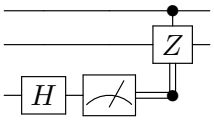

Fig. 6.6: Inverse of Gidney's relative-phase Toffoli gate [33]. We call this decomposition IGRT. This calculation preserves the phase when we input $|000\rangle, |010\rangle, |100\rangle$, or $|111\rangle$, those are output of GRT having valid phase. Control-Z is Clifford gate, and we use no T gate.

6.2.2 Relative-Phase Toffoli Gates

We now explain relative-phase Toffoli gates. Relative-phase Toffoli gates calculates following with some function f:

$$|x\rangle |y\rangle |z\rangle \to \mathrm{e}^{\mathrm{i}f(x,y,z)} |x\rangle |y\rangle |z \oplus (x \wedge y)\rangle ,$$

Then, relative-phase Toffoli gates calculate AND correctly, but the difference from (2.1) is $\mathrm{e}^{\mathrm{i}f(x,y,z)}$. In the quantum computation, we can reduce the cost of computation by using relative-phase Toffoli gates [33, 70, 111] and resetting the term $\mathrm{e}^{\mathrm{i}f(x,y,z)}$ in the total computation.

We now review Gidney's relative-phase Toffoligates (GRT) and the inverse of them (IGRT) [33]. These are constructed for FTQ computers. GRT is shown in Figure 6.5 and the corresponding unitary matrix of GRT in the computational basis is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} , \tag{6.1}$$

and we calculate correctly only when the target bit is $|0\rangle$. IGRT is shown in Figure 6.6.

Next, we review Maslov's relative-phase Toffoli gates [70]. Maslov proposed relative-phase Toffoli gates as Figure 6.7 and 6.8. These are constructed for NISQ computers. The

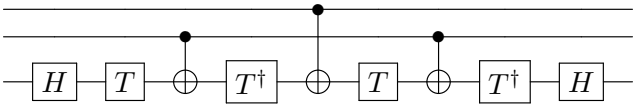

Fig. 6.7: A relative-phase Toffoli gate with 3 CNOT (RT3) [70] given by the unitary matrix (6.2). This calculation changes the phase when we input $|1\rangle|0\rangle|1\rangle$, $|1\rangle|1\rangle|0\rangle$, and $|1\rangle|1\rangle|1\rangle$. We call the inverse circuit of RT3 as IRT3.

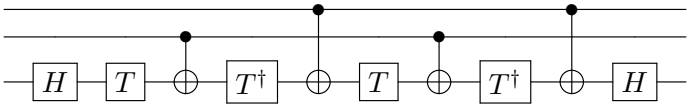

Fig. 6.8: A relative-phase Toffoli gate with 4 CNOT (RT4) [70] given by the unitary matrix (6.3). This calculation changes the phase when both control bits are 1. We call the inverse circuit of RT4 as IRT4.

corresponding unitary matrix of Figure 6.7 in the computational basis is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & 0 & 0 & i & 0 \end{bmatrix}. \tag{6.2}$$

This calculation changes the phase when we input $|1\rangle|0\rangle|1\rangle$, $|1\rangle|1\rangle|0\rangle$, or $|1\rangle|1\rangle|1\rangle$. We call this relative-phase Toffoli gate RT3, and we call its inverse IRT3. The corresponding unitary matrix of Figure 6.8 in the computational basis is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & 0 & 0 & -i & 0 \end{bmatrix}. \tag{6.3}$$

This calculation changes the phase when both control bits are 1. We call this relative-phase Toffoli gate RT4, and we call its inverse IRT4.

6.2.3 The General Construction of a Controlled Modular Adder

A controlled modular addition is defined by a control qubit x and n-bit numbers a, b, and N. a and b satisfy $0 \leq a, b \leq N-1$, and a and N are classical numbers. A controlled modular addition calculates

$$|x\rangle|b\rangle \rightarrow |x\rangle|b + xa \bmod N\rangle. \tag{6.4}$$

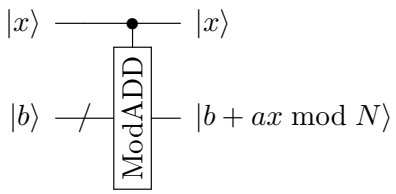

Fig. 6.9: Overview of a controlled modular adder. The first register has a single qubit, which is used as a control bit. The second register has n qubits, which are used to store the result. a and N are n-bit classical numbers.

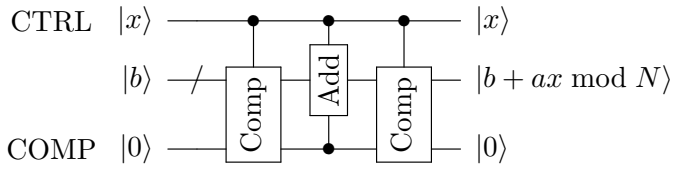

Fig. 6.10: The general construction of a controlled modular adder. Add means an adder, and Comp means a comparator. CTRL has a single qubit, which is used to hold the value of the control. $|b\rangle$ has n qubits, which are used to hold the result of a controlled modular addition. COMP has a single qubit, which is used to hold the result of a comparison. a and N are classical numbers.

An overview is shown in Figure 6.9.

This subsection explains the calculation of

$$|x\rangle |b\rangle |0\rangle \to |x\rangle |b + xa \bmod N\rangle |0\rangle ,$$

and the general construction of this controlled modular adder is shown in Figure 6.10. The first register has a single qubit, which is used to hold the value of the control. We call this the CTRL qubit. The second register has n qubits, which are used to hold the result of a controlled modular addition. The third register has a single qubit, which is used to hold the result of a comparison temporarily. We call this the COMP qubit. Precisely, we determine whether we subtract N or not based on COMP. We conduct a comparator with one control qubit and an adder with two control qubits, and we write these as a C-comparator and a CC-adder, respectively.

To execute a controlled modular adder, we conduct operations in this order:

1. We compare the second register $|b\rangle$ and the classical value $N - a$. If $b \geq N - a$, namely $a + b \geq N$, we flip COMP.
2. If both CTRL and COMP are 1, we subtract $N - a$ from the second register. If CTRL is 1 and COMP is 0, we add a. Otherwise, we add no value.
3. If the second register is strictly less than a, we flip COMP.

6.2.4 Carry-Lookahead Adder [26]

First, we explain the calculation of $a + b$ when a and b are n-bit numbers. We express a as $a_{n-1}a_{n-2} \ldots a_0$ and b as $b_{n-1}b_{n-2} \ldots b_0$, where a_i and b_i are 0 or 1. To calculate $a + b$, we introduce a carry c_i. Carry c_i is defined as an overflow from the $(i - 1)$-th bit to the

i-th bit. In more detail, we define c_i as

$$c_i = \begin{cases} 0 & \text{if } i = 0 \\ \left\lfloor \frac{a_{i-1} + b_{i-1} + c_{i-1}}{2} \right\rfloor & \text{otherwise} \end{cases}$$

Then, $(a+b)_i$, the i-th bit of $a+b$, is calculated as

$$(a+b)_i = a_i \oplus b_i \oplus c_i.$$

Thus, we need carries to calculate an addition.

We now give a brief explanation of a carry-lookahead adder. Before calculating an addition, we determine the propagation of a carry from the i-th bit to the j-th bit as the following three conditions:

- **Propagate**: A carry is propagated from the i-th bit to the j-th bit. Namely, $c_j = c_i$.
- **Generate**: A carry is generated in the j-th bit, namely $c_j = 1$, regardless of the value of c_i.
- **Kill**: A carry is killed in the j-th bit, namely $c_j = 0$, regardless of the value of c_i.

To calculate the propagation, we define two functions $p[i,j], g[i,j] \in \{0,1\}$. $p[i,j]$ is true when the carry from the i-th bit to the j-th bit should be propagated. Similarly, $g[i,j]$ is true when the value of the carry at the j-th bit is 1 independent of the carry at the i-th bit. We do not need a separate function for **kill**, as its value can be inferred from p and g. By using these functions, we can calculate the propagation state over a wider span. Specifically, when $i < k < j$,

$$p[i,j] = p[i,k] \wedge p[k,j], \tag{6.5}$$

$$g[i,j] = g[k,j] \oplus (g[i,k] \wedge p[k,j]), \tag{6.6}$$

where $\wedge$ is Boolean AND, and $\oplus$ is Boolean XOR. By using these properties, we calculate $c_j = g[0,j]$.

We now explain Draper et al.'s carry-lookahead adder for $|a\rangle|b\rangle \to |a\rangle|b+a\rangle$. This requires an additional n qubits for the carry register $|c\rangle$ and n qubits for register $|p\rangle$, containing $p[i,j]$. Thus, a carry-lookahead adder requires $4n$ qubits.

We now explain the implementation briefly. The detailed construction is given in later. This implementation consists of five phases, Initialization, P-rounds, G-rounds, C-rounds, and inverse P-rounds. In each round,

- **Initialization**: we calculate $g[i,i+1]$ in $|c_{i+1}\rangle$ and $p[i,i+1]$ in $|b_i\rangle$,
- **P-rounds**: we calculate the p-function and write result in $|p\rangle$,
- **G-rounds**: we calculate $|c_{2^k}\rangle$ $(k \in \mathbb{N})$ by calculating some g-function,
- **C-rounds**: we calculate all carries $|c\rangle$ by calculating some g-function,

and we clean $|p\rangle$ in inverse P-rounds. After inverse P-rounds, we calculate each bit of $a+b$ by using these carries $|c\rangle$. In this calculation, we run P-rounds and G-rounds simultaneously, and we run C-rounds and inverse P-rounds simultaneously. However, the value of carries remains on $|c\rangle$. Thus, we must clean $|c\rangle$ to $|0\rangle$ except for c_n. Draper et al. found that the value of carries c_i except for c_n in $a+b$ is the same in $a+(2^n-1-a-b)$. Therefore, we erase carries by performing the addition $a+(2^n-1-a-b)$ on the lower $n-1$ qubits. The abstract circuit is shown in Figure 6.11.

As noted above, a carry-lookahead adder is mainly constructed by a calculation of p and g. We calculate p and g with Eq. (6.5) or (6.6) respectively, and those are implemented

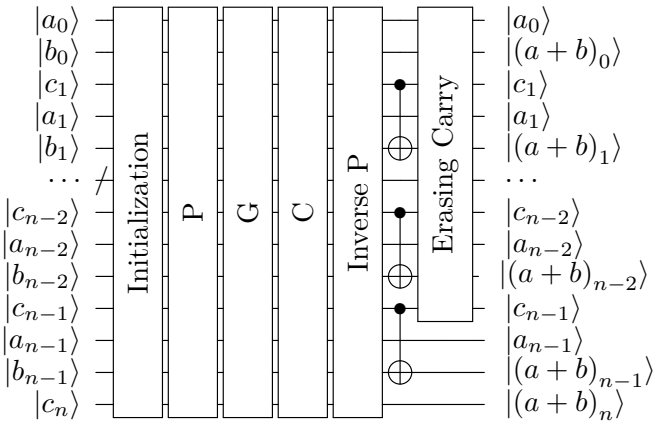

Fig. 6.11: An abstract figure of Draper et al.'s carry-lookahead adder. In this figure, we sort qubits from the lowest qubits to the highest qubits, which is different from Figure 6.10. $|c_i\rangle$ is given as $|0\rangle$ at the beginning of this circuit, and these are cleared as $|0\rangle$ after Erasing Carry.

(a) A calculation circuit of $p[i,j]$ as Eq. (6.5). (b) A calculation circuit of $g[i,j]$ as Eq. (6.6).

Fig. 6.12: A calculation circuit of $p\,[i,j]$ and $g\,[i,j]$.

by Toffoli gates as shown in Figure 6.12. In total, a carry-lookahead adder requires $10n$ Toffoli gates and $4n$ CNOT gates. Moreover, the Toffoli depth is $4\log n$.

Up to this point, we have explained the construction of an adder. Draper et al. also proposed other operations, such as a subtractor and a comparator, based on their adder. The number of gates and the depth in a subtractor is almost the same as those in an adder. In a comparator, the number of gates is 60% of an adder, and the depth is 50% of an adder. Draper et al. implement a comparator using only Initialization, P-rounds, G-rounds, and their inverses. More precisely, Draper et al. regard a and b as $2^{\lceil \log n \rceil}$-bit numbers by padding 0 in higher bits, but we do not use these qubits. If we calculate $p\,[i,j]$ or $g\,[i,j]$ when $i \leq n-1$ and $j \geq n$, we calculate $p\,[i,n]$ or $g\,[i,n]$ respectively. Then, we calculate $g\,[0,n]$ after G-rounds.

Detailed Explanation of Quantum Carry-lookahead Adder

Draper et al.'s carry-lookahead adder is given as follows:

Initialization (n Toffoli gates and n CNOT gates)

We calculate $g\,[i,i+1]$ and $p\,[i,i+1]$ $(0 \leq i \leq n-1)$, as follows:

$$g\,[i,i+1] = \begin{cases} 1 & \text{if } a_i = b_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$p\,[i,i+1] = \begin{cases} 1 & \text{if } a_i + b_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

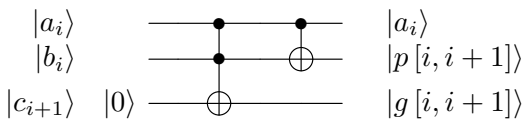

Fig. 6.13: A calculation circuit of $g[i,i+1]$ and $p[i,i+1]$ $(0 \leq i \leq n-1)$. We use $|c_{i+1}\rangle$ as the third qubit. We can run these gates simultaneously for $i = 0$ to $n-1$.

The circuit calculating these is shown in Figure 6.13.

P-rounds (n Toffoli gates and $\log n$ Toffoli depth)

We calculate the p-function by using Eq. (6.5). We use a parameter t_p representing the range of the propagation of carry. We increase t_p from 1 to $\lfloor \log n \rfloor - 1$. In each t_p, we calculate $p[2^{t_p} i, 2^{t_p}(i+1)]$ $(1 \leq i \leq \lfloor n/2^{t_p} \rfloor - 1)$ by setting $|p[2^{t_p} i, 2^{t_p}(i+1/2)]\rangle$ and $|p[2^{t_p}(i+1/2), 2^{t_p}(i+1)]\rangle$ as the control qubits in Toffoli gate in Figure 6.12a. These Toffoli gates are applied simultaneously in each t_p.

G-rounds (n Toffoli gates and $\log n$ Toffoli depth)

We calculate $|c_{2^k}\rangle$ $(k \in \mathbb{N})$ by using Eq. (6.6). We use a parameter t_g similar to the way we used it in P-rounds. We increase t_g from 1 to $\lfloor \log n \rfloor$. In each t_g, we calculate $g[2^{t_g} i, 2^{t_g}(i+1)]$ $(0 \leq i \leq \lfloor n/2^{t_g} \rfloor - 1)$ by setting $|c_{2^{t_g} i + 2^{t_g - 1}}\rangle$ and $|p[2^{t_g}(i+1/2), 2^{t_g}(i+1)]\rangle$ as the control qubits and $|c_{2^{t_g}(i+1)}\rangle$ as the target qubit in Toffoli gate in Figure 6.12b. These Toffoli gates are applied simultaneously in each t_g. Moreover, G-rounds with t_g can be run in parallel with former P-rounds with $t_g + 1$.

C-rounds (n Toffoli gates and $\log n$ Toffoli depth)

We calculate all carries $|c\rangle$ by using Eq. (6.6). We use a parameter t_c similar to the way we used it in P-rounds. We decrease t_c from $\lfloor \log(2n/3) \rfloor$ to 1. In each t_c, we calculate $|c_{2^{t_c} i + 2^{t_c - 1}}\rangle$ $\left(1 \leq i \leq \left\lfloor \left(n - 2^{t_c - 1}\right)/2^{t_c} \right\rfloor - 1\right)$ by setting $|c_{2^{t_c} i}\rangle$ and $|p\left[2^{t_c} i, 2^{t_c} i + 2^{t_c - 1}\right]\rangle$ as the control qubits and $|c_{2^{t_c} i + 2^{t_c - 1}}\rangle$ as the target qubit in Toffoli gate in Figure 6.12b. These Toffoli gates are applied simultaneously in each t_c.

Inverse P-rounds (n Toffoli gates and $\log n$ Toffoli depth)

We apply the same gates as P-rounds in reverse order. Rounds with t_p can be run in parallel with former C-round with $t_p + 1$.

Calculating $|a+b\rangle$ (n CNOT gates)

We calculate $(a+b)_i$ $(0 \leq i \leq n-2)$ on $|b_i\rangle$. We apply CNOT gates with the control qubit of $|c_{i+1}\rangle$ and the target qubit of $|b_{i+1}\rangle$. These CNOT gates are applied simultaneously.

Erasing Carry ($5n$ Toffoli gates, $2n$ CNOT gates, and $2\log n$ Toffoli depth)

We erase all carries by applying the inverse circuit of $a + (2^n - 1 - a - b)$ on the lower $n-1$ bits, as shown in Figure 6.14. We apply gates before P-rounds and after inverse Initialization to erase carries. We call these gates PE-rounds and inverse PE-rounds, respectively.

We now show the example circuit of Draper et al.'s carry-lookahead adder as given in Figure 6.15. In this example, we define a and b as 6-bit values, and we calculate $|a\rangle|b\rangle \rightarrow |a\rangle|a+b\rangle$. In Figure 6.15, in constrast to Figure 6.10, qubits are sorted from low order to high order.

T-count Minimization of a Carry-lookahead Adder

Thapliyal et al. [111] proposed T-count minimization by using relative-phase Toffoli gates. In the carry-lookahead adder, as in many circuits, we must clean our ancilla qubits, returning them to a known, disentangled state, typically $|0\rangle$. In this case, we can reduce our cost by measuring the ancilla on IGRT, assuming the cost of measurement is small.

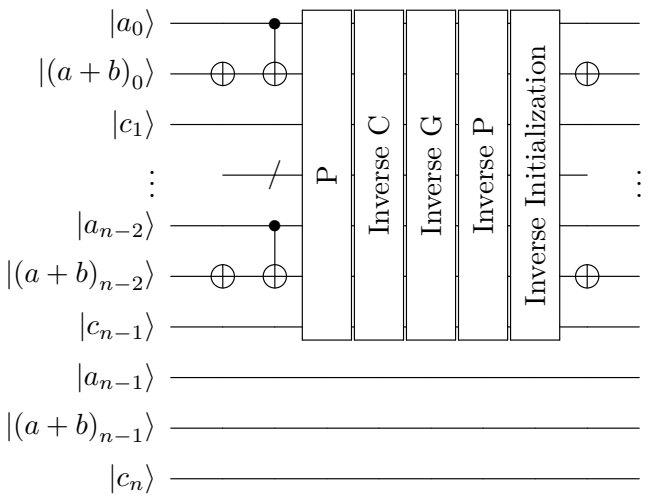

Fig. 6.14: Erasing $|c\rangle$. We apply gates only on the lower $n-1$ qubits of $|a\rangle$, $|b\rangle$, and $|c\rangle$. We apply the same gates in omitted qubits $|a_i\rangle$, $|(a+b)_i\rangle$, and $|c_{i+1}\rangle$. The P-rounds and inverse C-rounds can be run in parallel, as can the inverse G-rounds and inverse P-rounds. We define PE-rounds as the gates before P-rounds. Moreover, we define inverse PE-rounds as the gates after inverse Initialization.

By using GRT and IGRT, the number of T gates is reduced compared to using the only ST.

Thapliyal et al. proposed two constructions. The first construction replaced Toffoli gates in Initialization and P-rounds with GRT and Toffoli gates in the inverse rounds with IGRT. Other Toffoli gates are replaced with ST. Thapliyal et al. call this construction qubit-optimize. The number of qubits is $4n$, and the number of T gates is $40n$.

The second construction replaced all Toffoli gates into GRT or IGRT by increasing ancilla qubits. Thapliyal et al. call this construction T-optimize. Specifically, we replace Toffoli gates in Initialization, P-rounds, and the inverse of them as the first construction. Moreover, we replace Toffoli gates in G-rounds and C-rounds by the pair of GRT and IGRT as in Figure 6.16. We call these gates PGRT, where P is an abbreviation of "pair". In this construction, Thapliyal et al. claim that the number of qubits is $6n$, and the number of T gates is $20n$. However, we recalculated these results, and our results differ from the results in [111]. In our result, the number of qubits is $4.5n$, and the number of T gates is $28n$. The difference in the number of qubits occurs from our method for preparing ancilla qubits. Thapliyal et al. prepare new ancilla qubits for G-rounds and C-rounds, respectively, while recycling ancilla qubits for P-rounds. We apply this to G-rounds and C-rounds similarly.

6.3 First-Level Optimization: Our Construction of a Controlled Modular Adder

In this section, we explain the first-level optimization on the original construction [113]. In general construction, a comparator has about 1/2 the depth of a carry-lookahead adder. Now, a controlled modular adder is composed of two comparators and one adder in the general construction. Thus, by constructing a carry-lookahead adder using the same general construction, the depth is about the same as the two adders. In the original

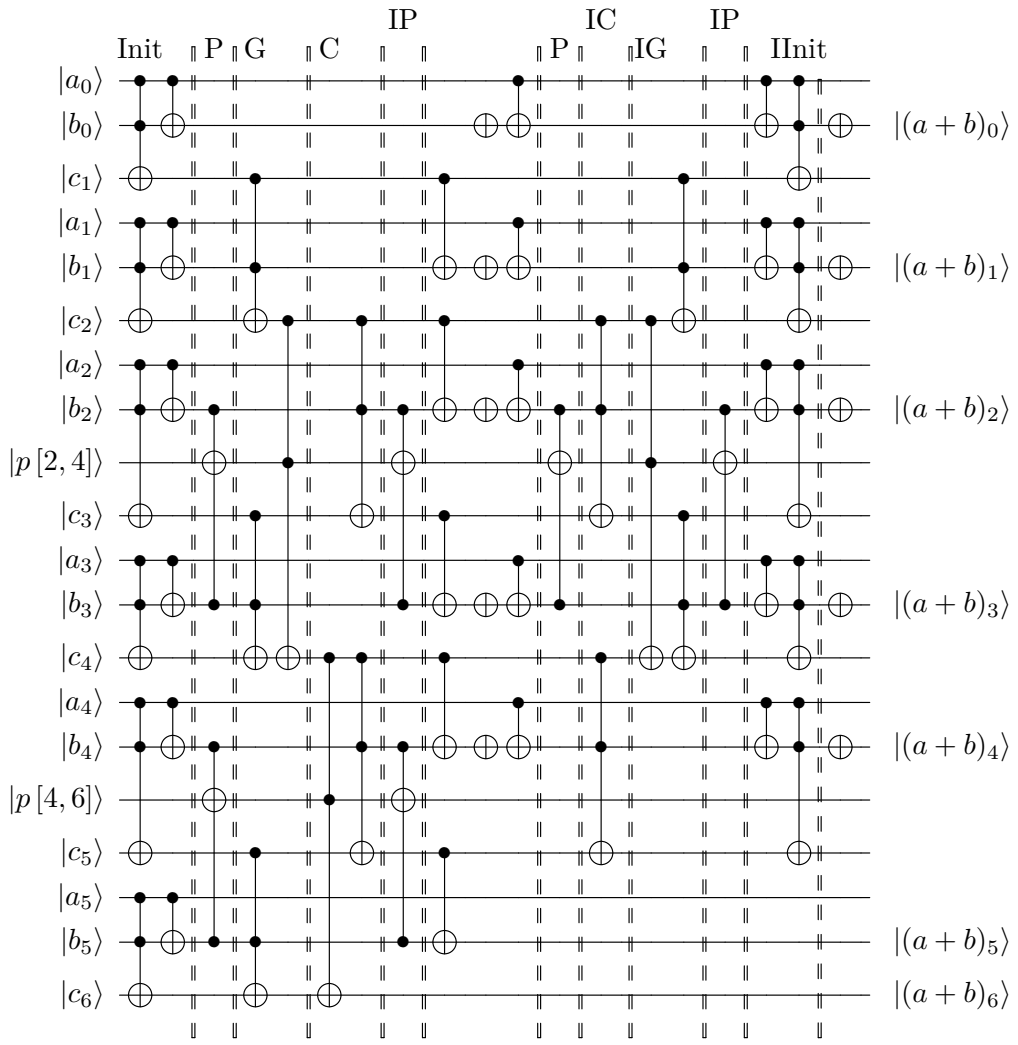

Fig. 6.15: An example of Draper et al.'s carry-lookahead adder. This circuit adds two 6-bit numbers a and b, namely $|a\rangle|b\rangle \rightarrow |a\rangle|a+b\rangle$. In this figure, we sort qubits from the lowest qubits to the highest qubits. The labels at the top are the rounds, including Toffoli gates. Init means Initialization. IP, IC, IG, and IInit means Inverse P-rounds, Inverse C-rounds, Inverse G-rounds, and Inverse Initialization.

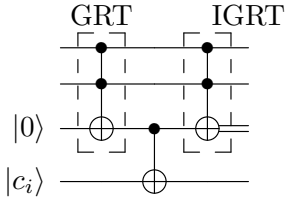

Fig. 6.16: Replacing Toffoli gates in G-rounds and C-rounds on a T-optimize carry-lookahead adder. We call this decomposition PGRT. We replace the first Toffoli gate with GRT and the second Toffoli gate with IGRT. The third qubit is an ancilla qubit. This qubit is measured in IGRT and be $|0\rangle$ after running PGRT.

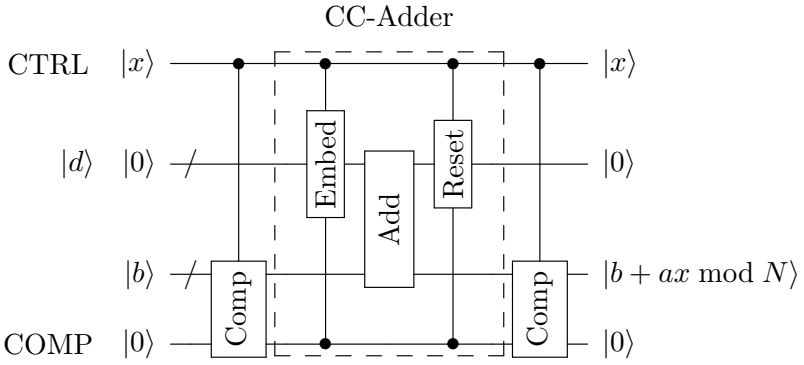

Fig. 6.17: Our construction of a controlled modular adder based on Figure 6.10. A CC-adder is constructed by embedding, an adder, and resetting. Then, we add the second register $|d\rangle$ as an n-qubit ancilla for embedding the value based on CTRL. The carry register $|c\rangle$ with n qubits and the p-function register $|p\rangle$ with n qubits are not represented in this figure for visibility. In a C-comparator, we do not use the second register. In total, our controlled modular adder requires $4n + 2$ qubits.

Table 6.1: Gate count and depth of our proposed controlled modular adder. The breakdown of this is shown in Table 6.2.

	Count		Depth	
Operation	Toffoli	CNOT	Toffoli	CNOT
C-comparator (twice)	$4n$	n	$2\log n$	$O(1)$
CC-adder	$9.5n$	$4.75n$	$4\log n$	$2\log n$
Total	$17.5n$	$6.75n$	$8\log n$	$2\log n$

construction, we use three adders. Thus, we use only 2/3 of the depth of the original construction. We then need to give the construction of

- C-comparator (Section 6.3.1)
- CC-adder (Section 6.3.2)

on a carry-lookahead adder. We do not decompose Toffoli gates in this construction because the decomposition of Toffoli gates is different in FTQ or NISQ computers, respectively. Thus, we leave Toffoli gates as they are, and we consider the decomposition of Toffoli gates in Section 6.4.

In our construction, we consider the classicality of a and N as described by Malkov and Saeedi [68] to realize higher efficiency. Moreover, we consider a C-comparator precisely that is not considered in the original construction. By doing these, we propose a circuit construction of a controlled modular adder.

Based on Figure 6.10, we construct our circuit as shown in Figure 6.17. We add the second n-qubit ancilla register for embedding value with CTRL. In addition to these registers, we use the carry register $|c\rangle$ with n qubits and the p-function register $|p\rangle$ with n qubits to realize the carry-lookahead adder, not represented in Figure 6.17. Thus, our controlled modular adder requires $4n + 2$ qubits. The number of gates and the depth is given in Table 6.1, and the breakdown of this is given in Table 6.2. We now explain our construction of the C-comparator and the CC-adder in Section 6.3.1 and 6.3.2, respectively. We then give the example circuit of our controlled modular adder in Section 6.3.3.

Table 6.2: Gate count and depth of our proposed controlled modular adder. We omit the rounds whose gate count is $O(1)$ and whose depth is $O(1)$.

		Count		Depth	
Operation	Rounds	Toffoli	CNOT	Toffoli	CNOT
C-comparator (twice)	Initialization	0	$0.5n$	0	$O(1)$
	P	n	0	$\}\log n$	0
	G	n	0		
	Inverse G	n	0	$\}\log n$	0
	Inverse P	n	0		
	Inverse Initialization	0	$0.5n$	0	$O(1)$
	Total	$4n$	n	$2\log n$	$O(1)$
CC-adder	Embedding	$O(1)$	$0.75n$	$O(1)$	$\log n$
	Initialization	$0.75n$	$0.75n$	$O(1)$	$O(1)$
	P	n	0	$\}\log n$	0
	G	n	0		
	C	n	0	$\}\log n$	0
	Inverse P	n	0		
	Calculating $\|a+b\rangle$	0	n	0	$O(1)$
	PE	0	$0.75n$	0	$O(1)$
	P	n	0	$\}\log n$	0
	Inverse C	n	0		
	Inverse G	n	0	$\}\log n$	0
	Inverse P	n	0		
	Inverse Initialization	$0.75n$	$0.75n$	$O(1)$	$O(1)$
	Resetting	$O(1)$	$0.75n$	$O(1)$	$\log n$
	Total	$9.5n$	$4.75n$	$4\log n$	$2\log n$
Total		$17.5n$	$6.75n$	$8\log n$	$2\log n$

6.3.1 Construction of a C-comparator

In a C-comparator, only COMP is changed, and other qubits do not change. Thus, to implement a C-comparator, it is sufficient that we add control operations only on the gates, including COMP, and we remain other gates.

In our construction of a controlled modular adder, we use two types of C-comparators. In the first C-comparator, we flip COMP if CTRL is 1 and $b \geq N - a$. In the final C-comparator, we flip COMP if CTRL is 1 and $b < a$. In both cases, we judge whether $b \geq d$ or $b < d$ with a classical value of d.

We construct these operations taking advantage of the classicality of d. The intuitive explanation of this operation is that we calculate $b + (2^n - d)$ and check whether there is an overflow in the n-th bit. Specifically,

$$b + (2^n - d) = 2^n + (b - d)$$

and there is an overflow when $b \geq d$. This construction is similar to previous constructions by Markov and Saeedi [68], but slightly different from them because our construction does not require X gates on $|b\rangle$. The number of gates and the depth is given in Table 6.1. The abstract construction of our C-comparator is given in Figure 6.18, and the example circuits are shown in Figure 6.20 and 6.22.

We now explain the construction of a C-comparator in more detail. In a C-comparator,

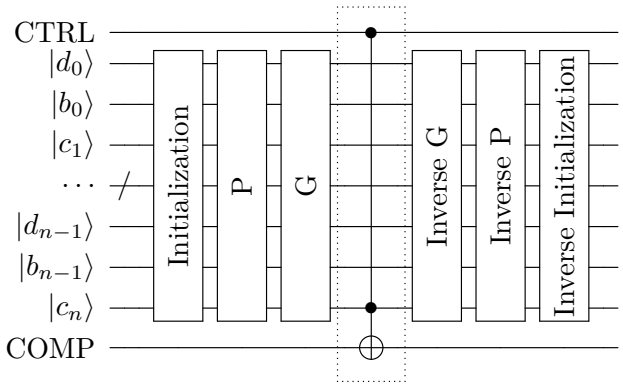

Fig. 6.18: Block-level view of our construction of a C-comparator. We sort qubits from the low-order qubits to the high-order qubits, top to bottom, in this figure. This circuit is symmetric about the Toffoli gate surrounded by a dotted box. $|c_i\rangle$ is given as $|0\rangle$ at the beginning of this circuit. Then, these qubits are cleared back to $|0\rangle$ after the computation. The example circuits are shown in Figure 6.20 or 6.22.

we judge whether $b \geq d$, where b is a quantum value and d is a classical value. We conduct this by calculating the carry out of the entire circuit $b + (2^n - d)$. Our construction is given as follows:

Initialization

If we conduct Initialization naively, we apply a Toffoli gate and a CNOT gate for each bit. However, the compilation of a quantum algorithm often requires compilation (selection of the sequence of gates) to be adapted to the specific classical values that are inputs to the overall algorithm. Because $2^n - d$ is a classical value, we can convert some Toffoli gates to CNOT gates and eliminate other gates. Then, we calculate each $(2^n - d)_i$ $(0 \leq i \leq n-1)$. If $(2^n - d)_i = 1$, we apply gates as follows:

1. CNOT gates with the control qubit $|b_i\rangle$ and the target qubit $|c_{i+1}\rangle$;
2. X gates with on $|b_i\rangle$.

These operations correspond to Toffoli gates or CNOT gates in the Initialization phase in Draper et al.'s construction, respectively.

P-rounds and G-rounds

We conduct P-rounds and G-rounds, similar to Draper et al.'s construction.

Writing result on the COMP qubit ($O(1)$ gates and $O(1)$ depth)

If we want to flip COMP when $b \geq d$, we apply Toffoli gates with the control qubits of CTRL and $|g[0,n]\rangle$, and with the target qubit of COMP. If we want to flip COMP when $b < d$, we apply Toffoli gates similarly to $b \geq d$, but we apply NOT gates on $|g[0,n]\rangle$ before and after the Toffoli gate.

Resetting qubits

We conduct inverse G-rounds and inverse P-rounds, similar to Draper et al.'s construction. Moreover, we conduct the inverse of our Initialization. Then, we reset all qubits except for COMP as the initial values.

6.3.2 Construction of a CC-adder

In a CC-adder, we embed values before and after an adder, similar to a C-adder [116]. Based on this construction, we apply optimization by considering the classicality of a and

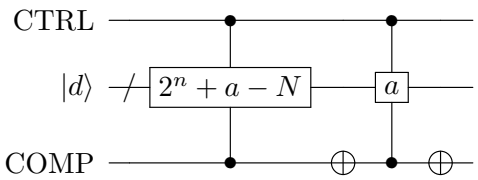

Fig. 6.19: Block-level diagram of the embedding circuit. We omit $|b\rangle$ in Figure 6.17. We embed $2^n + a - N$ or a on $|d\rangle$ based on CTRL and COMP. The example circuit of the embedding is shown in Figure 6.21.

N. From this point forward, we mainly focus on embedding on $|d\rangle$. In a CC-adder, we conduct the following:

- If CTRL is 1 and COMP is 1, we add a and subtract N. This operation can be realized by adding $2^n + a - N$ and disregarding the calculation of a carry c_n.
- If CTRL is 1 and COMP is 0, we add a.
- Otherwise, we add no value.

Thus, the embedding is conducted as in Figure 6.19. The resetting is conducted by inverting the embedding circuit.

After embedding, we apply a standard adder. Then, we conduct two optimizations as follows:

- Disregarding gates including $|g\,[0, n]\rangle$
- Eliminating gates in Initialization where we know the control bit is 0

The number of gates and the depth is given in Table 6.1. Moreover, we give the example circuit of a CC-adder in Figure 6.21.

We now give the construction of a CC-adder in more detail. First, we explain the construction of embedding in more detail. We want to embed as follows:

- If CTRL is 1 and COMP is 1, we embed $2^n + a - N$.
- If CTRL is 1 and COMP is 0, we embed a.
- Otherwise, we embed no value.

Therefore, we embed on the second register on Figure 6.17 as follows:

- If CTRL is 1 and $(2^n + a - N)_i = a_i = 1$, i-th qubit is $|1\rangle$.
- If CTRL is 1, COMP is 1, $(2^n + a - N)_i = 1$, and $a_i = 0$, i-th qubit is $|1\rangle$.
- If CTRL is 1, COMP is 0, $(2^n + a - N)_i = 0$, and $a_i = 1$, i-th qubit is $|1\rangle$.
- Otherwise, we do nothing.

In the above condition, the values of $(2^n + a - N)_i$ and a_i are classical information, and CTRL and COMP are quantum information. Thus, embedding in the first condition can be realized by CNOT gates with the control qubit of CTRL. Moreover, embedding in the second and third conditions can be realized by Toffoli gates with the control qubits of CTRL and COMP. However, the set of i in each classical condition has no overlap. Therefore, once we embed one of i, we can embed the remaining value as CNOT gates. In each set, we have average $n/4$ elements requiring $n/4$ CNOT gates, $O(1)$ additional gates. Thus, these embedding can be implemented by $3n/4$ CNOT gates. Moreover, because we can run these simultaneously, embedding requires $\log n$ CNOT depth. The reset of embedding can be implemented similarly.

Next, we explain the optimization in an adder. In our calculation, there is no carry for

$g[0,n]$ whether we subtract $N-a$ or add a. Thus, we can disregard calculation of carry qubit $g[0,n]$. To realize this, we omit calculation of $p[i,n]$ and $g[i,n]$ $(i<n)$. Moreover, by using the classicality of a and N, we know that we embed no value in average $n/4$ qubits on the second register of Figure 6.17. In these qubits, we can omit Initialization, inverse Initialization, and CNOT gates with the control qubit of $|a_i\rangle$ and the target qubit of $|b_i\rangle$ in erasing carry. By considering these optimizations, we reduce $n/2$ Toffoli gates and $3n/4$ CNOT gates.

6.3.3 Example of Our Controlled Modular Adder

We show an example of a 6-bit controlled modular adder when $N=59$ and $a=37$. Circuits are given in Figures 6.20–6.22.

In these example figures, registers are shown with low-order qubits at the top, in contrast to Figure 6.17. In this subsection, the register $|b\rangle$ contains a quantum value.

The algorithm follows in this order:

1. Conduct a C-comparator with the control qubit CTRL. Compare $|b\rangle$ and $N-a=22$. If $b\geq 22$, flip COMP. This is implemented by adding $2^6-(N-a)=42$ and using the carry out.
2. Conduct a CC-adder. If both CTRL and COMP are 1, subtract $N-a=22$. This is implemented by adding $2^6-(N-a)=42$ without calculating carry c_6. If CTRL is 1 and COMP is 0, add $a=37$, otherwise, add no value.
3. Conduct a C-comparator with the control qubit CTRL. Compare $|b\rangle$ and $a=37$. If $b<37$, flip COMP. This is implemented by calculating carry of adding $2^6-a=27$.

These steps correspond to Figure 6.20, 6.21, and 6.22, respectively.

6.4 Second-Level Optimization: Constructing a Controlled Modular Adder for FTQ and NISQ

In this section, we explain our second-level optimization. We evaluate the computational cost for both FTQ on the logical layer and NISQ, focusing on the decomposition of Toffoli gates. At first, we minimize the depth of our controlled modular adder. Then, we propose a controlled modular adder that is more efficient than Van Meter and Itoh [113], called the original construction in this section. For FTQ computers, we minimize the number of T gates and the T-depth using Gidney's relative-phase Toffoli gates. For NISQ computers, we apply Maslov's relative-phase Toffoli gates with a small number of CNOT gates [70].

Moreover, we consider minimizing KQ. In NISQ computers, $\mathrm{KQ}_{\mathrm{CX}}$ is minimized by our construction. However, our construction for FTQ computers does not take into consideration the cost of distillation. Then we minimize KQ_T by taking into account the cost of distillation by finding the maximal number of T gates which should be run simultaneously.

In the following discussion, we disregard the rounds with $O(1)$ gates. This section explains the optimization for FTQ computers in Section 6.4.1 and the optimization for NISQ computers in Section 6.4.2.

6.4.1 Evaluating the Computational Cost of Controlled Modular Addition for FTQ computers on the Logical Layer

We now consider the optimal circuit for FTQ computers on the Logical layer. First, we minimize the number of T gates on our controlled modular adder. In our CC-adder, we

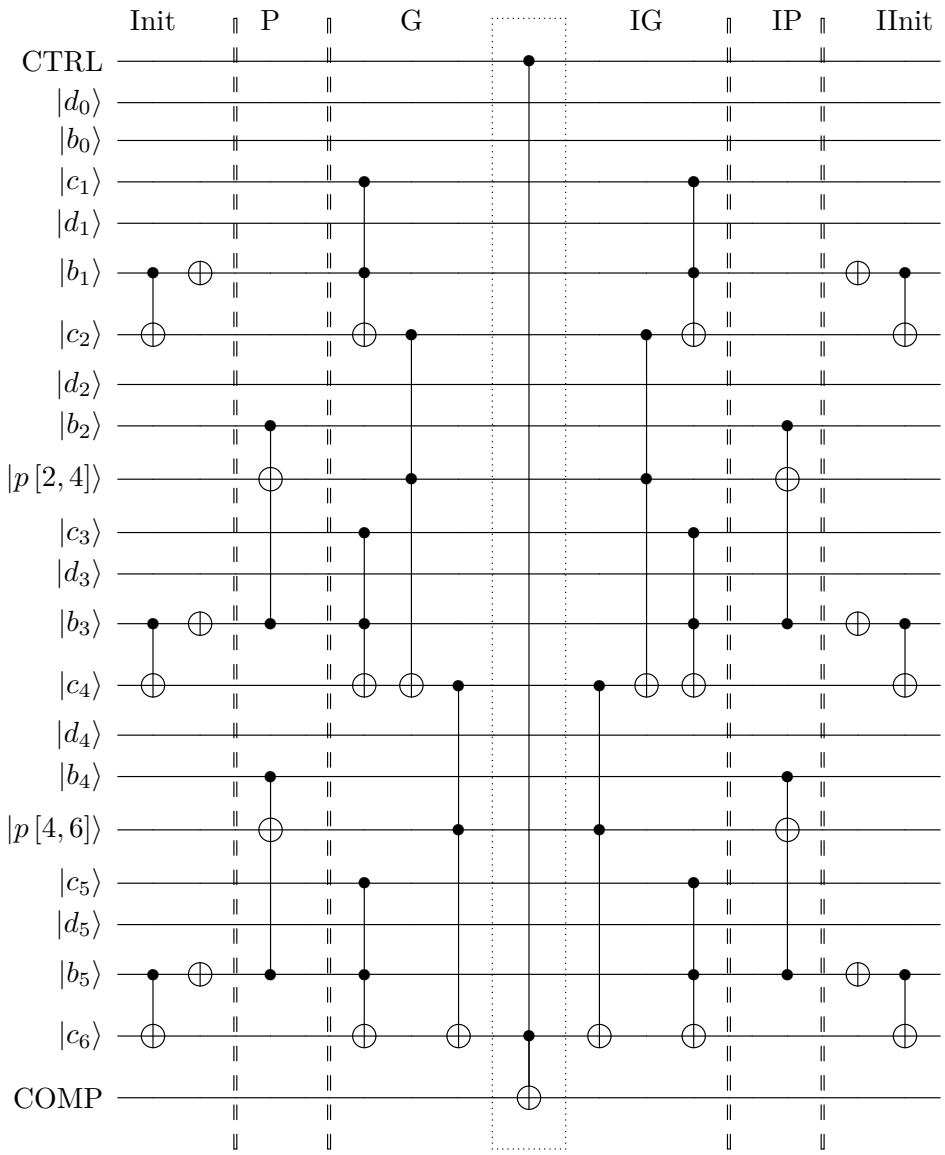

Fig. 6.20: An example circuit of the first C-comparator for flipping the COMP qubit if $b \geq 22$. To achieve this, We add $2^6 - 22 = 42 = 101010_2$ and use the COMP qubit as the carry out of the adder. The Init phase consists of pairs of gates, namely a CNOT and an X gate, on the second, fourth, and sixth groups of qubits, including $|d_i\rangle$, $|b_i\rangle$, and $|c_{i+1}\rangle$ from the lowest bit. This circuit is symmetric about the Toffoli gate surrounded by a dotted box. Init, IP, IG, and IInit mean Initialization, Inverse P-rounds, Inverse C-rounds, Inverse G-rounds, and Inverse Initialization.

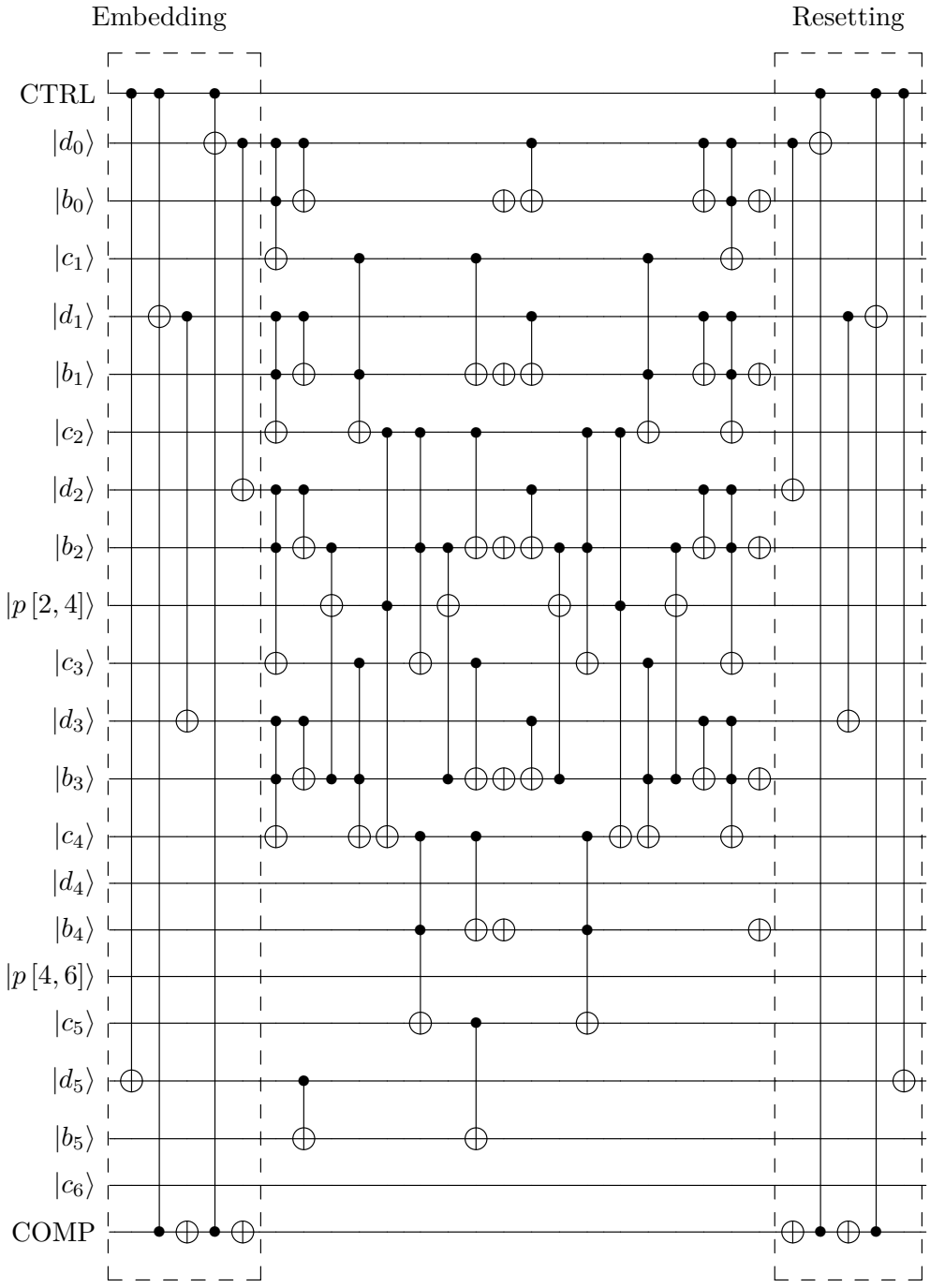

Fig. 6.21: An example of the CC-adder. If both CTRL and COMP are 1, we subtract $N - a = 22$. This is implemented by adding $2^6 - (N - a) = 42 = 101010_2$ without calculating carry c_6. If CTRL is 1 and COMP is 0, we add $a = 37 = 100101_2$. Based on these, we conduct embedding and resetting. The remaining part is an adder, and we omit the calculation of $p[i, 6]$ and $g[i, 6]$ $(i < 6)$.

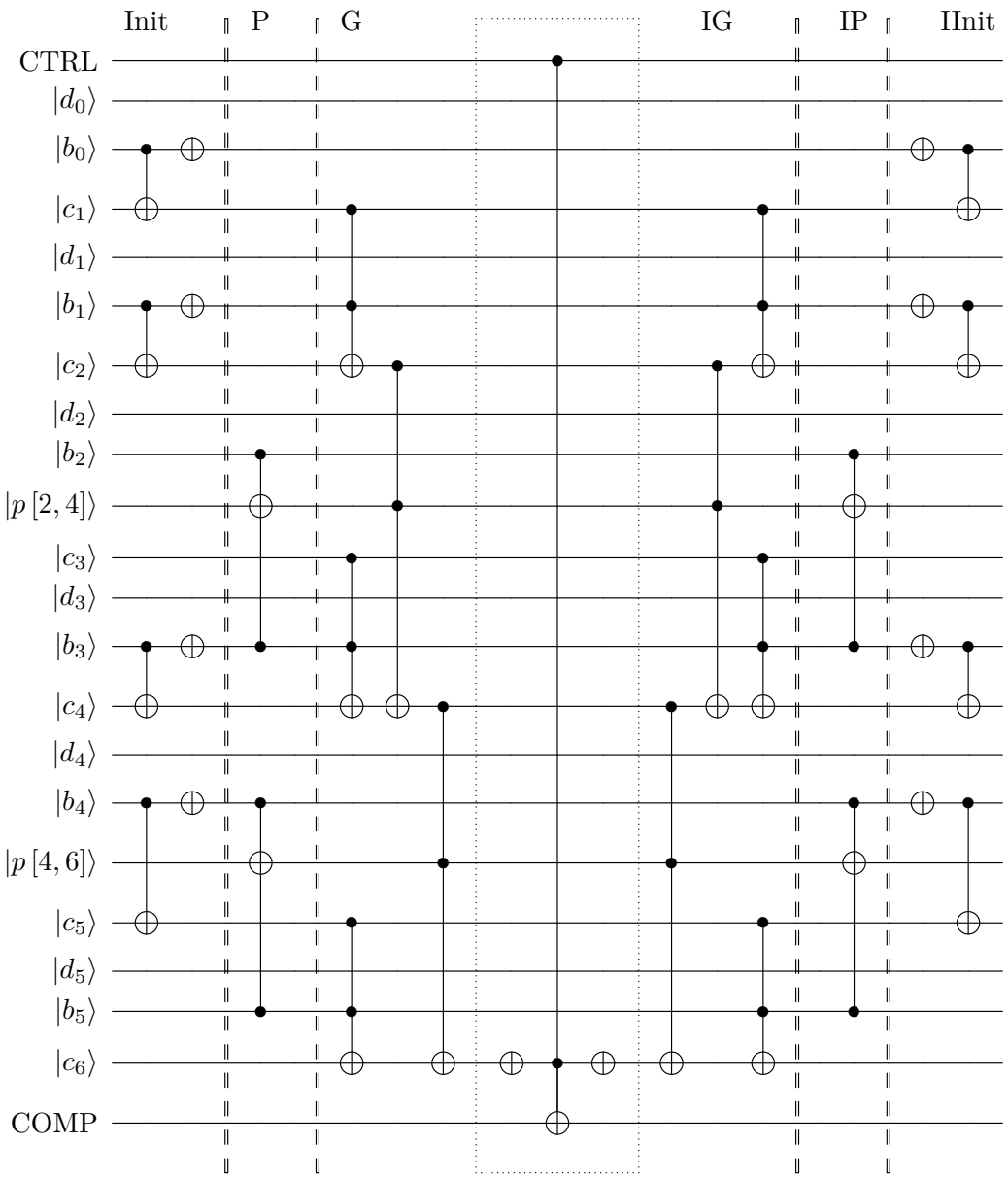

Fig. 6.22: An example of the last C-comparator. We flip the COMP qubit if $b < 37$. This is achieved by adding $2^6 - 37 = 27 = 011011_2$ and using the carry out. First, we apply pairs of gates, namely a CNOT and an X gate, on the first, second, fourth, and fifth groups of qubits. In contrast to Figure 6.20, we apply X gates before and after the center Toffoli gate. This circuit is symmetric about the Toffoli gate surrounded by a dotted box. Init, IP, IG, and IInit means Initialization, Inverse P-rounds, Inverse C-rounds, Inverse G-rounds, and Inverse Initialization.

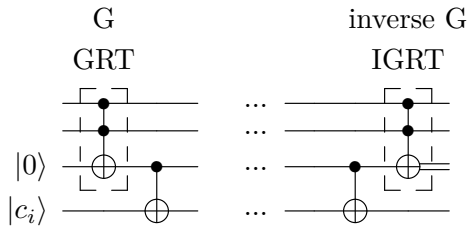

Fig. 6.23: Our construction of G-rounds and inverse G-rounds in a C-comparator. In Figure 6.16, we apply IGRT after the first CNOT gate immediately in G-rounds and inverse G-rounds. In our construction, we calculate the result of GRT in the third ancilla qubit and preserve this qubit until the corresponding Toffoli gate in inverse G-rounds. Then, we clear this ancilla qubit by IGRT.

Table 6.3: T-count of our controlled modular adder and prior work. The latter four constructions are based on our construction proposed in Section 6.3. The breakdown of the latter four constructions is shown in Table 6.5.

Construction	#comparators	#adders	Total T-count
Van Meter and Itoh [113]	0	3	$210n$
Draper et al. [26]	2	1	$122.5n$
Thapliyal et al. (qubit-optimize) [111]	2	1	$75n$
Thapliyal et al. (T-optimize) [111]	2	1	$51n$
Ours	2	1	$43n$

Table 6.4: T-depth of our controlled modular adder and prior work. The latter four constructions are based on our construction proposed in Section 6.3. The breakdown of the latter four constructions is shown in Table 6.6.

Construction	#comparators	#adders	Total T-count
Van Meter and Itoh [113]	0	3	$72 \log n$
Draper et al. [26]	2	1	$48 \log n$
Thapliyal et al. (qubit-optimize) [111]	2	1	$48 \log n$
Thapliyal et al. (T-optimize) [111]	2	1	$16 \log n$
Ours	2	1	$12 \log n$

adopt construction similar to Thapliyal et al., replacing Toffoli gates in G-rounds and C-rounds with PGRT especially. We minimize the number of T gates in a C-comparator. In a C-comparator, we replace Toffoli gates in Initialization and P-rounds with GRT and Toffoli gates in the inverse rounds with IGRT as the same as Thapliyal et al.'s construction. We focus on G-rounds. The abstract circuit of a C-comparator is shown in Figure 6.18, and we give example circuit as Figure 6.20 or 6.22. In these figures, Toffoli gates in G-rounds and inverse G-rounds are symmetric about the Toffoli gate surrounded by a dotted box. The control qubits of corresponding Toffoli gates in G-rounds and inverse G-rounds are the same. Therefore, we calculate with an accurate phase as Figure 6.23. This construction requires an additional n qubits to preserve. Fortunately, we do not use n qubits for $|d\rangle$ in Figure 6.17. Thus, we realize this construction without an overhead of qubits.

Table 6.5: The breakdown of Toffoli count and T-count of our controlled modular adder. Tof means the number of Toffoli gates in each round. Gate means the relative-phase Toffoli gates used in each round. Cost means the number of T gates in each relative-phase Toffoli gate. Count means T-count in each round. We omit the rounds whose T-count is $O(1)$. Inv means Inverse, C-comp means a C-comparator, CC-add means a CC-adder, and Init means Initialization.

			Toffoli Decomposition											
			Draper et al. [26]			Thapliyal et al. [111] (qubit-optimize)			Thapliyal et al. [111] (T-optimize)			**Ours**		
Operation	Rounds	Tof	gate	cost	count	gate	cost	count	gate	cost	count	gate	cost	count
C-comp (twice)	P	n	ST	7	$7n$	GRT	4	$4n$	GRT	4	$4n$	GRT	4	$4n$
	G	n	ST	7	$7n$	ST	7	$7n$	PGRT	4	$4n$	GRT	4	$4n$
	InvG	n	ST	7	$7n$	ST	7	$7n$	PGRT	4	$4n$	IGRT	0	0
	InvP	n	ST	7	$7n$	IGRT	0	0	IGRT	0	0	IGRT	0	0
	Total	$4n$	–	–	$28n$	–	–	$18n$	–	–	$12n$	–	–	$8n$
CC-add	Init	$0.75n$	ST	7	$5.25n$	GRT	4	$3n$	GRT	4	$3n$	GRT	4	$3n$
	P	n	ST	7	$7n$	GRT	4	$4n$	GRT	4	$4n$	GRT	4	$4n$
	G	n	ST	7	$7n$	ST	7	$7n$	PGRT	4	$4n$	PGRT	4	$4n$
	C	n	ST	7	$7n$	ST	7	$7n$	PGRT	4	$4n$	PGRT	4	$4n$
	InvP	n	ST	7	$7n$	IGRT	0	0	IGRT	0	0	IGRT	0	0
	P	n	ST	7	$7n$	GRT	4	$4n$	GRT	4	$4n$	GRT	4	$4n$
	InvC	n	ST	7	$7n$	ST	7	$7n$	PGRT	4	$4n$	PGRT	4	$4n$
	InvG	n	ST	7	$7n$	ST	7	$7n$	PGRT	4	$4n$	PGRT	4	$4n$
	InvP	n	ST	7	$7n$	IGRT	0	0	IGRT	0	0	IGRT	0	0
	InvInit	$0.75n$	ST	7	$5.25n$	IGRT	0	0	IGRT	0	0	IGRT	0	0
	Total	$9.5n$	–	–	$66.5n$	–	–	$39n$	–	–	$27n$	–	–	$27n$
Total		$17.5n$	–	–	$122.5n$	–	–	$75n$	–	–	$51n$	–	–	$43n$

Table 6.6: The breakdown of Toffoli count and T-depth of our controlled modular adder. Gate means the relative-phase Toffoli gates used in each round. Cost means the number of T gates in each relative-phase Toffoli gate. Depth means T-depth in each round. We omit the rounds whose T-depth is $O(1)$. Inv means Inverse, C-comp means a C-comparator, CC-add means a CC-adder, Init means Initialization, Embed means Embedding, and Reset means Resetting.

			Toffoli Decomposition											
			Draper et al. [26]			Thapliyal et al. [111] (qubit-optimize)			Thapliyal et al. [111] (T-optimize)			**Ours**		
Operation	Rounds		gate	cost	depth	gate	cost	depth	gate	cost	depth	gate	cost	depth
C-Comp (twice)	P	Toffoli	ST	6	} $6 \log n$	GRT	2	} $6 \log n$	GRT	2	} $2 \log n$	GRT	2	} $2 \log n$
	G	Toffoli	ST	6		ST	6		PGRT	2		GRT	2	
	InvG	Toffoli	ST	6		ST	6		PGRT	2		IGRT	0	} 0
	InvP	Toffoli	ST	6		IGRT	0		IGRT	0		IGRT	0	
	Total		–	–	$12 \log n$	–	–	$12 \log n$	–	–	$4 \log n$	–	–	$2 \log n$
CC-add	P	Toffoli	ST	6	} $6 \log n$	GRT	2	} $6 \log n$	GRT	2	} $2 \log n$	GRT	2	} $2 \log n$
	G	Toffoli	ST	6		ST	6		PGRT	2		PGRT	2	
	C	Toffoli	ST	6		ST	6		PGRT	2		PGRT	2	
	InvP	Toffoli	ST	6		IGRT	0		IGRT	0		IGRT	0	
	P	Toffoli	ST	6		GRT	2		GRT	2		GRT	2	
	InvC	Toffoli	ST	6		ST	6		PGRT	2		PGRT	2	
	InvG	Toffoli	ST	6		ST	6		PGRT	2		PGRT	2	
	InvP	Toffoli	ST	6		IGRT	0		IGRT	0		IGRT	0	
	Total		–	–	$24 \log n$	–	–	$24 \log n$	–	–	$8 \log n$	–	–	$8 \log n$
Total			–	–	$48 \log n$	–	–	$48 \log n$	–	–	$16 \log n$	–	–	$12 \log n$

The computational cost of our controlled modular adder is shown in Table 6.3, and 6.4. The breakdown of those based on our construction are shown in Table 6.5 and 6.6.

From Table 6.3 and 6.4, our construction is better in terms of both the number of T gates and T-depth. We now compare our circuit to the original construction.

First, we compare the number of T gates. The original construction requires $30n$ Toffoli gates implemented by ST requiring 7 T gates and $210n$ T gates in total. Thus, our construction requires 20% of the number of T gates of the original construction.

Next, we compare the T-depth. The original construction requires $12 \log n$ Toffoli depth implemented by ST requiring 6 T-depth and $72 \log n$ T gates in total. Thus, our construction requires 17% of the number of T-depth of the original construction.

We now focus on KQ of our controlled modular adder. In this circuit, we use $O(n)$ qubits and $O(\log n)$ depth, giving a KQ of $O(n \log n)$. However, we do not consider the computational costs for distillation in this calculation. We can trade space for time, with substantial flexibility, by allocating more qubits to ancilla "factories", corresponding to increasing the number of T gates that are in concurrent execution [106, 114]. For an accurate estimate of the cost, and to enable fair comparison with prior research, we must consider the T gate costs, including the space for distillation [51, 52]. Thus, we must consider distillation costs in the calculation of KQ similarly.

However, it is not easy to calculate computational costs for distillation precisely because the cost depends on many architecture-specific parameters. Instead of KQ, we define a new index KQ_T as the product of the number of logical qubits and the T-depth. Moreover, we define n_T as the T-width, the upper-bound of the number of T gates running simultaneously. We assume that we require a constant c_g logical qubits for the distillation step. By calculating n_T minimizing KQ_T, we reduce the computational cost of our controlled modular adder.

Our controlled modular adder uses $4n + 2$ qubits for calculation, as explained in Section 6.3. Also, we require ancilla qubits for running n_T T gates. Specifically, to run one T gate, we require one qubit $|Y\rangle$ for running S gates and c_g qubits for generating $|A\rangle$. Thus, when we run n_T T gates simultaneously, we use the following qubits:

- $|y\rangle$ (Contains $|Y\rangle$ states), n_T qubits
- $|g\rangle$ (Generates $|A\rangle$ states), $c_g n_T$ qubits

The number of qubits in $|y\rangle$ is given as n_T, because we consume one S gate in each T gate. Then, the number of qubits is

$$4n + (c_g + 1)n_T + 2. \tag{6.7}$$

We now calculate the T-depth of our controlled modular adder. To calculate the T-depth, we assume that we run GRT with the same timing, and each GRT has 2 T-depth from Figure 6.5. We focus on the parts of running simultaneously.

Except for Initialization, we run

- P-rounds and G-rounds simultaneously
- C-rounds and inverse P-rounds simultaneously
- P-rounds and inverse C-rounds simultaneously
- Inverse G-rounds and inverse P-rounds simultaneously

In the first and third steps, we run many T gates simultaneously at the start. Then, we run fewer T gates as the calculation progresses. In the second and fourth steps, we run only a few T gates simultaneously initially. Then, we run more T gates as the calculation progresses. Thus, there is a difference for T gates we can run simultaneously.

As noted in the above discussion, we define n_T as the upper-bound of the number of T gates running simultaneously, and we calculate T-depth based on n_T as in Figure 6.24.

There are parts where we can run more than n_T T gates in each round. However, by setting n_T, we separately run these T gates. Compared to this, in the parts having less than n_T T gates, we can run these T gates simultaneously.

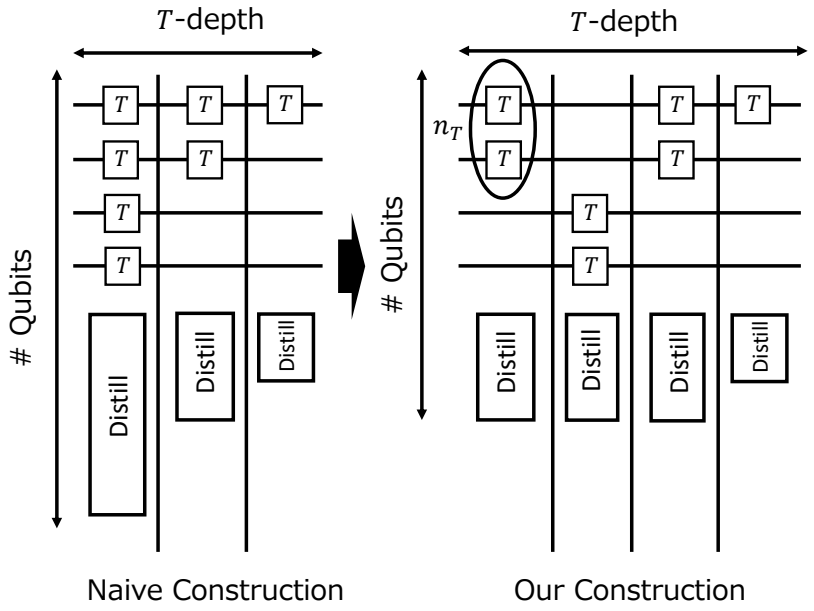

Fig. 6.24: Calculating T-depth. Distill means distillation circuits. In the naive construction, we run as many T gates as possible. In our construction, we restrict the upper-bound of the number of simultaneous T gates to n_T. When we reduce n_T, the total number of qubits is smaller, and the T-depth increases.

First, we consider the parts having fewer than n_T T gates, which happens when we run P-rounds and G-rounds simultaneously, C-rounds and inverse P-rounds simultaneously, P-rounds and inverse C-rounds simultaneously, and inverse G-rounds and inverse P-rounds simultaneously. In these rounds, if we have no restriction on running T gates, patterns are given as follows:

- In the first and the third cases, the number of T gates we can run simultaneously decreases by one half as the calculation progresses. Thus, in the latter part of the calculation, we run fewer than n_T T gates simultaneously. This part has T-depth $2\log n_T$ and n_T T gates in total.
- In the second and the fourth cases, the number of T gates we can run simultaneously doubles as the calculation progresses. Thus, in the former part of the calculation, we run less than n_T T gates simultaneously. This part has T-depth $2\log n_T$ and n_T T gates in total.

We have 6 parts, each with a small number of T gates, as follows:

- P-rounds and G-rounds in the first C-comparator
- P-rounds and G-rounds in the CC-adder
- C-rounds and inverse P-rounds in the CC-adder
- P-rounds and inverse C-rounds in the CC-adder
- Inverse G-rounds and inverse P-rounds in the CC-adder
- P-rounds and G-rounds in the final C-comparator

Thus, we consume $12\log n_T$ T-depth and $6n_T$ T gates in these.

Next, we consider the remaining parts. In these parts, we run T gates n_T each. The number of total T gates is $43n$ from Table 6.3, and we run $43n - 6n_T$ T gates.

Thus, T-depth of this part is given by

$$\frac{2\,(43n - 6n_T)}{n_T} = \frac{86n}{n_T} - 12.$$

In conclusion, T-depth is given by

$$\frac{86n}{n_T} + 12\log n_T - 12. \tag{6.8}$$

We now minimize KQ_T on n_T. From Eqs. (6.7) and (6.8), KQ_T is

$$(4n + (c_g + 1)n_T + 2)\left(\frac{86n}{n_T} + 12\log n_T - 12\right). \tag{6.9}$$

We minimize this on $n_T > 0$.

Letting the expression in Eq. (6.9) be $f\,(n_T)$, we see that

$$\frac{\mathrm{d}^2 f\,(n_T)}{\mathrm{d}n_T^2} > 0$$

in $n_T > 0$. Thus, $f\,(n_T)$ is a convex function and it is sufficient to search for only one optimal value of n_T. Then, the optimal value of n_T is given as

$$n_T = \sqrt{\frac{86}{3(c_g + 1)}}\frac{n}{\sqrt{\log n}} \tag{6.10}$$

Thus, $O\left(\dfrac{n}{\sqrt{\log n}}\right)$ T-width minimizes KQ_T. Substituting the value of n_T in Eq. (6.10) into Eq. (6.9),

$$\begin{aligned} &4n + (c_g + 1)n_T + 2 \sim 4n \\ &\frac{86n}{n_T} + 12\log n_T - 12 \sim 12\log n \end{aligned}$$

Therefore, the dominant term of KQ_T is $48n\log n$.

6.4.2 Optimization of a Controlled Modular Adder for NISQ Computers

We now propose the controlled modular adder reducing CNOT gates. We use relative-phase Toffoli gates with differences in phase as in Figures 6.7 and 6.8, proposed by Maslov [70]. By using these relative-phase Toffoli gates, we reduce the number of CNOT gates. We now consider which Toffoli gates can be replaced into relative-phase Toffoli gates.

First, we consider which Toffoli gates can be replaced in a C-comparator. The structure of a C-comparator is shown in Figure 6.18, and we give an example circuit as in Figure 6.20 or 6.22. In these figures, all Toffoli gates are symmetric about the Toffoli gate surrounded by a dotted box in the middle of the circuit. We replace the Toffoli gates on the left and right side by RT3 and IRT3. Thus, we can replace all of the Toffoli gates except for this middle one with RT3 or IRT3.

Table 6.7: CNOT count of our controlled modular adder and prior work. The latter four constructions are based on our construction proposed in Section 6.3. The breakdown of the latter four constructions is shown in Table 6.9.

Construction	#comparators	#adders	Total CNOT count
Van Meter and Itoh [113]	0	3	$184.5n$
Draper et al. [26]	2	1	$111.75n$
Thapliyal et al. (qubit-optimize) [111]	2	1	$88n$
Thapliyal et al. (T-optimize) [111]	2	1	$104n$
Ours	2	1	$64.75n$

Table 6.8: CNOT-depth and $\mathrm{KQ}_{\mathrm{CX}}$ of our controlled modular adder and prior work. The latter four constructions are based on our construction proposed in Section 6.3. The breakdown of the latter four constructions is shown in Table 6.10.

Construction	#qubits	The depth of the circuit	$\mathrm{KQ}_{\mathrm{CX}}$
Van Meter and Itoh [113]	$4n$	$78 \log n$	$312n \log n$
Draper et al. [26]	$4n$	$50 \log n$	$200n \log n$
Thapliyal et al. (qubit-optimize) [111]	$4n$	$50 \log n$	$200n \log n$
Thapliyal et al. (T-optimize) [111]	$4.5n$	$66 \log n$	$297n \log n$
Ours	$4n$	$30 \log n$	$120n \log n$

Next, we address which Toffoli gates can be replaced in a CC-adder. We find that those in P-rounds can be replaced by RT3 and those in inverse P-rounds by IRT3. The other Toffoli gates are used to calculate the value of carries. The values of the control bits are different between calculating a carry and erasing it. It would seem to rule out using anything but pure Toffoli gates. However, looking more closely, we see that the value of a carry changes at most once, namely when both control bits are $|1\rangle$. Thus, if we calculate correctly in other situations, we can calculate and clear carries correctly. RT4 satisfies this. Therefore, we can replace Toffoli gates by RT4 in the Initialization, G-rounds, and C-rounds, and we can replace Toffoli gates by IRT4 in the inverse rounds.

As a result, the cost of our controlled modular adder is shown in Table 6.7 and 6.8. The breakdown of those based on our construction are shown in Table 6.9 and 6.10.

From Table 6.7 and 6.8, our construction is better in terms of both the number of CNOT gates and CNOT-depth. We now compare our circuit to the original construction.

First, we compare the CNOT count. Our construction requires $64.75n$ CNOT gates. The original construction requires $30n$ Toffoli gates implemented by ST using 6 CNOT gates, and we use an additional $4.5n$ CNOT gates in embedding or resetting. Thus, the original construction requires $184.5n$ CNOT gates in total. Therefore, our construction reduces the number of CNOT gates to only 35% of the original.

Next, we compare CNOT-depth. Our construction requires $30 \log n$ CNOT-depth. The original construction requires $12 \log n$ Toffoli depth implemented by ST requiring 6 CNOT-depth, and we require $6 \log n$ CNOT-depth for the embedding step. Thus, the original construction requires $78 \log n$ CNOT-depth. Therefore, our construction requires the only 38% of the CNOT-depth of the original construction. We now focus on KQ by using $\mathrm{KQ}_{\mathrm{CX}}$, defined as the product of the number of qubits and CNOT-depth. Then, our construction requires the only 38% of the $\mathrm{KQ}_{\mathrm{CX}}$ of the original construction.

Table 6.9: The breakdown of the Toffoli and CNOT counts of our controlled modular adder. Gate indicates the type of relative-phase Toffoli gates used in each round. Cost means the number of CNOT gates in each relative-phase Toffoli gate. Count means the CNOT count in each round. We do not show rounds whose CNOT count is $O(1)$. Inv indicates an inverse, C-comp is a C-comparator, CC-add means a CC-adder, Init represents an initialization, Embed indicates an embedding, Calc is a calculation of $|a+b\rangle$, and Reset is a resetting.

				Toffoli Decomposition											
				Draper et al. [26]			Thapliyal et al. [111] (qubit-optimize)			Thapliyal et al. [111] (T-optimize)			**Ours**		
Operation	Rounds			gate	cost	count	gate	cost	count	gate	cost	count	gate	cost	count
C-Comp (twice)	Init	CNOT	$0.5n$	–	–	$0.5n$	–	–	$0.5n$	–	–	$0.5n$	–	–	$0.5n$
	P	Toffoli	n	ST	6	$6n$	GRT	6	$6n$	GRT	6	$6n$	RT3	3	$3n$
	G	Toffoli	n	ST	6	$6n$	ST	6	$6n$	PGRT	8	$8n$	RT3	3	$3n$
	InvG	Toffoli	n	ST	6	$6n$	ST	6	$6n$	PGRT	8	$8n$	IRT3	3	$3n$
	InvP	Toffoli	n	ST	6	$6n$	IGRT	1	n	IGRT	1	n	IRT3	3	$3n$
	InvInit	CNOT	$0.5n$	–	–	$0.5n$	–	–	$0.5n$	–	–	$0.5n$	–	–	$0.5n$
	Total			–	–	$25n$	–	–	$20n$	–	–	$24n$	–	–	$13n$
CC-add	Embed	CNOT	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$
	Init	Toffoli	$0.75n$	ST	6	$4.5n$	GRT	6	$4.5n$	GRT	6	$4.5n$	RT4	4	$3n$
		CNOT	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$
	P	Toffoli	n	ST	6	$6n$	GRT	6	$6n$	GRT	6	$6n$	RT3	3	$3n$
	G	Toffoli	n	ST	6	$6n$	ST	6	$6n$	PGRT	8	$8n$	RT4	4	$4n$
	C	Toffoli	n	ST	6	$6n$	ST	6	$6n$	PGRT	8	$8n$	RT4	4	$4n$
	InvP	Toffoli	n	ST	6	$6n$	IGRT	1	n	IGRT	1	n	IRT3	3	$3n$
	Calc	CNOT	n	–	–	n	–	–	n	–	–	n	–	–	n
	PE	CNOT	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$
	P	Toffoli	n	ST	6	$6n$	GRT	6	$6n$	GRT	6	$6n$	RT3	3	$3n$
	InvC	Toffoli	n	ST	6	$6n$	ST	6	$6n$	PGRT	8	$8n$	IRT4	4	$4n$
	InvG	Toffoli	n	ST	6	$6n$	ST	6	$6n$	PGRT	8	$8n$	IRT4	4	$4n$
	InvP	Toffoli	n	ST	6	$6n$	IGRT	1	n	IGRT	1	n	IRT3	3	$3n$
	InvInit	Toffoli	$0.75n$	ST	6	$4.5n$	IGRT	1	$0.75n$	IGRT	1	$0.75n$	RT4	4	$3n$
		CNOT	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$
	Reset	CNOT	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$	–	–	$0.75n$
	Total			–	–	$61.75n$	–	–	$48n$	–	–	$56n$	–	–	$38.75n$
Total				–	–	$111.75n$	–	–	$88n$	–	–	$104n$	–	–	$64.75n$

Table 6.10: The breakdown of Toffoli count and CNOT-depth of our controlled modular adder. Gate indicates the type of relative-phase Toffoli gate used in each round. Cost is the number of CNOT gates in each relative-phase Toffoli gate. Depth is the CNOT-depth in each round. We omit the rounds whose CNOT-depth is $O(1)$. Inv is an inverse, C-comp is a C-comparator, CC-add indicates a CC-adder, Init is an initialization, Embed represents an embedding, and Reset is a resetting.

			Toffoli Decomposition											
			Draper et al. [26]			Thapliyal et al. [111] (qubit-optimize)			Thapliyal et al. [111] (T-optimize)			**Ours**		
Operation	Rounds		gate	cost	depth	gate	cost	depth	gate	cost	depth	gate	cost	depth
C-Comp (twice)	P	Toffoli	ST	6	$\}6\log n$	GRT	6	$\}6\log n$	GRT	6	$\}8\log n$	RT3	3	$\}3\log n$
	G	Toffoli $\}\log n$	ST	6		ST	6		PGRT	8		RT3	3	
	InvG	Toffoli	ST	6	$\}6\log n$	ST	6	$\}6\log n$	PGRT	8	$\}8\log n$	IRT3	3	$\}3\log n$
	InvP	Toffoli $\}\log n$	ST	6		IGRT	1		IGRT	1		IRT3	3	
	Total		–	–	$12\log n$	–	–	$12\log n$	–	–	$16\log n$	–	–	$6\log n$
CC-add	Embed	CNOT $\log n$	–	–	$\log n$	–	–	$\log n$	–	–	$\log n$	–	–	$\log n$
	P	Toffoli	ST	6	$\}6\log n$	GRT	6	$\}6\log n$	GRT	6	$\}8\log n$	RT3	3	$\}4\log n$
	G	Toffoli $\}\log n$	ST	6		ST	6		PGRT	8		RT4	4	
	C	Toffoli	ST	6	$\}6\log n$	ST	6	$\}6\log n$	PGRT	8	$\}8\log n$	RT4	4	$\}4\log n$
	InvP	Toffoli $\}\log n$	ST	6		IGRT	1		IGRT	1		IRT3	3	
	P	Toffoli	ST	6	$\}6\log n$	GRT	6	$\}6\log n$	GRT	6	$\}8\log n$	RT3	3	$\}4\log n$
	InvC	Toffoli $\}\log n$	ST	6		ST	6		PGRT	8		IRT4	4	
	InvG	Toffoli	ST	6	$\}6\log n$	ST	6	$\}6\log n$	PGRT	8	$\}8\log n$	IRT4	4	$\}4\log n$
	InvP	Toffoli $\}\log n$	ST	6		IGRT	1		IGRT	1		IRT3	3	
	Reset	CNOT $\log n$	–	–	$\log n$	–	–	$\log n$	–	–	$\log n$	–	–	$\log n$
	Total		–	–	$26\log n$	–	–	$26\log n$	–	–	$34\log n$	–	–	$18\log n$
Total #qubits			–	–	$50\log n$	–	–	$50\log n$	–	–	$66\log n$	–	–	$30\log n$
$\mathrm{KQ}_{\mathrm{CX}}$			–	–	$4n$	–	–	$4n$	–	–	$4.5n$	–	–	$4n$
			–	–	$200n\log n$	–	–	$200n\log n$	–	–	$297n\log n$	–	–	$120n\log n$

6.5 Security of RSA Scheme against FTQ computers

In this section, we evaluate the security of the RSA scheme against FTQ computers. We evaluate the required time for modular exponentiation similar to the previous research [52]. Moreover, we estimate the period during which our construction breaks the RSA scheme by focusing on the executable depth.

Before evaluating the computational cost, we discuss the ancilla qubits for modular exponentiation. As indicated in Section 2.2.3, the following qubits are required for constructing modular exponentiation:

- n qubits for multipliers
- $1.5n$ qubits for the exponent

Only one qubit is required for an exponent using a qubit recycling technique [69]. Thus, we require $5n + O(1)$ qubits to construct modular exponentiation based on our construction.

We now evaluate the computational time of our construction based on the previous analysis [52]. The Toffoli-depth of the previous construction is $16n^2 \log n$, and this corresponds to $96n^2 \log n$ T-depth. It should be noted that the original construction reduces the number of controlled modular adders by the following optimizations:

- Summarizing two controlled modular adders
- Allowing approximate calculation

These optimizations will require more consideration to apply our construction in a future study; we evaluate our construction without these optimizations.

We then estimate the period during which our construction breaks the RSA scheme. From the IBM roadmap [49], the number of qubits increases as follows:

- **2019**: 27 qubits (released)
- **2020**: 65 qubits (released)
- **2021**: 127 qubits
- **2022**: 433 qubits
- **2023**: 1121 qubits

Following the plan developed by IBM, the number of qubits will double per year, and the number of qubits may reach 65×2^t in t years. However, an accurate computation is not realized on all qubits, and the number of qubits is not the only index representing the machine power of quantum computers. To evaluate the machine power of quantum computers, the quantum volume (QV) [20] has been adopted. In brief, QV is defined by $2^{\min(m,d)}$, where m and d are the numbers of qubits and the CNOT-depth of a quantum circuit based on most accurate calculations. The correctness is verified by solving the heavy output generation problem [1] on each machine. In this experiment, the heavy output is defined as the 2^{m-1} outputs with high probability. We regard a quantum computation as successful when the machine outputs a heavy output with a probability of 2/3. In an experiment conducted by IBM, m and d are set to the same values. Here, a 27 qubit machine from IBM achieves an almost correct calculation when $m = d = 6$ [53], and the machine has a QV of 64.

Currently, some of the qubits are used correctly in the current machines. As technology improves, all qubits may be used correctly. The executable depth will also develop toward increasing the number of qubits. However, the executable depth, especially the executable T-depth, will be limited because it requires much CNOT-depth.

Based on these estimation, we now describe an assumption based on the executable

depth in estimating the period when the RSA scheme is broken. First, we assume that the executable CNOT-depth is the same as the number of qubits because quantum computers' current development focuses on maximizing QV. Then, the executable CNOT-depth is 65×2^t in t years later. Next, we assume that CNOT-depth required for generating one T gate is 12. In the previous result [52], the distillation circuit shown in Figure 6.4 requires 6 time steps, namely 5 CNOT-depth and 1 measurement. Similar to the previous result, we deal with these gates similarly. Moreover, the previous study employs the distillation circuit twice for one T gate. Based on the above discussion, the CNOT-depth required for generating one T gate is 12.

We now review the previous construction. In the 1024-bit RSA scheme, the previous construction uses 1.01×10^9 T-depth and requires 1.81 days to break the 1024-bit RSA scheme, and 1.01×10^9 T-depth corresponds to 1.21×10^{10} CNOT-depth. This CNOT-depth is realized 27.5 years later based on our assumption. Moreover, in the 2048-bit RSA scheme, the previous construction uses 4.43×10^9 T-depth and 5.32×10^{10} CNOT-depth. The required time is

$$1.81 \times \frac{4.43 \times 10^9}{1.01 \times 10^9} = 7.94 \text{ [days]}.$$

Moreover, this CNOT-depth is realized 29.6 years later based on our assumption.

We now evaluate the security of the RSA scheme based on our construction. We consume $12 \log n$ T-depth in a controlled modular adder. Thus, $3n^2$ controlled modular adders are called, and the total T-depth is $36n^2 \log n$.

First, we evaluate the security on a 1024-bit RSA scheme, namely, $n = 1024$. When $n = 1024$, the T-depth is $36n^2 \log n = 3.77 \times 10^8$. Thus, the required time is

$$1.81 \times \frac{3.77 \times 10^8}{1.01 \times 10^9} = 0.68 \text{ [days]},$$

which corresponds to 16.2 hours. Now, CNOT-depth is 4.52×10^9, and this depth will be realized 26.1 years later, which is 1.4 years faster than the original construction. Thus, the 1024-bit RSA scheme can be broken in 2046.

Next, we evaluate the security on a 2048-bit RSA scheme, namely, $n = 2048$. When $n = 2048$, the T-depth is $36n^2 \log n = 1.66 \times 10^9$. Thus, the required time is

$$1.81 \times \frac{1.66 \times 10^9}{1.01 \times 10^9} = 2.97 \text{ [days]}.$$

Now, CNOT-depth is 1.99×10^{10}, and this depth will be realized 28.2 years later, which is 1.4 years faster than the original construction. Thus, the 2048-bit RSA scheme can be broken in 2048.

6.6 Conclusion and Future Work

In this study, we proposed a method for optimizing a controlled modular adder based on a carry-lookahead adder [26] and Van Meter-Itoh's construction [113]. First, we showed that the general construction given in Figure 6.10 has approximately 2/3 of the depth of the original construction. We then constructed a more efficient circuit. We evaluated the computational cost for use in FTQ computing. Then, we showed that our circuit requires the only 20% of the T gates and 17% of the T-depth of the original. Moreover, we showed that our circuit achieves its minimum KQ_T when we run $\Theta\left(\frac{n}{\sqrt{\log n}}\right)$ T gates simultaneously. We also proposed an efficient circuit for use in the NISQ era. We showed

that our circuit requires the only 35% of the CNOT gates and 38% of the CNOT-depth of the original. Finally, we evaluated the security of the RSA scheme against FTQ computers. We estimated that the 2048-bit RSA scheme can be broken 28.2 years later based on IBM's plan [49] which is 1.4 years earlier than Van Meter-Itoh's construction.

In this study, we focused on optimizing the Toffoli gates using relative-phase Toffoli gates. However, in previous studies [75, 110], other researchers have used gates such as Fredkin and Peres gates. These gates may also be simplified by replacing them with relative-phase gates. Thus, we expect that those circuits will also show an improvement when applying these techniques.

We considered only a single controlled modular adder and evaluated the security of the RSA scheme. However, to realize a more efficient circuit, we must consider more optimization to construct a controlled modular multiplier and modular exponentiation. Mainly, more optimization is applied in the previous constructions [35, 113]. With these constructions, the following optimizations are adopted:

- A summarization of many controlled modular adders
- The allowance of an approximate calculation

The first optimization, called window arithmetic [34], can be applied to our construction with a small modification. However, this optimization uses fewer T gates instead of consuming many more CNOT gates. In particular, the construction of Gidney and Ekerå gates reduces the Toffoli count to $0.3n^3 + 0.0005n^3 \log n$ and the Toffoli-depth to $500n^2 + n^2 \log n$. Then, the T-count is $O\left(n^3 \log n\right)$ and the T-depth is $O\left(n^2 \log n\right)$. The number of CNOT gates is $O\left(n^4\right)$ and the CNOT-depth is $O\left(n^3\right)$. Thus, the CNOT gate is the dominant cost in this construction. Therefore, to use window arithmetic, we must not reduce only the T gates while maintaining the number of CNOT gates. The second optimization is summarizing multiple modular arithmetic operations. The number of controlled modular additions decreases by such summarization, although small errors are incurred from the expected result. In particular, Gidney and Ekerå's construction is faster than our approach due to this approximate calculation. Thus, we should also consider decreasing the calculation costs by allowing an approximate calculation. Also, it is crucial to minimize the depth by reordering the gates [71, 88].

Our construction does not consider a linear nearest neighbor architecture of the quantum computers [18, 28, 45]. Thus, we will consider the appropriate architecture and additional costs incurred for our construction in the next stage.

Finally, we focused only on the logical layer of FTQ computers in this study. In a future study, we must consider the mapping to physical qubits and the distillation protocols.

Chapter 7

Conclusion

This thesis focuses on the security of public-key cryptosystems against side-channel and quantum attacks.

Chapter 3 described the security against side-channel attacks recovering the CRT-RSA secret keys from the correct sliding window leakage. We improved the side-channel attacks on the CRT-RSA scheme implemented using the left-to-right sliding window method. We proposed a new secret key recovery algorithm from the correct square-and-multiply sequences. As a result, we recover 21% of the CRT-RSA secret keys in our proposed method compared to 13% of the secret keys in the previous method for a 2048-bit CRT-RSA when the window size w is 5.

Next, Chapter 4 gave the security analysis against side-channel attacks recovering the CRT-RSA secret keys from noisy sliding window leakage. We proposed a new algorithm for recovering the CRT-RSA secret keys from noisy sliding window leakage. Moreover, we calculated the amount of error our method recovers for the CRT-RSA secret keys. Then, we verified that our method can be applied to 1.1% actual errors when the window size w is less than or equal to 4 based on numerical experiments.

Next, Chapter 5 gave the security analysis against side-channel attacks on Ring-LWE and Module-LWE based schemes. We provided the exact security analysis on the LPR cryptosystem against NTT leakage. We evaluated the number of recovered coefficients of the secret polynomial based on an erasure model for the multiplication. We assumed that we extract an input of multiplication with a probability of $1 - \delta$. Then, we analyzed the recovery algorithm of LPR secret keys and showed that our method recovers the LPR secret keys when $\delta \leq 0.78$ under the currently available computational power.

Finally, Chapter 6 described the security of the RSA scheme against quantum attacks. We proposed an efficient controlled modular adder, which is the core arithmetic of Shor's algorithm. We provided a more efficient construction than Van Meter-Itoh's approach based on a carry-lookahead adder [113] by using relative-phase Toffoli gates. Moreover, we described an efficient construction assuming actual devices, i.e., FTQ or NISQ computers. To minimize the computational cost in FTQ computers, we reduced the number of T gates in our controlled modular adder. Then, we showed that our circuit requires the only 20% of the T gates and 17% of the T-depth of the original. Moreover, taking the distillation into consideration, we found that we can minimize KQ_T by running $\Theta\left(n/\sqrt{\log n}\right)$ T gates simultaneously. We then proposed an efficient circuit for use in the NISQ era and demonstrated that our circuit requires the only 35% of the CNOT gates and 38% of the CNOT-depth of the original. Finally, we evaluated the security of the RSA scheme against FTQ computers. We estimated that the 2048-bit RSA scheme will be broken 28.2 years later based on the plan developed by IBM [49] which is 1.4 years earlier than Van Meter-Itoh's construction.

Acknowledgement

First, I would like to thank my supervisors Noboru Kunihiro and Tsuyoshi Takagi. During my bachelor's course, I did not understand how to conduct research. I also did not fully understand the importance of how to report my research. Noboru Kunihiro discussed the research topic with me and commented on my files, including papers and presentations. I always found new aspects after such discussions with him, and I deepened my understanding of this research dramatically. Moreover, I am extremely grateful for helping me at various conferences. At each conference I attended, I was troubled by how I should report the results effectively, but I was successful in all of my presentations based on his help. During my doctoral course, I was supervised by Tsuyoshi Takagi. Thanks to him, I had a wonderful time during my 3-year doctoral course. He guided me with many new aspects of my research through our discussions. Moreover, he introduced me to many presentations in the academic world and the field of business. By participating at many presentations and interchanging with the participants, I recognized the importance of my research and gained confidence. In addition to Noboru Kunihiro and Tsuyoshi Takagi, I am also profoundly grateful to Hirosuke Yamamoto. Hirosuke Yamamoto was the head of our laboratory during my bachelor's and master's courses. He welcomed me warmly from the beginning, and I happily conducted my research from then on.

Next, I would like to thank our laboratory members. In Kunihiro's laboratory, Atsushi Takayasu, Takashi Yamakawa, and Shuichi Katsumata taught me many things for improving my presentation. They gave me fruitful comments at seminars and improved my research with their advice. In particular, Shuichi Katsumata gave me guidance in how to make my presentation. I recall how he improved my PowerPoint presentation at my first international conference. Thanks to his help, I succeeded in my presentation. Also, I would like to thank Xiaoxuan Huang. I conducted a joint project with Xiaoxuan Huang and learned many new this during our collaboration. Through this joint project, I gained confidence in my research, and I have continued to conduct my research with a more positive mind. In Takagi's laboratory, I am also profoundly grateful to Yasuhiko Ikematsu, Yuntao Wang, and Hiroshi Onuki. Yasuhiko Ikematsu and Yuntao Wang planned all events in the laboratory. We discussed how to make our new laboratory more comfortable, which was very exciting for me. Through this experience, I learned some laboratory management skills. I would also like to thank Hiroshi Onuki. My seat is next to his, and we had many discussions whenever we were in the laboratory together. Because of his kindness, I did not hesitate to talk with him, and I learned many things from our discussions. I would also like to thank all of the other laboratory members.

Moreover, I would like to thank the members of KQCC at Keio University. At KQCC, I experienced a variety of exciting things regarding new technologies used in quantum computers. I want to thank Kohei Itoh and Naoki Yamamoto, who welcomed me warmly. During my research at KQCC, I learned many new things. Yutaka Shikano discussed cryptography against quantum computers, and I obtained many new perspectives from such discussions. Rodney Van Meter and Takahiko Satoh also introduced me to many good papers and problems in quantum computing. Moreover, Tomoki Tanaka, Shumpei Uno, and Yuki Tanaka discussed research and job hunting. Their comments give me a

clear image of research conducted in a company.

I am also grateful to the members of Mitsubishi Electric, where I joined as an intern. During my 1-month internship, I learned many things about research in general and my research in the real world. I obtained a clear direction on how I can progress. I would also like to thank Daisuke Suzuki for our discussions, which made me consider my research's theoretical and practical meanings. I would also like to thank the members of Mitsubishi Electric, Mitsuru Matsui, Katsuyuki Takashima, and Shoei Nashimoto. Owing to their courtesy, I was able to enjoy my internship.

Some results in the thesis are supported by Research Fellowships of Japan Society for the Promotion of Science for Young Scientists. Finally, I am deeply grateful to my family for supporting me in all stages of my life.

Bibliography

[1] Aaronson, S. and Chen, L.: Complexity-Theoretic Foundations of Quantum Supremacy Experiments. CCC 2017. pp. 1–67. ACM (2017).
[2] Albrecht, M.R., Cid, C., Faugère, J.-C., Fitzpatrick, R., and Perret, L.: On the Complexity of the BKW Algorithm on LWE. Designs, Codes, and Cryptography. **74**(2), 325–354 (2015).
[3] Albrecht, M.R., Deo, A., and Paterson, K.G.: Cold Boot Attacks on Ring and Module LWE Keys under the NTT. IACR Transactions on Cryptographic Hardware and Embedded Systems. **2018**(3), 173–213 (2018).
[4] Albrecht, M.R., Fitzpatrick, R., and Göpfert, F.: On the Efficacy of Solving LWE by Reduction to Unique-SVP. ICISC 2013. LNCS, vol. 8565, pp. 293–310. Springer (2014).
[5] Amiet, D., Curiger, A., Leuenberger, L., and Zbinden, P.: Defeating NewHope with a Single Trace. PQCrypto 2020. LNCS, vol. 12100, pp. 189–205. Springer (2020).
[6] Arute, F. et al.: Quantum Supremacy Using a Programmable Superconducting Processor. Nature. **574**(7779), 505–510 (2019).
[7] Aysu, A., Tobah, Y., Tiwari, M., Gerstlauer, A., and Orshansky, M.: Horizontal Side-Channel Vulnerabilities of Post-Quantum Key Exchange Protocols. HOST 2018. pp. 81–88. IEEE (2018).
[8] Beauregard, S.: Circuit for Shor's Algorithm Using 2n+3 Qubits. Quantum Information & Computation. **3**(2), 175–185 (2003).
[9] Beckman, D., Chari, A.N., Devabhaktuni, S., and Preskill, J.: Efficient Networks for Quantum Factoring. Physical Review A. **54**(2), 1034 (1996).
[10] Bernstein, D.J., Breitner, J., Genkin, D., Groot Bruinderink, L., Heninger, N., Lange, T., Van Vredendaal, C., and Yarom, Y.: Sliding Right into Disaster: Left-to-Right Sliding Windows Leak. CHES 2017. LNCS, vol. 10529, pp. 555–576. Springer (2017).
[11] Blum, A., Kalai, A., and Wasserman, H.: Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model. Journal of the ACM. **50**(4), 506–519 (2003).
[12] Brakerski, Z., Gentry, C., and Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. ITCS 2012. pp. 309–325. ACM (2012).
[13] Breitner, J.: More on Sliding Right. IACR eprint: 2018/1163 (2018).
[14] Breitner, J. and Skorski, M.: Analytic Formulas for Renyi Entropy of Hidden Markov Models. eprint arXiv: 1709.09699 (2017).
[15] Cabrera Aldaya, A., Pereida García, C., Alvarez Tapia, L.M., and Brumley, B.B.: Cache-Timing Attacks on RSA Key Generation. IACR Transactions on Cryptographic Hardware and Embedded Systems. **2019**(4), 213–242 (2019).
[16] Chari, S., Rao, J.R., and Rohatgi, P.: Template Attacks. CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer (2003).
[17] Chen, Y. and Nguyen, P.Q.: BKZ 2.0: Better Lattice Security Estimates. ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer (2011).
[18] Choi, B.-S. and Van Meter, R.: A θ $(\sqrt{n})$-Depth Quantum Adder on the 2D NTC Quantum Computer Architecture. ACM Journal on Emerging Technologies in Computing Systems. **8**(3), 1–22 (2012).
[19] Córcoles, A.D., Kandala, A, Javadi-Abhari, A., McClure, D.T., Cross, A.W., Temme,

K., Nation, P.D., Steffen, M., and Gambetta, J.M.: Challenges and Opportunities of Near-Term Quantum Computing Systems. Proceedings of the IEEE. **108**(8), 1338–1352 (2019).
[20] Cross, A.W., Bishop, L.S., Sheldon, S., Nation, P.D., and Gambetta, J.M.: Validating Quantum Computers Using Randomized Model Circuits. Physical Review A. **100**(3), 032328 (2019).
[21] Cuccaro, S.A., Draper, T.G., Kutin, S.A., and Moulton, D.P.: A New Quantum Ripple-Carry Addition Circuit. eprint arXiv: quant-ph/0410184 (2004).
[22] Davies, J.T., Rickerd, C.J., Grimes, M.A., and Güney, D.Ö.: An n-bit General Implementation of Shor's Quantum Factoring Algorithm. Quantum Information & Computation. **16**(7–8), 700–718 (2016).
[23] Devitt, S.J., Munro, W.J., and Nemoto, K.: Quantum Error Correction for Beginners. Reports on Progress in Physics. **76**(7), 076001 (2013).
[24] Diffie, W. and Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory. **22**(6), 644–654 (1976).
[25] Draper, T.G.: Addition on a Quantum Computer. eprint arXiv: quant-ph/0008033 (2000).
[26] Draper, T.G., Kutin, S.A., Rains, E.M., and Svore, K.M.: A Logarithmic-Depth Quantum Carry-Lookahead Adder. Quantum Information & Computation. **6**(4), 351–369 (2006).
[27] Ekerå, M. and Håstad, J.: Quantum Algorithms for Computing Short Discrete Logarithms and Factoring RSA Integers. PQCrypto 2017. LNCS, vol. 10346, pp. 347–363. Springer (2017).
[28] Fowler, A.G., Devitt, S.J., and Hollenberg, L.C.L.: Implementation of Shor's Algorithm on a Linear Nearest Neighbor Qubit Array. Quantum Information & Computation. **4**(4), 237–251 (2004).
[29] Fowler, A.G., Stephens, A.M., and Groszkowski, P.: High-Threshold Universal Quantum Computation on the Surface Code. Physical Review A. **80**(5), 052312 (2009).
[30] Fujisaki, E. and Okamoto, T.: Secure Integration of Asymmetric and Symmetric Encryption Schemes. CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer (1999).
[31] Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E.: Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation. CHES 2015. LNCS, vol. 9293, pp. 207–228. Springer (2015).
[32] Genkin, D., Shamir, A., and Tromer, E.: RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. CRYPTO 2014. LNCS, vol. 8616, pp. 444–461. Springer (2014).
[33] Gidney, C.: Halving the Cost of Quantum Addition. Quantum. **2**, 74 (2018).
[34] Gidney, C.: Windowed Quantum Arithmetic. eprint arXiv: 1905.07682 (2019).
[35] Gidney, C. and Ekerå, M.: How to Factor 2048 Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits. eprint arXiv: 1905.09749 (2019).
[36] Gidney, C. and Fowler, A.G.: Efficient Magic State Factories with a Catalyzed $|CCZ\rangle$ to $2\,|T\rangle$ Transformation. Quantum. **3**, 135 (2019).
[37] Gidney, C. and Fowler, A.G.: Flexible Layout of Surface Code Computations Using AutoCCZ States. eprint arXiv:1905.08916 (2019).
[38] Göttert, N., Feller, T., Schneider, M., Buchmann, J., and Huss, S.: On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes. CHES 2012. LNCS, vol. 7428, pp. 512–529. Springer (2012).
[39] Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. STOC 1996. pp. 212–219. ACM (1996).
[40] Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., and Felten, E.W.: Lest We Remember: Cold-Boot

Attacks on Encryption Keys. Communications of the ACM. **52**(5), 91–98 (2009).
[41] Häner, T., Jaques, S., Naehrig, M., Roetteler, M., and Soeken, M.: Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. PQCrypto 2020. LNCS, vol. 12100, pp. 425–444. Springer (2020).
[42] Henecka, W., May, A., and Meurer, A.: Correcting Errors in RSA Private Keys. CRYPTO 2010. LNCS, vol. 6223, pp. 351–369. Springer (2010).
[43] Heninger, N. and Shacham, H.: Reconstructing RSA Private Keys from Random Key Bits. CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer (2009).
[44] Hhan, M., Xagawa, K., and Yamakawa, T.: Quantum Random Oracle Model with Auxiliary Input. ASIACRYPT 2019. LNCS, vol. 11921, pp. 584–614. Springer (2019).
[45] Hirata, Y., Nakanishi, M., Yamashita, S., and Nakashima, Y.: An Efficient Conversion of Quantum Circuits to a Linear Nearest Neighbor Architecture. Quantum Information & Computation. **11**(1), 142–166 (2011).
[46] Hoeffding, W.: Probability Inequalities for Sums of Bounded Random Variables. Journal of the American Statistical Association. **58**(301), 13–30 (1963).
[47] Homma, N., Miyamoto, A., Aoki, T., Satoh, A., and Samir, A.: Comparative Power Analysis of Modular Exponentiation Algorithms. IEEE Transactions on Computers. **59**(6), 795–807 (2009).
[48] IBM: IBM Quantum Experience. `https://quantum-computing.ibm.com` (2020).
[49] IBM: IBM's Roadmap for Scaling Quantum Technology. `https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/` (2020).
[50] İnci, M.S., Gulmezoglu, B., Irazoqui, G., Eisenbarth, T., and Suner, B.: Cache Attacks Enable Bulk Key Recovery on the Cloud. CHES 2016. LNCS, vol. 9813, pp. 368–388. Springer (2016).
[51] Isailovic, N., Whitney, M., Patel, Y., and Kubiatowicz, J.: Running a Quantum Circuit at the Speed of Data. ACM SIGARCH Computer Architecture News. **36**(3), 177–188 (2008).
[52] Jones, N.C., Van Meter, R., Fowler, A.G., McMahon, P.L., Kim, J. Ladd, T.D., and Yamamoto, Y.: Layered Architecture for Quantum Computing. Physical Review X. **2**(3), 031007 (2012).
[53] Jurcevic, P. et al.: Demonstration of Quantum Volume 64 on a Superconducting Quantum Computing System. eprint arXiv: 2008.08571 (2020).
[54] Kirchner, P. and Fouque, P.-A.: An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices. CRYPTO 2015. LNCS, vol. 9215, pp. 43–62. Springer (2015)
[55] Koblitz, N.: Elliptic Curve Cryptosystems. Mathematics of Computation. **48**(177), 203–209 (1987).
[56] Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer (1996).
[57] Kocher, P., Jaffe, J., and Jun, B.: Differential Power Analysis. CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer (1999).
[58] Kunihiro, N. and Honda, J.: RSA Meets DPA: Recovering RSA Secret Keys from Noisy Analog Data. CHES 2014. LNCS, vol. 8731, pp. 261–278. Springer (2014).
[59] Kunihiro, N., Shinohara, N., and Izu, T.: Recovering RSA Secret Keys from Noisy Key Bits with Erasures and Errors. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences. **E97-A**(6), 1273–1284 (2014).
[60] Kunihiro, N. and Takahashi, Y.: Improved Key Recovery Algorithms from Noisy RSA Secret Keys with Analog Noise. CT-RSA 2017. LNCS, vol. 10159, pp. 328–343. Springer (2017).
[61] Lenstra, A.K., Lenstra, H.W., and, Lovász, L.: Factoring Polynomials with Rational

Coefficients. Mathematische Annalen. **261**(4), 515–534 (1982).
[62] Lidar, D.A. and Brun, T.A.: Quantum Error Correction. Cambridge University Press (2013).
[63] Longa, P. and Naehrig, M.: Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. CANS 2016. LNCS, vol. 10052, pp. 124–139. Springer (2016).
[64] Lu, X., Liu, Y., Zhang, Z., Jia, D., Xue, H., He, J., Li, B., and Wang, K.: LAC: Practical Ring-LWE Based Public-Key Encryption with Byte-Level Modulus. IACR eprint: 2018/1009. Submission to [80] (2018).
[65] Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehle, D., and Bai, S.: CRYSTALS-DILITHIUM. Submission to [80]. `https://pq-crystals.org/` (2017).
[66] Lyubashevsky, V., Peikert, C., and Regev, O.: On Ideal Lattices and Learning with Errors over Rings. EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer (2010).
[67] Lyubashevsky, V., Peikert, C., and Regev, O.: On Ideal Lattices and Learning with Errors over Rings. Journal of the ACM. **60**(6), 1–35 (2013).
[68] Markov, I.L. and Saeedi, M.: Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation. Quantum Information & Computation. **12**(5–6), 361–394 (2012).
[69] Martín-López, E., Laing, A., Lawson, T., Alvarez, R., Zhou, X.-Q., and O'Brien, J.L.: Experimental Realization of Shor's Quantum Factoring Algorithm Using Qubit Recycling. Nature Photonics. **6**(11), 773-776 (2012).
[70] Maslov, D.: Advantages of Using Relative-Phase Toffoli Gates with an Application to Multiple Control Toffoli Optimization. Physical Review A. **93**(2), 022311 (2016).
[71] Maslov, D., Dueck, G.W., Miller, D.M., and Negrevergne, C.: Quantum Circuit Simplification and Level Compaction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. **27**(3), 436–444 (2008).
[72] Menezes, A.J., Van Oorschot, P.C., and Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996).
[73] Messerges, T.S., Dabbish, E.A., and Sloan, R.H.: Power Analysis Attacks of Modular Exponentiation in Smartcards. CHES 1999. LNCS, vol. 1717, pp. 144–157. Springer (1999).
[74] Miller, V.S.: Use of Elliptic Curves in Cryptography. CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer (1986).
[75] Mogensen, T.Æ.: Reversible In-Place Carry-Lookahead Addition with Few Ancillae. RC 2019. LNCS, vol. 11497, pp. 224–237. Springer (2019).
[76] Moriarty, K., Kaliski, B., Jonsson, J., and Rusch, A.: PKCS #1: RSA Cryptography Specifications Version 2.2. `https://tools.ietf.org/html/rfc8017` (2016).
[77] Murali, P., Debroy, D.M., Brown, K.R., and Martonosi, M.: Architecting Noisy Intermediate-Scale Trapped Ion Quantum Computers. eprint arXiv: 2004.04706 (2020).
[78] National Institute of Standards and Technology: Announcing the ADVANCED ENCRYPTION STANDARD (AES). `https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf` (2001).
[79] National Institute of Standards and Technology: Digital Signature Standard (DSS). `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf` (2013).
[80] National Institute of Standards and Technology: Post-Quantum Cryptography. `https://csrc.nist.gov/projects/post-quantum-cryptography` (2017).
[81] Nielsen, M.A. and Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press (2000).
[82] Oonishi, K., Huang, X., and Kunihiro, N.: Improved CRT-RSA Secret Key Recovery Method from Sliding Window Leakage. ICISC 2019. LNCS, vol. 11975, pp. 278–296.

Springer (2020).
[83] Oonishi, K. and Kunihiro, N.: Recovering CRT-RSA Secret Keys Using the Information of the Operations Based on Sliding Window Method. SCIS 2018. (2018).
[84] Oonishi, K. and Kunihiro, N.: Attacking Noisy Secret CRT-RSA Exponents in Binary Method. ICISC 2018. LNCS, vol. 11396, pp. 37–54. Springer (2019).
[85] Oonishi, K. and Kunihiro, N.: Recovering CRT-RSA Secret Keys from Noisy Square-and-Multiply Sequences in the Sliding Window Method. ACISP 2020. LNCS, vol. 12248, pp. 642–652. Springer (2020).
[86] Oonishi, K., Tanaka, T., Uno, S., Satoh, T., Van Meter, R., and Kunihiro, N.: Efficient Construction of a Control Modular Adder on a Carry-Lookahead Adder Using Relative-Phase Toffoli Gates. eprint arXiv: 2010.00255 (2020).
[87] Paterson, K.G., Polychroniadou, A., and Sibborn, D.L.: A Coding-Theoretic Approach to Recovering Noisy RSA Keys. ASIACRYPT 2012. LNCS, vol. 7658, pp. 386–403. Springer (2012).
[88] Pavlidis, A. and Gizopoulos, D.: Fast Quantum Modular Exponentiation Architecture for Shor's Factoring Algorithm. Quantum Information & Computation. **14**(7&8), 649–682 (2014).
[89] Percival, C.: Cache Missing for Fun and Profit. `http://www.daemonology.net/papers/htt.pdf` (2005)
[90] Pessl, P. and Primas, R.: More Practical Single-Trace Attacks on the Number Theoretic Transform. LATINCRYPT 2019. LNCS, vol. 11774, pp. 130–149. Springer (2019).
[91] Pino., J.M., Dreiling, J.M., Figgatt, C., Gaebler, J.P., Moses, S.A., Allman, M.S., Baldwin, C.H., Foss-Feig, M., Hayes, D., Mayer, K., Ryan-Anderson, C., and Neyenhuis, B.: Demonstration of the QCCD Trapped-Ion Quantum Computer Architecture. eprint arXiv: 2003.01293 (2020).
[92] Pöppelmann, T., Alkim, E., Avanzi, R., Bos, J., Ducas, L., De La Piedra, A., Schwabe, P., Stebila, D., Albrecht, M.R., Orsini, E., Osheter, V., Paterson, K.G., Peer, G., and Smart, N.P.: NewHope. Submission to [80]. `http://newhopecrypto.org/` (2017).
[93] Preskill, J.: Fault-Tolerant Quantum Computation. Introduction to Quantum Computation and Information. pp. 213–269. World Scientific Publishing (1998).
[94] Preskill, J.: Quantum Computing in the NISQ Era and Beyond. Quantum. **2**, 79 (2018).
[95] Primas, R., Pessl, P., and Mangard, S.: Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer (2017).
[96] Ravi, P., Bhasin, S., Sinha Roy, S., and Chattopadhyay, A.: Drop by Drop You Break the Rock - Exploiting Generic Vulnerabilities in Lattice-based PKE/KEMs Using EM-based Physical Attacks. IACR eprint: 2020/549 (2020).
[97] Ravi, P., Sinha Roy, S., Chattopadhyay, A., and Bhasin, S.: Generic Side-Channel Attacks on CCA-secure Lattice-Based PKE and KEMs. IACR Transactions on Cryptographic Hardware and Embedded Systems. **2020**(3), 307–335 (2020).
[98] Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. Journal of the ACM. **56**(6), 1–40 (2009).
[99] Rivest, R.L., Shamir, A., and Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM. **21**(2), 120–126 (1978).
[100] Roetteler, M., Naehrig, M., Svore, K.M., and Lauter, K.: Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms. ASIACRYPT 2017. LNCS, vol. 10625, pp. 241–270. Springer (2017).
[101] Satoh, T., Ohkura, Y., and Van Meter, R.: Subdivided Phase Oracle for NISQ Search Algorithms. eprint arXiv: 2001.06575 (2020).

[102] Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., and Stehle, D.: CRYSTALS-KYBER. Submission to [80]. `https://pq-crystals.org/` (2017).
[103] Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Review. **41**(2), 303–332 (1999).
[104] Smith, R.S., Curtis, M.J., and Zeng, W.J.: A practical Quantum Instruction Set Architecture. eprint arXiv: 1608.03355 (2016).
[105] Smith, W.L.: Renewal Theory and Its Ramifications. Journal of the Royal Statistical Society. Series B (Methodological). **20**(2), 243–302 (1958).
[106] Steane, A.M.: Space, Time, Parallelism and Noise Requirements for Reliable Quantum Computing. Fortschritte der Physik: Progress of Physics. **46**(4–5), 443–457 (1998).
[107] Steane, A.M.: Overhead and Noise Threshold of Fault-Tolerant Quantum Error Correction. Physical Review A. **68**(4), 042322 (2003).
[108] Takahashi, Y., Tani, S., and Kunihiro, N.: Quantum Addition Circuits and Unbounded Fan-Out. Quantum Information & Computation. **10**(9), 872–890 (2010).
[109] Terhal, B.M.: Quantum Error Correction for Quantum Memories. Reviews of Modern Physics. **87**(2), 307 (2015).
[110] Thapliyal, H., Jayashree, H.V., Nagamani, A.N., and Arabnia, H.R.: Progress in Reversible Processor Design: A Novel Methodology for Reversible Carry Look-Ahead Adder. Transactions on Computational Science XVII. LNCS, vol. 7420, pp. 73–97. Springer (2013).
[111] Thapliyal, H., Muñoz-Coreas, E., and Khalus, V.: T-count and Qubit Optimized Quantum Circuit Designs of Carry Lookahead Adder. eprint arXiv: 2004.01826 (2020).
[112] Tu Darmstadt: Learning With Errors Challenge. `https://www.latticechallenge.org/lwe_challenge/challenge.php` (2015).
[113] Van Meter, R. and Itoh, K.M.: Fast Quantum Modular Exponentiation. Physical Review A. **71**(5), 052320 (2005).
[114] Van Meter, R., Ladd, T.D., Fowler, A.G., and Yamamoto, Y.: Distributed Quantum Computation Architecture Using Semiconductor Nanophotonics. International Journal of Quantum Information. **8**(01n02), 295–323 (2010).
[115] Van Vredendaal, C.: Exploiting Mathematical Structures in Cryptography. Eindhoven: Technische Universiteit Eindhoven (2018).
[116] Vedral, V., Barenco, A., and Ekert, A.: Quantum Networks for Elementary Arithmetic Operations. Physical Review A. **54**(1), 147 (1996).
[117] Walter, C.D.: Sliding Windows Succumbs to Big Mac Attack. CHES 2001. LNCS, vol. 2162, pp. 286–299. Springer (2001).
[118] Xu, Z., Pemberton, O., Sinha Roy, S., and Oswald D.: Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems with Chosen Ciphertexts: The Case Study of Kyber. IACR eprint: 2020/912 (2020).
[119] Yarom, Y. and Falkner, K.: FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. USENIX 2014. pp. 719–732. USENIX (2014).
[120] Yarom, Y., Genkin, D., and Heninger, N.: CacheBleed: A Timing Attack on OpenSSL Constant Time RSA. CHES 2016. LNCS, vol. 9813, pp. 346–367. Springer (2016).
[121] Zalka, C.: Fast Versions of Shor's Quantum Factoring Algorithm. eprint arXiv: quant-ph/9806084 (1998).