



國立臺灣大學理學院數學研究所

碩士論文

Graduate Institute of Mathematics

College of Science

National Taiwan University

Master Thesis

Saber 的理論研究與推廣

Theoretical Survey of Saber and Its
Generalizations

陳和謙

Ho-Chien Chen

指導教授：陳君明 博士

Advisor: Jiun-Ming Chen, Ph.D.

中華民國 111 年 3 月

March 2022



摘要

Saber 是一個基於模上的 learning with rounding 難題假設的密碼系統，因此可以抵禦後量子電腦的攻擊。在這篇論文中，我們首先研究 Saber 的密文分布與解密失敗機率。藉由分析 Saber 中所使用的環的代數性質，我們可以證明 Saber 中的密文分布是均勻的，但是其所發布的解密失敗機率的估計使用了錯誤的假設。我們也會給出一個 Saber 解密失敗機率的上界。再來我們考慮 Saber 的一種名為「NTT-friendly Saber」的變體。藉由把原本 2 的冪次的模數改為一些特定的質數，我們可以讓 NTT-friendly Saber 的實作速度更快。我們接著討論 NTT-friendly Saber 的參數選擇、密文分布與解密失敗機率。最後，由於 NTT-friendly Saber 的設計理念和另一個也是大家所熟知的 Kyber 十分類似，我們會討論一些和 Kyber 的比較。

關鍵字：後量子密碼學、Saber、learning with rounding、快速數論變換 (NTT)

Abstract

Saber is a cryptosystem based on the hardness of the module-learning with rounding problem, hence has resistance against quantum computers. In this paper, we first examine the ciphertext distribution and error rate of Saber. By some algebraic analysis of the rings used in Saber, we will see that the ciphertext distribution is uniform in Saber, while the proposed error rate estimations contain a false assumption and thus lead to questionable results. We also give an upper bound of the error rates of Saber.

We then consider a variance of Saber called "NTT-friendly Saber." By changing the moduli from power-of-2 to some specific primes, this NTT-friendly Saber will have a faster implementation speed. We then discuss the parameters choosing, ciphertext distribution, and error rate. At last, since the design rationales are pretty similar to another known cryptosystem named Kyber, we will compare our NTT-friendly Saber and Kyber.

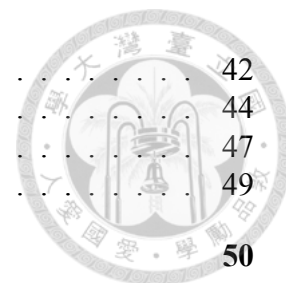
Keywords: post-quantum cryptography, Saber, learning with rounding, number theoretic transform (NTT)



Contents

1	Introduction	1
2	Preliminaries	3
2.1	Lattice	3
2.2	Rounding	4
2.3	Computational hardness assumptions	5
2.4	Number-theoretic transform	6
3	Saber	7
3.1	Introduction to Saber	7
3.2	Preliminaries and notations	8
3.3	Parameters and constants	9
3.4	Algorithm	10
3.4.1	Key generation	10
3.4.2	Encryption	10
3.4.3	Decryption	11
3.4.4	Parameter sets	11
4	Ciphertext Distribution and Error Rate of Saber	12
4.1	Algebraic structure of the rings used in Saber	12
4.2	Ciphertext distribution of Saber	15
4.3	Error rate of Saber	19
4.3.1	Some important lemmas	22
4.3.2	Flaw of the original estimation	24
4.3.3	A correct bound of the error rate of Saber	27
5	NTT-friendly Variance of Saber	37
5.1	Motivation	37
5.2	Number theoretic transform (NTT)	37
5.3	Notations and parameters	39
5.4	Algorithm	40
5.4.1	Key generation	40
5.4.2	Encryption	41
5.4.3	Decryption	41
5.4.4	Parameter sets	42

5.5	Ciphertext distribution	42
5.6	Error rate estimation	44
5.7	Comparison with the original Saber	47
5.8	Comparison with Kyber	49
6	Conclusion	50
A	Code to Estimate the Error Rates of Saber	52
B	Code to Estimate the Error Rates of NTT-Friendly Saber	56





Chapter 1

Introduction

In 1994, Peter Shor published the well-known Shor's algorithm, a polynomial-time quantum algorithm to factor integers. In the following decades, Shor's algorithm has been generalized to solve various discrete logarithm problems in polynomial time. On the other hand, quantum computers with more and more qubits were successfully built. As a result, it is likely that present cryptographic primitives, especially RSA and ECDSA, will be broken shortly. Therefore, we urgently need post-quantum cryptography (PQC), which is secure against even quantum computers.

In 2016, the National Institute of Standards and Technology (NIST) called for proposals to standardize quantum-secure cryptographic primitives. 59 encryption/KEM schemes were submitted by the initial submission deadline at the end of 2017. Saber was one of them and turned out to be one of the round-3 finalist candidates as announced on July 22, 2020.

In this paper, we introduce Saber, then survey some fundamental properties of Saber plus the background theories, including the distribution of ciphertext and estimation of error rate. Next, we discuss the different moduli-choosing strategies of Saber. While the moduli in the

original Saber are all power-of-2, we attempt to use NTT-friendly moduli and thoroughly examine the impacts on various aspects. Since this change will make our cryptosystem alike with another well-known lattice-based cryptosystem named Kyber, we also present a comparison with the Kyber.

This paper is organized as follows: in Chapter 2 we present some preliminaries; we introduce Saber, including the algorithm and the parameter sets in Chapter 3. We develop some theorems to illustrate the algebraic structure of the rings used in Saber in Section 4.1. We then use the results to examine the ciphertext distribution in Section 4.2 and error rates of Saber in Section 4.3. At last, we present the algorithm "NTT-friendly Saber" in Chapter 5. Primarily, we give the algorithm in Section 5.4, discuss the ciphertext distribution in Section 5.5 and estimate the error rate in Section 5.6. We further show a comparison between the NTT-friendly Saber and the Kyber in Section 5.7.



Chapter 2

Preliminaries

2.1 Lattice

Let $S = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\} \subseteq \mathbb{R}^n$ be a linearly independent set. Then $V = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_n\mathbf{v}_n \mid a_i \in \mathbb{Z}\}$ is called the **lattice** generated by S . If lattice V may be generated by a set S then S is called a **basis** of V , and $|S|$ is the **dimension** of V .

Let V be an additive subgroup of \mathbb{R}^n , then V is said to be **discrete** if there exists $\varepsilon > 0$ such that for any $v, w \in V$ where $v \neq w$, we have $\|v - w\| > \varepsilon$. Another equivalent definition is that a lattice is a discrete additive subgroup of \mathbb{R}^n .

2.2 Rounding



Let $q > p$ be two integers. Then we may define a rounding φ from \mathbb{Z}_q to \mathbb{Z}_p as follows:

$$\varphi : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$$

$$\bar{x} \mapsto \left\lfloor \frac{px}{q} + r \right\rfloor$$

Here r is a fixed real number and $\lfloor \cdot \rfloor$ is the floor function. We note that for any element in \mathbb{Z}_q we may find an integer x such that \bar{x} is equal to this element in \mathbb{Z}_q , and then we plug x in the right hand side to get the result of rounding. Furthermore, since when x differs by a multiple of q the value of $\frac{px}{q} + r$ differs by a multiple of p , we may see that φ is well-defined.

When p divides q , the roundings φ may be reduced to the following form:

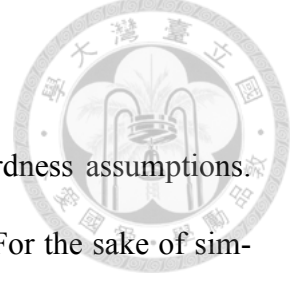
$$\varphi : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$$

$$\bar{x} \mapsto \left\lfloor \frac{px+k}{q} \right\rfloor$$

Here k is a fixed integer.

For normal roundings, $r = \frac{1}{2}$ or $k = \frac{q}{2}$. These constants are set so that if we try to recover the value of x by computing $\frac{q}{p}x$ we get the minimal error on average. For different r or k we say that this rounding is tweaked.

2.3 Computational hardness assumptions



In this section, we will introduce some well-known computational hardness assumptions. These assumptions are the cornerstones of lattice-based cryptography. For the sake of simplicity, we only present the form put into practice most often.

Shortest vector problem (SVP). Let $L \subseteq \mathbb{R}^n$ be a lattice. Find $\mathbf{v} \in L - \{0\}$ such that $\|\mathbf{v}\|$, the Euclidean norm of \mathbf{v} , is minimal.

Closest vector problem (CVP). Let $L \subseteq \mathbb{R}^n$ be a lattice and $\mathbf{v} \in \mathbb{R}^n$. Find $\mathbf{w} \in L$ such that $\|\mathbf{w} - \mathbf{v}\|$ is minimal.

Learning with errors problem (LWE problem). Let \mathbb{Z}_q denote the ring of integers modulo q . Fix $\mathbf{s} \in \mathbb{Z}_q^n$. Let $\mathbf{a} \in \mathbb{Z}_q^n$ be a uniform random variable over \mathbb{Z}_q^n and $e \in \mathbb{Z}_q$ be another random variable being independent with \mathbf{a} and associated with a certain distribution χ . Define $b = \langle \mathbf{a}, \mathbf{s} \rangle + e$. Given some samples of (\mathbf{a}, b) , find \mathbf{s} .

Ring learning with errors problem (RLWE problem). Let $f(x) \in \mathbb{Z}_q[x]$ be a polynomial and define $R_q = \mathbb{Z}_q[x]/f(x)$. Fix $s \in R_q$. Let $a \in R_q$ be a uniform random variable over R_q and $e \in R_q$ be another random variable being independent with a and associated with a certain distribution χ . Define $b = a \cdot s + e$. Given some samples of (a, b) , find s .

Module learning with errors problem (MLWE problem). Let $f(x) \in \mathbb{Z}_q[x]$ be a polynomial and define $R_q = \mathbb{Z}_q[x]/f(x)$. Fix $\mathbf{s} \in R_q^n$. Let $\mathbf{a} \in R_q^n$ be a uniform random variable over R_q^n and $e \in R_q$ be another random variable being independent with \mathbf{a} and associated with a certain distribution χ . Define $b = \langle \mathbf{a}, \mathbf{s} \rangle + e$. Given some samples of (\mathbf{a}, b) , find \mathbf{s} .

Learning with rounding problem (LWR problem). Let $q > p$. Define the rounding function $\varphi : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$ such that $\varphi(\bar{x}) = \left\lfloor \frac{p}{q}x \right\rfloor \pmod{p}$, where $\lfloor x \rfloor = \lfloor x + 0.5 \rfloor$. Fix $\mathbf{s} \in \mathbb{Z}_q^n$. Let $\mathbf{a} \in \mathbb{Z}_q^n$ be a uniform random variable over \mathbb{Z}_q^n and define $b = \varphi(\langle \mathbf{a}, \mathbf{s} \rangle)$. Given some samples of (\mathbf{a}, b) , find \mathbf{s} .

Ring learning with rounding (RLWR) and module learning with rounding (MLWR) problems may be defined similarly.

2.4 Number-theoretic transform

Number-theoretic transform (NTT) is an algorithm that helps us compute polynomial multiplication faster. To be more precisely, suppose p is a prime such that $p \equiv 1 \pmod{2n}$, where $n = 2^k$ is a power of 2. For $a, b \in \mathbb{Z}_p[x]/(x^n + 1)$, computing $a \cdot b$ by NTT costs only $O(n \log n)$ operations, given that one addition or multiplication in \mathbb{Z}_p costs 1 operation.

Sometimes the condition $p \equiv 1 \pmod{2n}$ greatly restricts the choose of the prime p . By using incomplete NTT, the condition may be loosened to $p \equiv 1 \pmod{n/2}$ while the computation time almost keeps the same.

Though it seems that NTT can only be used under restricted circumstances, it turns out that by some easy tricks, NTT may also accelerate multiplications in $\mathbb{Z}[x]$, hence $\mathbb{Z}_p[x]/(x^n + 1)$ for general p, n . We show more details in Section 5.2.



Chapter 3

Saber

In this chapter, all notations and contents follow the round 3 submission of Saber in the NIST post-quantum cryptography competition [2].

3.1 Introduction to Saber

Saber is a lattice-based cryptosystem that relies on the hardness of the MLWE problem. There are two parts of Saber, called Saber-PKE and Saber-KEM. Though both of them may let both sides achieve a consensus, Saber-KEM furthermore has merit that no one can easily manipulate the consensus string, thus being safer for many applications.

Saber-KEM is established by applying the Fujisaki-Okamoto transform on Saber-PKE. We will omit the KEM part since we are only concerned with the public-key encryption (PKE) part in this paper.

3.2 Preliminaries and notations



Structures and operators.

$\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denotes the ring of integers modulo q .

$R_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 256$ are fixed for all variances of Saber.

Let R be an arbitrary ring. $R^{m \times n}$ denotes an $m \times n$ matrix with all its coefficients in R .

$>>$ and $<<$ are bitwise shift operators. To be more precisely, $a << n = 2^n \cdot a$, and $a >>$

$n = \lfloor \frac{a}{2^n} \rfloor$. If $\frac{q}{p} = 2^n$, define the map φ as follows:

$$\varphi: \mathbb{Z}_q \rightarrow \mathbb{Z}_p$$

$$\bar{x} \mapsto \left\lfloor \frac{x}{2^n} \right\rfloor$$

Then we may check that this φ is well-defined. Thus for $k \in \mathbb{Z}_q$ we may define $k >> n \in \mathbb{Z}_p$.

Similarly, for $k \in \mathbb{Z}_p$ we may define $k << n \in \mathbb{Z}_q$.

For the operators $<< n$, $>> n$ and $(\text{mod } p)$, if they are applied on (matrices of) R_q , then they are applied coefficient-wisely.

Conventions.

Elements in R , R_q , \mathbb{Z} , or \mathbb{Z}_q are represented by regular font letters. Vectors with components in R , R_q , \mathbb{Z} , or \mathbb{Z}_q are represented by bold lower-case letters represent. Matrices with components in R , R_q , \mathbb{Z} , or \mathbb{Z}_q are represented by bold upper-case letters represent. If $\mathbf{v} \in R_q^{l \times 1}$ is a vector, then it is a column vector and v_i denotes the i -th entry of \mathbf{v} .

Probability distributions and sampling.

Let S be a finite set. Then $\mathcal{U}(S)$ denotes a uniform distribution over S . $x \leftarrow \mathcal{U}(S)$ means that x is a uniformly chosen element of S . On the other hand, $x = \mathcal{U}(S)$ represents that x is a random variable such that $P(x = s) = 1/|S|$ for all $s \in S$. Let χ be a probability distribution over S , then $x \leftarrow \chi$ means that x is sampled according to χ . Furthermore, if χ is defined over \mathbb{Z}_q , then $\mathbf{X} \leftarrow \chi(R_q^{a \times b})$ means that each coefficient of each entry of \mathbf{X} are sampled according to χ .

β_μ is the centered binomial distribution with range in $[-\mu/2, \mu/2]$. To be more precisely, if

$$x = \beta_\mu \text{ then } P(x = k) = \frac{1}{2^\mu} \binom{\mu}{\frac{\mu}{2} - k} \text{ for } |k| \leq \frac{\mu}{2}.$$

3.3 Parameters and constants

$n = 256$ is fixed throughout this chapter. Hence $R_q = \mathbb{Z}_q[x]/(x^n + 1) = \mathbb{Z}_q[x]/(x^{256} + 1)$. l is the rank of the matrix used in Saber. Larger l results in higher security, but lower correctness.

q, p, T are the moduli used in Saber and they are all power-of-2 satisfying $q > p > T$. This guarantees that $p|q$, thus an element in \mathbb{Z}_q may be regarded as in \mathbb{Z}_p by simply taking modulo p . ϵ_q is defined to be $\log_2 q$. These notations are defined such that $((x + q/(2p)) \gg (\epsilon_q - \epsilon_p))$ is a normal rounding from \mathbb{Z}_q to \mathbb{Z}_p .

μ determines the range of the centered binomial distribution, which is usually small. $\text{gen} :$

$\{0, 1\}^{256} \rightarrow R_q^{l \times l}$ is a function to generate a matrix by a seed.

$h_1, h_2 \in R_q$ and $\mathbf{h} \in R_q^{l \times 1}$ are defined as constants in Saber. The coefficients of h_1 are all

defined to be $2^{\epsilon_q - \epsilon_p - 1}$ and the coefficients of h_2 are all defined to be $2^{\epsilon_p - 2} - 2^{\epsilon_p - \epsilon_T - 1} + 2^{\epsilon_q - \epsilon_p - 1}$. All the entries of \mathbf{h} are h_1 . The values of h_1, h_2 define some tweaked roundings due to the need of a security proof as explained in [1]. As we will see later, the error rate will be as if the roundings are all normal roundings.

3.4 Algorithm

In this section, we describe the algorithm of the Saber public-key encryption algorithm. By convention, we assume that Alice generates private and public keys, and Bob sends the message to Alice.

3.4.1 Key generation

- (1) Alice picks $seed_{\mathbf{A}} \leftarrow \{0, 1\}^{256}$ and generates $\mathbf{A} = \text{gen}(seed_{\mathbf{A}}) \in R_q^{l \times l}$.
- (2) Alice picks secret key $\mathbf{s} \leftarrow \beta_{\mu}(R_q^{l \times 1})$.
- (3) Compute $\mathbf{b} = (\mathbf{A}^T \mathbf{s} + \mathbf{h}) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times 1}$.
- (4) Alice publishes public key $(seed_{\mathbf{A}}, \mathbf{b})$.

3.4.2 Encryption

- (1) Bob encodes his 256-bit message as $m \in R_2$.
- (2) Bob generates $\mathbf{A} = \text{gen}(seed_{\mathbf{A}}) \in R_q^{l \times l}$ and picks $\mathbf{s}' \leftarrow \beta_{\mu}(R_q^{l \times 1})$.
- (3) Compute $\mathbf{b}' = (\mathbf{A} \mathbf{s}' + \mathbf{h}) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times 1}$ and $v' = \mathbf{b}^T \mathbf{s}' \in R_p$.

(4) Bob computes ciphertext $c_m = (v' - 2^{\epsilon_p - 1}m + h_1) \gg (\epsilon_p - \epsilon_T) \in R_T$.

(5) Bob sends (c_m, \mathbf{b}') to Alice.



3.4.3 Decryption

(1) Compute $v = \mathbf{b}^T \mathbf{s} \in R_p$.

(2) Alice computes $m' = (v - 2^{\epsilon_p - \epsilon_T} c_m + h_2) \gg (\epsilon_p - 1) \in R_2$ as decrypted message.

3.4.4 Parameter sets

Three parameter sets are proposed to meet different security level requirements of NIST. In the following table, different values of the parameters q, p, T, μ, l and the sizes of the public key, secret key, and ciphertext (in bytes) are shown as follows.

Scheme	q	p	T	μ	l	pk(B)	sk(B)	ct(B)
LightSaber	2^{13}	2^{10}	2^6	10	2	672	832	736
Saber	2^{13}	2^{10}	2^4	8	3	992	1248	1088
FireSaber	2^{13}	2^{10}	2^3	6	4	1312	1664	1472



Chapter 4

Ciphertext Distribution and Error Rate of Saber

4.1 Algebraic structure of the rings used in Saber

In this section, we consider rings of the form $R = \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$. We know that the Saber uses the case $(n, k) = (13, 8)$ and $(n, k) = (10, 8)$. Consequently, the algebraic structure reflects on various aspects of the analysis of Saber

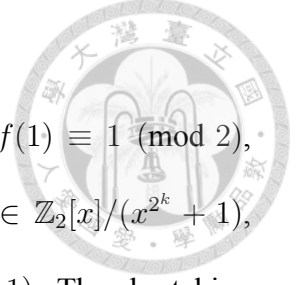
Theorem 4.1. Let $R = \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$, where $n, k \in \mathbb{N}$, and $f \in \mathbb{Z}[x]$. Then $\bar{f} \in R^\times$ if and only if $f(1) \equiv 1 \pmod{2}$.

Proof. \Rightarrow : Suppose $\bar{f} \cdot \bar{g} = 1$, then we have $f(x)g(x) = 1 + (x^{2^k} + 1)h_1(x) + 2^n h_2(x)$. Apply $x = 1$ and the result follows.

\Leftarrow : We first note that $|R| < \infty$, thus it suffices to prove that \bar{f} is not a zero divisor in R .

We prove by induction on n . When $n = 1$, $R = \mathbb{Z}_2[x]/(x^{2^k} + 1) = \mathbb{Z}_2[x]/((x + 1)^{2^k})$.

Suppose $\bar{f}\bar{g} = \bar{0} \in R$, then $\bar{f} \cdot \bar{g}$ is a multiple of $(x + 1)^{2^k}$ in $\mathbb{Z}_2[x]$. Furthermore $f(1) \equiv 1 \pmod{2}$ implies that $(x + 1)$ does not divide \bar{f} in $\mathbb{Z}_2[x]$, thus $(x + 1)^{2^k}$ divides \bar{g} in $\mathbb{Z}_2[x]$,



which implies that $\bar{g} = \bar{0} \in R$. Therefore, \bar{f} is not a unit.

Assume that the statement holds for $n - 1$. Assume that $f, g \in \mathbb{Z}[x]$, $f(1) \equiv 1 \pmod{2}$, and $\bar{f}\bar{g} = 0 \in \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$. Note that it also holds that $\bar{f}\bar{g} = 0 \in \mathbb{Z}_2[x]/(x^{2^k} + 1)$, similarly with the preceding paragraph we have $\bar{g} = 0 \in \mathbb{Z}_2[x]/(x^{2^k} + 1)$. Thus by taking the remainder divided by $x^{2^k} + 1$, we may assume that $\deg g(x) < 2^k$, and we find that the coefficients of $g(x)$ are all even. Hence we assume $g(x) = 2g'(x)$, and then we have $\bar{f}\bar{g}' = 0 \in \mathbb{Z}_{2^{n-1}}[x]/(x^{2^k} + 1)$. By induction hypothesis we have $\bar{g}' = 0 \in \mathbb{Z}_{2^{n-1}}[x]/(x^{2^k} + 1)$, which along with $\deg g' < 2^k$ implies that the coefficients of g' are all multiples of 2^{n-1} , thus the coefficients of $g(x) = 2g'(x)$ are all multiples of 2^n , which means that $\bar{g} = 0 \in \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$. Therefore \bar{f} is not a zero divisor in R . This completes the proof.

With the help of Theorem 4.1, we may further investigate the algebraic structure of $R = \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$. The results are presented in the next theorem:

Theorem 4.2. Let $R = \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$, where $n, k \in \mathbb{N}$, and $I = (x - 1)$ be an ideal of R .

Then:

- (i) $I = (x - 1, 2)$.
- (ii) $I^{2^k} = (2)$, thus I is nilpotent.
- (iii) $R^\times = R - I$.
- (iv) For all $0 \leq t < n \cdot 2^k$, we have $|I^t| = 2^{n \cdot 2^k - t}$ and $|I^t - I^{t+1}| = 2^{n \cdot 2^k - t - 1}$.
- (v) For any $r \in R$, we have $r = (x - 1)^t \cdot u$ for some non-negative t and $u \in R^\times$. Furthermore, if $r \neq 0$, then the value t is unique, and $(r) = I^t$.
- (vi) All ideals of R are of the form I^t for some t .



Proof. (i) It suffices to show that $2 \in I$, which is true since

$$2 = -(x-1)(x^{2^k-1} + \dots + x + 1).$$

(ii) Let $f(x) = (x-1)^{2^k} - (x^{2^k} + 1)$. We know that $I^{2^k} = (f(x))$.

\subseteq : Since the coefficients of $f(x)$ are all even.

\supseteq : If $n = 1$ then both sides are the zero ideal. Otherwise, Let $f(x) = 2g(x)$, then we have $g(1) = -1 \equiv 1 \pmod{2}$, thus g is a unit in R by Theorem 4.1. Hence $2 = f \cdot g^{-1} \in (f(x)) = I^{2^k}$. It follows that $I^{n \cdot 2^k} = (0)$, thus I is nilpotent.

(iii) By Theorem 4.1, $\bar{f} \in R^\times$ if and only if $f(1) \equiv 1 \pmod{2}$. On the other hand, by (i) we know that $\bar{f} \in I$ if and only if $f(1) \equiv 0 \pmod{2}$. The result follows.

(iv) For non-negative integer t , we define

$$\varphi: I^t \rightarrow I^{t+1}$$

$$r \mapsto (x-1)r$$

It's clear that φ is a surjective additive group homomorphism, therefore $I^t/\ker(\varphi) \cong I^{t+1}$, which implies that $|I^t|/|\ker(\varphi)| = |I^{t+1}|$.

Solve $(\sum_{i=0}^{2^k-1} a_i x^i)(x+1) = 0$ by directly comparing the coefficients, we may find that the candidates for $\ker(\varphi)$ are only 0 and $\kappa := \sum_{i=0}^{2^k-1} 2^{n-1} x^i$. Note that by (ii) we have

$I^{(n-1) \cdot 2^k} = (2^{n-1})$ and thus $\kappa = 2^{n-1} \cdot (x-1)^{2^k-1} \in I^{2^n-1}$, we may conclude that for $t < n \cdot 2^k$ we have $|\ker(\varphi)| = |\{0, \kappa\}| = 2$. Therefore, we have $|I^t| = 2|I^{t+1}|$ for all $t < n \cdot 2^k$. Along with the facts that $|I^0| = |R| = 2^{n \cdot 2^k}$ and $I^{t+1} \subseteq I^t$ for all t , the result follows.

(v) If $r = 0$ we may let $t = n \cdot 2^k$ and $u = 1$. Otherwise, we note that $R = I^0 \supseteq I^1 \supseteq I^2 \supseteq \dots \supseteq I^{n \cdot 2^k} = (0)$. Since $r \neq 0$, we may let t be the largest integer such that $r \in I^t$. Hence $r = (x-1)^t \cdot u$ for some $u \in R$, and since r is the largest we must have $u \in R - I = R^\times$.

It is clear that $I^t = ((x-1)^t) = (r)$. Since $r \neq 0$, by (iv) we know that the ideals $I^0, I^1, \dots, I^{n \cdot 2^k-1}$ are all distinct, the t which satisfies that $I^t = (r)$ should be unique.

(vi) Since R is finite, all ideals of R are finitely-generated. Say that $J = (r_1, r_2, \dots, r_u)$ is an ideal of R . By (v) we assume that $(r_i) = I^{t_i}$ for all i , then $J = (r_1, \dots, r_u) = (I^{t_1}, \dots, I^{t_u}) = I^{\min(t_1, \dots, t_u)}$.

4.2 Ciphertext distribution of Saber

Recall that when S is a finite set, $x = \mathcal{U}(S)$ means that x is a discrete random variable such that $P(x = s) = 1/|S|$ for all $s \in S$.

Theorem 4.3. Let R be a finite commutative ring, $a_1, \dots, a_l = \mathcal{U}(R)$ be independent and $s_1, \dots, s_l \in R$. Then we have

$$a_1 s_1 + \dots + a_l s_l = \mathcal{U}((s_1, \dots, s_l))$$



Proof. Let $I = (s_1, \dots, s_l)$ be the ideal in R . We define

$$\begin{aligned} \varphi : \quad R^l &\rightarrow I \\ (a_1, \dots, a_l) &\mapsto a_1 s_1 + \dots + a_l s_l \end{aligned}$$

It is clear that φ is a surjective additive group homomorphism, thus for all $s \in I$ we have $|\varphi^{-1}(s)| = |\ker(\varphi)|$. Therefore, $P(a_1 s_1 + \dots + a_l s_l = s) = |\ker(\varphi)|/|I|$ is a constant with respect to $s \in I$, which proves the result.

With the theorems established above, we may start our works in this section. Our goal is to prove that the distribution of ciphertext of Saber is uniform, hence no insecurity is caused.

Theorem 4.4. Let $R = \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$ and $I = (x - 1)$ be an ideal of R . Given $s_1, \dots, s_l \in R$ such that they are not all 0. Then $(s_1, \dots, s_l) = I^t$ if and only if:

- (i) $s_i \in I^t$ for all i .
- (ii) There exists i such that $s_i \notin I^{t+1}$.

Proof. \Rightarrow : By Theorem 4.2(vi) the t must exist. (i) is clear. Assume that (ii) does not hold, then we will have $(s_1, \dots, s_l) \subseteq I^{t+1} \subsetneq I^t$ since I^t is not the zero ideal, which leads to a contradiction.

At last, since the conditions of (i)(ii) combined are disjoint for each t , the converse automatically holds.



Theorem 4.5. Let $R_n = \mathbb{Z}_{2^n}[x]/(x^{2^k} + 1)$ and $R_m = \mathbb{Z}_{2^m}[x]/(x^{2^k} + 1)$, where $n, m, k \in \mathbb{N}^*$ and $n > m$. Fix $h \in R_n$, we define the rounding φ as follows:

$$\begin{aligned} \varphi: R_n &\rightarrow R_m \\ r &\mapsto (r + h) \gg (n - m) \end{aligned}$$

Let $I = (x - 1)$ be an ideal of R_n . If $r = \mathcal{U}(I^t)$, where $t \leq (n - m)2^k$, then we have $\varphi(r) = \mathcal{U}(R_m)$. Furthermore, $\varphi(r)|(r - 2^{n-m}\varphi(r)) = \mathcal{U}(R_m)$.

Proof. Let $J = I^{(n-m)2^k} = (2^{n-m})$. Since $t \leq (n - m)2^k$, $J \leq I^t$ as additive subgroup, thus we may write I^t as a disjoint union of cosets of J . That is, $I^t = \cup_i (r_i + J)$.

For any coset $r_i + J$, we observe that φ restricted on $r_i + J$ is injective: First if $\varphi(s_1) = \varphi(s_2)$ for $s_1, s_2 \in r_i + J$, then we may write $s_2 = s_1 + t$ for some $t \in J = (2^{n-m})$. Since t is a multiple of 2^{n-m} , we have $\varphi(s_2) = \varphi(s_1 + t) = \varphi(s_1) + \frac{t}{2^{n-m}}$. Hence $t = 0$ and we may conclude that $\varphi|_{r_i+J}$ is injective. Note that $|r_i + J| = |R_m| = 2^{m2^k}$, we know that $\varphi|_{r_i+J}$ is bijective. Therefore, conditioning on $r \in r_i + J$ we have $\varphi(r) = \mathcal{U}(R_m)$ for each coset $r_i + J$, thus $\varphi(r) = \mathcal{U}(R_m)$. Since when r_i is fixed, for any $r \in r_i + J$ the value of $r - 2^{n-m}\varphi(r)$ is the same, the last result follows.

The following theorems will follow the notations in Chapter 3. Especially, we note that $R_q = \mathbb{Z}_q[x]/(x^{256} + 1)$ and $q = 2^{\epsilon_q}$. Furthermore, if $\mathbf{v} \in R_q^{l \times 1}$ then v_i denotes the i -th entry

of \mathbf{v} for $i = 1, 2, \dots, l$.

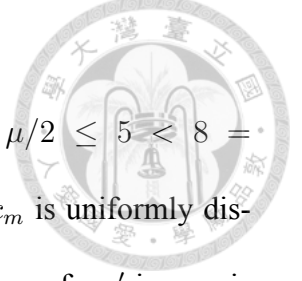


Theorem 4.6. Let $\mathbf{A} = \mathcal{U}(R_q^{l \times l})$ and $I = (x - 1)$ be an ideal. Fix $\mathbf{h} \in R_q^{l \times 1}$ and given \mathbf{s}, \mathbf{s}' . If there exists $u \leq l$ such that $s_u \notin I^{256(\epsilon_q - \epsilon_p) + 1}$, then $\mathbf{b} = (\mathbf{A}^T \mathbf{s} + \mathbf{h}) \gg (\epsilon_q - \epsilon_p) = \mathcal{U}(R_p^{l \times 1})$. Furthermore, if there exists $v \leq l$ such that $s'_v \notin I^{256(\epsilon_p - \epsilon_T) + 1}$ and fix $h_1 \in R_p$, then $c_m = (\mathbf{b}^T \mathbf{s}' + h_1 + 2^{\epsilon_p - 1} m) \gg (\epsilon_p - \epsilon_T) = \mathcal{U}(R_T)$.

Proof. For each i , we have $b_i = (A_{i1}s_1 + A_{i2}s_2 + \dots + A_{il}s_l) \gg (\epsilon_q - \epsilon_p)$. Let $r = A_{i1}s_1 + A_{i2}s_2 + \dots + A_{il}s_l$ then by Theorem 4.3. we have $r = \mathcal{U}((s_1, \dots, s_l))$. By Theorem 4.2(vi), we may assume $(s_1, \dots, s_l) = I^t$ for some t , and since $s_u \notin I^{256(\epsilon_q - \epsilon_p) + 1}$ we may use Theorem 4.4 to conclude that $I^{256(\epsilon_q - \epsilon_p)} \subseteq (s_u) \subseteq (s_1, \dots, s_l) = I^t$, hence $t \leq 256(\epsilon_q - \epsilon_p)$ and $r = \mathcal{U}(I^t)$. By Theorem 4.5, we have $b_i = r \gg (\epsilon_q - \epsilon_p) = \mathcal{U}(R_p)$. At last, since all entries of \mathbf{A} are independent, all entries of \mathbf{b} are also independent, thus $\mathbf{b} = \mathcal{U}(R_p^{l \times 1})$. The other part is similar.

Corollary 4.7. If $\max(\|\mathbf{s}\|, \|\mathbf{s}'\|) < \min(q/p, p/T)$, then conditioning on $\mathbf{s} \neq 0$ and $\mathbf{s}' \neq 0$, we have $c_m = \mathcal{U}(R_T)$.

Proof. For any fixed \mathbf{s}, \mathbf{s}' satisfying the given conditions, we may note that since $\mathbf{s} \neq 0$, one of the coefficients of \mathbf{s} is nonzero. Since $\|\mathbf{s}\| < q/p$, this nonzero coefficient is not a multiple of q/p . Therefore, \mathbf{s} is not a multiple of q/p , which implies that $\mathbf{s} \notin (q/p) = (2^{\epsilon_q - \epsilon_p}) = I^{256(\epsilon_q - \epsilon_p)}$, where $I = (x - 1)$ is an ideal. Similarly, $\mathbf{s}' \notin (p/T) = I^{256(\epsilon_p - \epsilon_T)}$. Hence $c_m = \mathcal{U}(R_T)$ by Theorem 4.6. At last, since $c_m = \mathcal{U}(R_T)$ holds for all fixed nonzero \mathbf{s} and \mathbf{s}' , it holds when conditioning on the event that $\mathbf{s} \neq 0$ and $\mathbf{s}' \neq 0$.



For LightSaber, Saber, and FireSaber, we have $\max(\|\mathbf{s}\|, \|\mathbf{s}'\|) \leq \mu/2 \leq 5 < 8 = \min(q/p, p/T)$. Hence whenever \mathbf{s}, \mathbf{s}' are both nonzero, the ciphertext c_m is uniformly distributed on the ciphertext space R_T . Since the probability that at least one of \mathbf{s}, \mathbf{s}' is zero is negligible, we may say that the ciphertext of Saber is uniformly distributed.

4.3 Error rate of Saber

As one of the round-3 candidates of the NIST post-quantum cryptography competition, Saber surely has an officially-proposed error rate estimation. However, we will see some flaws in the estimations. Thus, the results are questionable. This section will point out the flaws and figure out some approaches to remedy them.

First of all, we see the formulas for encryption and decryption in Saber:

$$c_m = (v' + h_1 - 2^{\epsilon_p-1}m) \gg (\epsilon_p - \epsilon_T) = (v' + h_1) \gg (\epsilon_p - \epsilon_T) - 2^{\epsilon_T-1}m$$

$$m' = (v - 2^{\epsilon_p-\epsilon_T}c_m + h_2) \gg (\epsilon_p - 1)$$

The coefficients of h_1 are all set to be $2^{\epsilon_q-\epsilon_p-1}$ while those of h_2 are set to be $2^{\epsilon_p-2} - 2^{\epsilon_p-\epsilon_T+1} + 2^{\epsilon_q-\epsilon_p-1}$. Our target is to compute $P(m \neq m')$. For the sake of convenience, we let $c_1 = 2^{\epsilon_q-\epsilon_p-1}$ and $c_2 = 2^{\epsilon_p-2} - 2^{\epsilon_p-\epsilon_T+1} + 2^{\epsilon_q-\epsilon_p-1}$ be the coefficients of h_1 and h_2 respectively. Furthermore, for $h \in \mathbb{Z}[x]/(x^{256} + 1)$, we write $h = \sum_{i=0}^{255} a_i x^i$ and define $M(h) = \max\{a_i\}$,

$L(h) = \min\{a_i\}$ and $\delta_i(h) = a_i$ for $0 \leq i \leq 255$. We first note that

$$\begin{aligned} m' &= (v - 2^{\epsilon_p - \epsilon_T} c_m + h_2) \gg (\epsilon_p - 1) \\ &= (2^{\epsilon_T - 1} m + v - 2^{\epsilon_p - \epsilon_T} ((v' + h_1) \gg (\epsilon_p - \epsilon_T))) + h_2 \gg (\epsilon_p - 1) \end{aligned}$$



Therefore, $m = m'$ when the following condition holds:

$$\begin{cases} M(v - 2^{\epsilon_p - \epsilon_T} ((v' + h_1) \gg (\epsilon_p - \epsilon_T))) < \frac{p}{2} - c_2 \\ L(v - 2^{\epsilon_p - \epsilon_T} ((v' + h_1) \gg (\epsilon_p - \epsilon_T))) \geq -c_2 \end{cases}$$

Although these are not necessary conditions for $m = m'$ since the coefficients are in \mathbb{Z}_p , it is still clear that there is likely a decryption error when the conditions above fail to hold.

We write

$$e'_r = v' - 2^{\epsilon_p - \epsilon_T} ((v' + h_1) \gg (\epsilon_p - \epsilon_T))$$

Then the conditions become

$$\begin{cases} M(v - v' + e'_r) < \frac{p}{2} - c_2 \\ L(v - v' + e'_r) \geq -c_2 \end{cases}$$

On the other hand, we may observe that the value of e'_r is uniquely determined by $(v' \bmod 2^{\epsilon_p - \epsilon_T})$,

and for each i we have $-c_1 \leq \delta_i(e'_r) < 2^{\epsilon_p - \epsilon_T} - c_1$. We note that if we let $e_r = e'_r + (c_1 -$

$2^{\epsilon_p - \epsilon_T - 1})(1 + x + \dots + x^{255})$, then we have $-2^{\epsilon_p - \epsilon_T - 1} \leq \delta_i(e_r) < 2^{\epsilon_p - \epsilon_T - 1} - 1$ for all i and

the conditions become

$$\begin{cases} M(v - v' + e_r) < p/4 \\ L(v - v' + e_r) \geq -p/4 \end{cases}$$



These conditions are as if $(c_1, c_2) = (2^{\epsilon_p - \epsilon_T - 1}, 2^{\epsilon_p - 2})$ were the constants for normal roundings, and e_r is also uniquely determined by $(v' \bmod 2^{\epsilon_p - \epsilon_T})$. This explains why when we compute the error rate of Saber we may compute as if $(c_1, c_2) = (2^{\epsilon_p - \epsilon_T - 1}, 2^{\epsilon_p - 2})$.

We recall that $v = \mathbf{b}'^T \mathbf{s}$ and $v' = \mathbf{b}^T \mathbf{s}'$. Furthermore, $\mathbf{b} = (\mathbf{A}^T \mathbf{s}) \gg (\epsilon_q - \epsilon_p)$ and $\mathbf{b}' = (\mathbf{A} \mathbf{s}') \gg (\epsilon_q - \epsilon_p)$. We want to lift to R_q to analyze the formula, thus we define

$$\begin{cases} \mathbf{e} = \frac{q}{p} \mathbf{b} - \mathbf{A}^T \mathbf{s} \in R_q^{l \times 1} \\ \mathbf{e}' = \frac{q}{p} \mathbf{b}' - \mathbf{A} \mathbf{s}' \in R_q^{l \times 1} \end{cases}$$

We have

$$\begin{aligned} v - v' &= \mathbf{b}'^T \mathbf{s} - \mathbf{b}^T \mathbf{s}' \\ &= \frac{p}{q} ((\mathbf{A} \mathbf{s}' + \mathbf{e}')^T \mathbf{s} - (\mathbf{A}^T \mathbf{s} + \mathbf{e})^T \mathbf{s}') \\ &= \frac{p}{q} (\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + (\mathbf{s}'^T \mathbf{A}^T \mathbf{s} - \mathbf{s}^T \mathbf{A} \mathbf{s}')) \\ &= \frac{p}{q} (\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') \end{aligned}$$

The last equality is because $(\mathbf{s}'^T \mathbf{A}^T \mathbf{s})^T = \mathbf{s}^T \mathbf{A} \mathbf{s}'$ and thus as 1×1 matrices they are equal.

The conditions for a successful decryption finally become



$$\begin{cases} M(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) < q/4 \\ L(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) \geq -q/4 \end{cases}$$

Therefore, letting \mathcal{P} denote the error rate, we may bound \mathcal{P} as follows:

$$\begin{aligned} \mathcal{P} &\leq P(M(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) \geq q/4) + P(L(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) < -q/4) \\ &\leq \sum_{i=0}^{255} P(\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) \geq q/4) + \\ &\quad \sum_{i=0}^{255} P(\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) < -q/4) \end{aligned}$$

Hence it remains to compute the distribution of $\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r)$ for each i . It is easy to see that the distribution is identical for each i , thus we may work one any specific i . To sum up, for any $0 \leq i \leq 255$ we have

$$\mathcal{P} \leq 256 \left(P \left(\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) \geq q/4 \right) + P \left(\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r) < -q/4 \right) \right)$$

4.3.1 Some important lemmas

Saber cited Lemma 5.5 and Theorem 5.2 of another paper by Z. Jin and Y. Zhao [4] and claimed that analogs to them can be used to estimate the error rate of Saber without explicitly interpreting these lemmas. Since it is necessary to carefully survey the content and conditions of these lemmas to determine whether an algorithm to estimate the error rate of Saber is

correct, we first establish the lemmas in this subsection.

For the following lemmas we define $\lfloor x \rfloor = \lfloor x + \frac{1}{2} \rfloor$ for any $x \in \mathbb{R}$. q, p are fixed as the parameters of Saber, that is, 8192 and 1024 respectively, though the lemmas hold whenever p, q are power-of-2 and $p < q$. We define $\lfloor x \rfloor_p = \lfloor \frac{p}{q}x \rfloor$ and $\{x\}_p = x - \frac{p}{q} \lfloor x \rfloor_p$ for $x \in \mathbb{Z}$. These notations are defined so that $\mathbf{b} = (\mathbf{A}^T \mathbf{s} + \mathbf{h}) \gg (\epsilon_q - \epsilon_p) = \lfloor \mathbf{A}^T \mathbf{s} \rfloor_p$ and similarly $\mathbf{b}' = \lfloor \mathbf{A} \mathbf{s}' \rfloor_p$, thus

$$\begin{cases} \mathbf{e} = \frac{q}{p} \mathbf{b} - \mathbf{A}^T \mathbf{s} = -\{\mathbf{A}^T \mathbf{s}\}_p \\ \mathbf{e}' = \frac{q}{p} \mathbf{b}' - \mathbf{A} \mathbf{s}' = -\{\mathbf{A} \mathbf{s}'\}_p \end{cases}$$

We note that we may regard the operator $\{\cdot\}_p$ as a map from R_q to $R_{q/p}$. By doing so, there is an extra advantage that the operator $\{\cdot\}_p$ is an additive group homomorphism.

Let $\text{Inv}(\mathbf{s}_1, \mathbf{s}_2)$ denote the event that there exist invertible elements of ring $R_{q/p}$ in both vectors \mathbf{s}_1 and \mathbf{s}_2 . This plays an important role in the following lemmas, but on the other hand results in a flaw in the estimation of error rate of Saber.

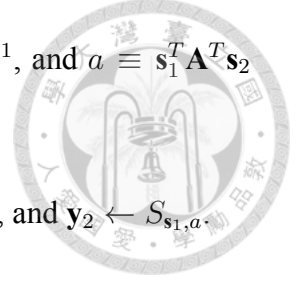
Lemma 4.8. Define $S_{\mathbf{x},a} = \{\mathbf{y} \in R_{q/p}^l \mid \mathbf{x}^T \mathbf{y} \bmod (q/p) = a\}$ for $a \in R_{q/p}$ and $\mathbf{x} \in R_{q/p}^l$. Fix $a \in R_{q/p}$. Conditioned on $\text{Inv}(\mathbf{s}_1, \mathbf{s}_2)$ and $\mathbf{s}_1^T \mathbf{A}^T \mathbf{s}_2 \bmod (q/p) = a$, the random variables $\{\mathbf{A}^T \mathbf{s}_2\}_p$ and $\{\mathbf{A} \mathbf{s}_1\}_p$ are independent and are subjected to uniform distribution over $S_{\mathbf{s}_1,a}$ and $S_{\mathbf{s}_2,a}$ respectively.

Proof. A simple analogue of the proof of Lemma 5.5 in [4] holds.

Lemma 4.9. Under the condition $\text{Inv}(\mathbf{s}_1, \mathbf{s}_2)$, the following two distributions are identical:

(i) $(a, \mathbf{s}_1, \mathbf{s}_2, \{\mathbf{A}\mathbf{s}_1\}_p, \{\mathbf{A}^T \mathbf{s}_2\}_p)$, where $\mathbf{A} \leftarrow R_q^{l \times l}, \mathbf{s}_1 \leftarrow \chi^{l \times 1}, \mathbf{s}_2 \leftarrow \chi^{l \times 1}$, and $a \equiv \mathbf{s}_1^T \mathbf{A}^T \mathbf{s}_2 \pmod{(q/p)}$

(ii) $(a, \mathbf{s}_1, \mathbf{s}_2, \mathbf{y}_1, \mathbf{y}_2)$, where $a \leftarrow R_{q/p}, \mathbf{s}_1 \leftarrow \chi^{l \times 1}, \mathbf{s}_2 \leftarrow \chi^{l \times 1}, \mathbf{y}_1 \leftarrow S_{\mathbf{s}_2, a}$, and $\mathbf{y}_2 \leftarrow S_{\mathbf{s}_1, a}$.



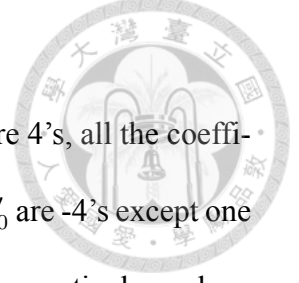
Proof. A simple analogue of the proof of Theorem 5.2 in [4] holds.

4.3.2 Flaw of the original estimation

We may conclude from Lemma 4.9 that $\mathbf{e}'^T \mathbf{s}$ and $\mathbf{e}^T \mathbf{s}'$ are independent if conditioned on $\mathbf{s}'^T \mathbf{A} \mathbf{s} \equiv a \pmod{q/p}$, where $a \in R_{q/p}$. However, the python script provided by the submitters of Saber to compute the error rate of Saber has assumed that $\mathbf{e}'^T \mathbf{s}$ and $\mathbf{e}^T \mathbf{s}'$ are independent if conditioned on $\delta_i(\mathbf{s}'^T \mathbf{A} \mathbf{s}) \equiv a \pmod{q/p}$, where $a \in Z_{q/p}$ and $0 \leq i \leq 255$.

Unfortunately, this assumption cannot be true and is likely to underestimate the error rate.

We will show by a counterexample. Let us say that we want to compute the distribution of $\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}')$ of the NIST level-3 Saber under the assumption that $\text{Inv}(\mathbf{s}, \mathbf{s}')$. Especially, we find the largest possible value of $\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}')$ and then find the probability for the occurrence of this value. We may easily see from definition that the coefficients of \mathbf{e}, \mathbf{e}' are in the range $[-3, 4]$, and for the NIST level-3 Saber the coefficients of \mathbf{s} and \mathbf{s}' are in the range $[-4, 4]$. Therefore, to make the value of $\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}')$ be largest, it is clear that the all the coefficients of \mathbf{e}' and \mathbf{e} should be 4's, all the coefficients of \mathbf{s} should be 4's, and all the coefficients of \mathbf{s}' should be -4's. However, this violates the condition $\text{Inv}(\mathbf{s}, \mathbf{s}')$, and to fix it we need to add one of the coefficients of \mathbf{s} by -1 and add one of the coefficients of \mathbf{s}' by 1. In this case we may check that $\mathbf{e}'^T \mathbf{s} \equiv \mathbf{e}^T \mathbf{s}' \equiv 4x^{255} + \dots + 4 \pmod{q/p}$, thus by the previous



lemmas this combination of $(\mathbf{e}'^T, \mathbf{s}, \mathbf{e}, \mathbf{s}')$ is possible to occur.

Here, we define $\mathbf{e}_0, \mathbf{s}_0, \mathbf{s}'_0 \in R_q^{3 \times 1}$ such that all the coefficients of \mathbf{e}_0 are 4's, all the coefficients of \mathbf{s}_0 are 4's except one of them being 3, and all the coefficients of \mathbf{s}'_0 are -4's except one of them being -3. There are $256 \times 3 = 768$ different possible \mathbf{s}_0 and \mathbf{s}'_0 respectively, and we just fix an arbitrary one. We have $\delta_{255}(\mathbf{e}_0^T \mathbf{s}_0 - \mathbf{e}_0^T \mathbf{s}'_0) = 24568$, which is the largest possible value as discussed previously.

We first compute the correct $P(\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') = 24568)$ by Lemma 4.9. By our previous discussion we have $P(\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') = 24568) = 768^2 P((\mathbf{e}', \mathbf{s}, \mathbf{e}, \mathbf{s}') = (\mathbf{e}_0, \mathbf{s}_0, \mathbf{e}_0, \mathbf{s}'_0))$. Write $\kappa = 4x^{255} + \dots + 4$. By Lemma 4.9 we have

$$P((\mathbf{e}', \mathbf{s}, \mathbf{e}, \mathbf{s}') = (\mathbf{e}_0, \mathbf{s}_0, \mathbf{e}_0, \mathbf{s}'_0)) = \frac{1}{|R_{q/p}|} P(\mathbf{s} = \mathbf{s}_0) P(\mathbf{s}' = \mathbf{s}'_0) \cdot \frac{1}{|S_{\mathbf{s}_0, \kappa}|} \cdot \frac{1}{|S_{\mathbf{s}'_0, \kappa}|}$$

By Theorem 4.3, let $\mathbf{x} = \mathcal{U}(\{\{\mathbf{y}\}_p | \mathbf{y} \in R_q^l\})$ we have $P(\mathbf{s}_0^T \mathbf{x} \equiv \kappa \pmod{q/p}) = \frac{1}{|R_{q/p}|}$. Thus we may conclude that $|S_{\mathbf{s}_0, \kappa}| = |S_{\mathbf{s}'_0, \kappa}| = \frac{|R_{q/p}|^3}{|R_{q/p}|}$ and hence

$$P((\mathbf{e}', \mathbf{s}, \mathbf{e}, \mathbf{s}') = (\mathbf{e}_0, \mathbf{s}_0, \mathbf{e}_0, \mathbf{s}'_0)) = \frac{1}{|R_{q/p}|^5} P(\mathbf{s} = \mathbf{s}_0) P(\mathbf{s}' = \mathbf{s}'_0)$$

Therefore,

$$P(\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') = 24568) = \frac{768^2}{|R_{q/p}|^5} P(\mathbf{s} = \mathbf{s}_0) P(\mathbf{s}' = \mathbf{s}'_0)$$

Next, let us try to compute $P(\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') = 24568)$ by the assumption that $\mathbf{e}'^T \mathbf{s}$ and $\mathbf{e}^T \mathbf{s}'$ are independent if conditioned on $\delta_{255}(\mathbf{s}'^T \mathbf{A} \mathbf{s}) \equiv a \pmod{q/p}$. By Lemma 4.9, we may conclude that $P(\delta_{255}(\mathbf{e}'^T \mathbf{s}) \equiv a \pmod{q/p}) = \frac{p}{q} = \frac{1}{8}$ for any a . Therefore, let \mathcal{A} denote the

event that $\delta_{255}(\mathbf{e}'^T \mathbf{s}') \equiv \delta_{255}(\mathbf{e}'^T \mathbf{s}) \equiv \delta_{255}(\mathbf{e}_0^T \mathbf{s}_0) \pmod{q/p}$, we have

$$\begin{aligned} P((\mathbf{e}', \mathbf{s}, \mathbf{e}, \mathbf{s}') = (\mathbf{e}_0, \mathbf{s}_0, \mathbf{e}_0, \mathbf{s}'_0)) &= P(\mathcal{A})P((\mathbf{e}', \mathbf{s}) = (\mathbf{e}_0, \mathbf{s}_0)|\mathcal{A})P((\mathbf{e}, \mathbf{s}') = (\mathbf{e}_0, \mathbf{s}'_0)|\mathcal{A}) \\ &= P(\mathcal{A}) \cdot \frac{P(\mathbf{e}' = \mathbf{e}_0)P(\mathbf{s} = \mathbf{s}_0)}{P(\mathcal{A})} \cdot \frac{P(\mathbf{e} = \mathbf{e}_0)P(\mathbf{s}' = \mathbf{s}'_0)}{P(\mathcal{A})} \\ &= \frac{1}{|R_{q/p}|^6 P(\mathcal{A})} P(\mathbf{s} = \mathbf{s}_0)P(\mathbf{s}' = \mathbf{s}'_0) \end{aligned}$$

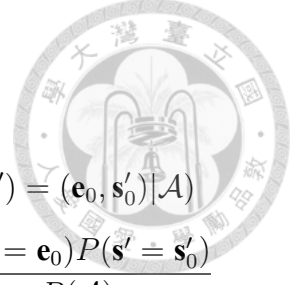
Therefore, under this assumption we have

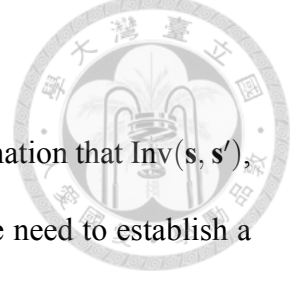
$$P(\delta_{255}(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') = 24568) = \frac{768^2}{|R_{q/p}|^6 / 8} P(\mathbf{s} = \mathbf{s}_0)P(\mathbf{s}' = \mathbf{s}'_0)$$

It is 2^{-765} times of the correct probability.

As we may see in this example, this assumption tends to underestimate the probability that $\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}')$ is extremely large for some i . This is the major flaw of the original estimation of the error rate of Saber. Unfortunately, it is computationally infeasible to compute the error rate of Saber by the same algorithm and Lemma 4.9, since there are $(q/p)^{256} = 2^{768}$ possible values of a .

There is another flaw of using Lemma 4.9 to estimate the error rate of Saber. That is, the condition $\text{Inv}(\mathbf{s}, \mathbf{s}')$ may fail to hold with non-negligible probability. By Theorem 4.1, we may conclude that the probability of that s_i is invertible in $R_{q/p}$ is $\frac{1}{2}$ for each i , hence $\text{Inv}(\mathbf{s})$ fails to hold with probability 2^{-l} , where $l = 2, 3, 4$ in LightSaber, Saber, and FireSaber respectively. This is far from negligible in cryptography, thus we find another flaw to fix.





4.3.3 A correct bound of the error rate of Saber

In this subsection, we first estimate the error rate of Saber under the estimation that $\text{Inv}(\mathbf{s}, \mathbf{s}')$, and then deal with the case that $\text{Inv}(\mathbf{s}, \mathbf{s}')$ does not hold. Before that, we need to establish a lemma to help the estimation of error rate. Let $\text{Ind}(a, b)$ denote that two random variables a, b are independent.

Lemma 4.10. e_r is independent of $\mathbf{e}^T \mathbf{s}'$ if conditioned on $\mathbf{s} \neq 0$ and $\|\mathbf{s}\| < q/p$.

Proof. First we note that e_r is uniquely determined by v' , so it suffices to prove that $\text{Ind}(v', \mathbf{e}^T \mathbf{s}')$. By definition, $\mathbf{e} = -\{\mathbf{A}^T \mathbf{s}\}_p$ and $\mathbf{b} = \lfloor \mathbf{A}^T \mathbf{s} \rfloor_p$. By Theorem 4.3, $\mathbf{A}^T \mathbf{s} = (s_1, \dots, s_l)$. Conditioned on $\mathbf{s} \neq 0$ and $\|\mathbf{s}\| < q/p$, we have $\mathbf{A}^T \mathbf{s} = \mathcal{U}(I^t)$ for some $t < 256(q/p)$, where $I = (x - 1)$, thus by Theorem 4.5, we have $\mathbf{b}|\mathbf{e} = \mathcal{U}(R_m)$ conditioning on any fixed $\mathbf{s} \neq 0$, and thus $\mathbf{b}|\mathbf{e} = \mathcal{U}(R_m)$ conditioning on $\mathbf{s} \neq 0$. This implies that $\text{Ind}(\mathbf{b}, \mathbf{e})$. At last, we have

$$\text{Ind}(\mathbf{b}, \mathbf{e}) \Rightarrow \text{Ind}(\mathbf{b}^T \mathbf{s}', \mathbf{e}^T \mathbf{s}') \Rightarrow \text{Ind}(v', \mathbf{e}^T \mathbf{s}').$$

This completes the proof.

Recall that we want to bound the error rate by

$$256(P(\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p}e_r) \geq q/4) + P(\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}' + \frac{q}{p}e_r) < -q/4))$$

Under the condition that $\text{Inv}(\mathbf{s}, \mathbf{s}')$, the coefficients of \mathbf{e} are all uniformly distributed in $[-3, 4]$

and independent, and the coefficients of \mathbf{s} follow the centered binomial distribution and are independent. Hence the distribution of $\delta_i(\mathbf{e}'^T \mathbf{s})$ may be calculated by simple convolutions. The same holds for $\mathbf{e}^T \mathbf{s}'$, and since $\text{Ind}(\mathbf{e}^T \mathbf{s}', e_r)$ by Lemma 4.10, we may also calculate the distribution of $\delta_i(-\mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r)$ by convolution.

The problem is that $\mathbf{e}'^T \mathbf{s}$ and $-\mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r$ are dependent and so far we have no computationally feasible way to calculate the distribution. Thus here we use the worst estimation: Let $X = \delta_i(\mathbf{e}'^T \mathbf{s})$ and $Y = \delta_i(-\mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r)$, for all $k \in \mathbb{Z}$ we have

$$P(X + Y > q/4 - 1) \leq P(X > k) + P(Y > q/4 - 1 - k)$$

$$P(X + Y < -q/4) \leq P(X < k) + P(Y < -q/4 - k)$$

Therefore, we may find k to get the optimal bounds for these two terms, and then we may get a bound for the error rate of Saber.

Our C++ code is attached in Appendix A. We get the results as follows:

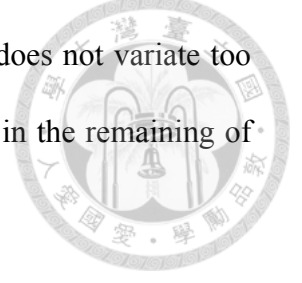
Scheme	Security Category	Claimed Error Rate	Estimated Error Rate
LightSaber	1	2^{-120}	$2^{-56.7}$
Saber	3	2^{-136}	$2^{-64.7}$
FireSaber	5	2^{-165}	$2^{-79.3}$

Table 4.1: Error rates of Saber

Our estimated error rates are about the square root of the claimed error rates.

It remains to deal with the case that $\text{Inv}(\mathbf{s}, \mathbf{s}')$ does not hold. It turns out that we may prove

that when we consider the case that $\text{Inv}(\mathbf{s}, \mathbf{s}')$ does not hold, the result does not vary too much. We summarize our result as the following theorem and prove it in the remaining of this subsection.



Theorem 4.11. Let $\mathbf{A} = \mathcal{U}(R_q^{l \times l})$, $\mathbf{s}, \mathbf{s}' = \beta_\mu(R_q^{l \times 1})$ with $\mu > 0$, and $\mathbf{e}, \mathbf{e}', e_r, \text{Inv}(\mathbf{s}, \mathbf{s}')$ are as previously defined. Let $X = \delta_i(\mathbf{e}'^T \mathbf{s})$ and $Y = \delta_i(-\mathbf{e}^T \mathbf{s}' + \frac{q}{p} e_r)$. If we define

$$\mathcal{P}_k^+ = P(X \geq k | \text{Inv}(\mathbf{s}, \mathbf{s}')) + P(Y \geq q/4 - k | \text{Inv}(\mathbf{s}, \mathbf{s}')),$$

$$\mathcal{P}_k^- = P(X \leq k | \text{Inv}(\mathbf{s}, \mathbf{s}')) + P(Y \leq -q/4 - k - 1 | \text{Inv}(\mathbf{s}, \mathbf{s}')),$$

$$\mathcal{P}'_k^+ = P(X \geq k) + P(Y \geq q/4 - k),$$

and

$$\mathcal{P}'_k^- = P(X \leq k) + P(Y \leq -q/4 - k - 1).$$

Then we have

$$\min_{(k_1, k_2)} (\mathcal{P}'_{k_1}^+ + \mathcal{P}'_{k_2}^-) \leq 1.01 \min_{(k_1, k_2)} (\mathcal{P}_{k_1}^+ + \mathcal{P}_{k_2}^-) + 2^{-200}.$$

Note that $\min_{(k_1, k_2)} (\mathcal{P}'_{k_1}^+ + \mathcal{P}'_{k_2}^-)$ is the desired bound for error rates while $\min_{(k_1, k_2)} (\mathcal{P}_{k_1}^+ + \mathcal{P}_{k_2}^-)$ is the results in the rightmost of Table 4.1. We may see that this affects very little to the exponent of 2. Therefore, our results in Table 4.1 need no further correction.

Proof. Since in our approach the dependency of $\mathbf{e}'^T \mathbf{s}$ and $\mathbf{e}^T \mathbf{s}'$ does not matter, we just need to

find the distribution of $\delta_i(\mathbf{e}^T \mathbf{s})$ and $\delta_i(\mathbf{e}^T \mathbf{s}')$ respectively without the assumption that $\text{Inv}(\mathbf{s}, \mathbf{s}')$.

These two distributions are the same by symmetry.

Recall that $\mathbf{e} = -\{\mathbf{A}^T \mathbf{s}\}_p$. By Theorem 4.3 and 4.2(vi), for \mathbf{s} fixed we have $\mathbf{A}^T \mathbf{s} = \mathcal{U}((I^t)^{l \times 1})$ for some t , where $I = (x - 1)$. Since $\{\cdot\}_p$ is an operator from R_q to $R_{q/p}$ which may be regarded as a surjective module homomorphism, we have $\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})$, where this $I = (x - 1)$ is regarded as an ideal of $R_{q/p}$.

We then want to find the probability of $\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})$, which depends on the distribution of \mathbf{s} .

Lemma 4.12. Let $\mu > 0$ and $s = \beta_\mu(R_q)$, that is, $s \in R_q$ and the coefficients of s are subjected to a centered binomial distribution with range $[-\mu/2, \mu/2]$. Let $I = (x - 1)$ be an ideal of R_q . Then for $t \leq 256$ we have $P(s \in I^t) = 2^{-t}$. If $\mathbf{s} = \beta_\mu(R_q^{l \times 1})$, $\mathbf{A} = \mathcal{U}(R_q^{l \times l})$, and $\mathbf{e} = -\{\mathbf{A}^T \mathbf{s}\}_p$, then for $t < 256$ we have $P(\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})) = 2^{-tl} - 2^{-(t+1)l}$.

Proof of lemma. Since $t \leq 256$, by Theorem 4.2 we have $(2) = I^{256} \subseteq I^t$, hence $I^t = ((x - 1)^t, 2)$. So $s \in I^t$ implies that $\bar{s} \in R_2$ is a multiple of $(x - 1)^t$. There are 2^{256-t} multiples of $(x - 1)^t$ in R_2 , and since we know that the coefficients of s have probability $\frac{1}{2}$ to be even and the other $\frac{1}{2}$ to be odd, $\bar{s} \in R_2$ has an odd of 2^{-256} to be equal to one of the multiples. Therefore we may conclude that $P(s \in I^t) = 2^{-256} \cdot 2^{256-t} = 2^{-t}$.

At last, by Theorem 4.4, for $t < 256$ we have

$$\begin{aligned}
P(\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})) &= P(\mathbf{A}^T \mathbf{s} = \mathcal{U}((I^t)^{l \times 1})) \\
&= P((s_1, \dots, s_l) = I^t) \\
&= \prod_{i=1}^l P(s_i \in I^t) - \prod_{i=1}^l P(s_i \in I^{t+1}) \\
&= 2^{-tl} - 2^{-(t+1)l}
\end{aligned}$$



This completes the proof of lemma.

Next, we want to find the distribution of $\delta_i(\mathbf{e}^T \mathbf{s}')$ conditioned on $\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})$, which may be easily derived by convolution if we know the distribution of $\delta_i(e_1^T s'_1)$ conditioned on $e_1 = \mathcal{U}(I^t)$. It's easy to see that distribution of $\delta_i(e_1^T s'_1)$ is the same for all i . Here we choose to compute the distribution of $\delta_{255}(e_1^T s'_1) = \sum_{i=0}^{255} \delta_i(e_1) \delta_{255-i}(s'_1)$ conditioned on $e_1 = \mathcal{U}(I^t)$ with $t < 256$.

To solve this problem we need some definitions and observations first. In the following, a polynomial is called k -**binary** if the degree of the polynomial is less than k and each coefficient of the polynomial is either 0 or 1. We first see that for any $r \in R_{q/p}$ there exists unique t -binary polynomial s such that $r - s \in I^t$: This is because by Theorem 4.2 we have $I^t = ((x-1)^t, 2)$ and thus we may divide r by $(x-1)^t$ first and then divide the coefficients by 2 to get a t -binary polynomial s as remainder, and it is clear that such s is unique.

Let \mathcal{R} denote the set of finite support functions from \mathbb{Z} to \mathbb{R} equipped with point-wise addition as addition, discrete convolution as multiplication, and point-wise \mathbb{R} -scalar multiplication as \mathbb{R} -scalar multiplication. It is easy to check that \mathcal{R} is a commutative \mathbb{R} -algebra. A

probability distribution of a bounded integer-valued random variable X may be regarded as an element of \mathcal{R} as we may define $f \in \mathcal{R}$ such that $f(n) = P(X = n)$ for all $n \in \mathbb{Z}$.

Now we may proceed on the problem. Let $y_0, \dots, y_{255} = \mathcal{U}([-3, 4])$ and $z_0, \dots, z_{255} = \beta_\mu$ be independent random variables. The problem may be interpreted as finding the distribution of $(\sum_{i=0}^{255} y_i z_i) | y_0 + y_1 x + \dots + y_{255} x^{255} \in I^t$. For $n \leq 256$ and t -binary polynomial α we define $f(n, \alpha) \in \mathcal{R}$ such that for $k \in \mathbb{Z}$, $f(n, \alpha)(k) = P(\sum_{i=0}^{n-1} y_i z_i = k \cap (\sum_{i=0}^{n-1} y_i x^i) - \alpha \in I^t)$. Then we may conclude that

$$f(2n, \alpha) = \sum_{\beta_1 + \beta_2 x^n - \alpha \in I^t} f(n, \beta_1) f(n, \beta_2)$$

Our goal is to find $\frac{1}{P(\sum_{i=0}^{255} y_i x^i \in I^t)} \cdot f(256, 0)$, where $P(\sum_{i=0}^{255} y_i x^i \in I^t) = 2^{-t}$ by similar approach of the proof of Lemma 4.12. Since the values of $f(1, \alpha)$ are easy to determine for all α and there are 2^t t -binary polynomials, by the formula above it takes $7 \cdot 4^t + 2^t$ convolutions to find $f(256, 0)$. This is still computationally infeasible for large t . However, we will see that the results for small t 's are enough for a good estimation.

For $n \geq t$ and being a power-of-2, we have $x^n - 1 \in I^t$ since $x^n - 1 \equiv (x - 1)^n \pmod{2}$ and $I^t = ((x - 1)^t, 2)$. Therefore, for such n we have

$$f(2n, \alpha) = \sum_{\beta_1 + \beta_2 - \alpha \in I^t} f(n, \beta_1) f(n, \beta_2)$$

Since β_1, β_2, α are all t -binary, the degree of $\beta_1 + \beta_2 - \alpha$ is less than t , hence $\beta_1 + \beta_2 - \alpha \in I^t$ is equivalent to $\beta_1 + \beta_2 \equiv \alpha \pmod{2}$. It turns out that we may compute $\{f(2n, \alpha) | \alpha \text{ is } t\text{-binary}\}$ with $\{f(n, \alpha) | \alpha \text{ is } t\text{-binary}\}$ given by Hadamard transform. We cite this book [3] for refer-

ence and omit the details here.

For t -binary polynomials α and β , we define $\langle \alpha, \beta \rangle = \sum_{i=0}^{t-1} \delta_i(\alpha) \delta_i(\beta)$. Then we define

$$\hat{f}(n, \alpha) = \sum_{\beta: t\text{-binary}} (-1)^{\langle \alpha, \beta \rangle} f(n, \beta)$$

We know that this is Hadamard transform, thus for $n \geq t$ being power-of-2, we have

$$\hat{f}(2n, \alpha) = \hat{f}(n, \alpha)^2$$

For $t \leq 8$, we may first calculate $f(8, \alpha)$ for each t -binary polynomial α , compute Hadamard transform to get $\hat{f}(8, \alpha)$, and then compute $\hat{f}(256, \alpha) = \hat{f}(8, \alpha)^{32}$ for each α by taking 5 self-convolutions. At last, by property of Hadamard transform we have

$$f(256, 0) = \frac{1}{2^t} \sum_{\alpha: t\text{-binary}} \hat{f}(256, \alpha)$$

On the other hand, we note that the desired distribution is $2^t f(256, 0) = \sum_{\alpha: t\text{-binary}} \hat{f}(256, \alpha)$.

Furthermore, the term $\hat{f}(256, 0)$ is the probability distribution of $\delta_i(\mathbf{e}^T \mathbf{s}') | \mathbf{e} = \mathcal{U}(R_{q/p})$.

For $r \in \mathcal{R}$, we define the norm of r to be $\sum_{n \in \mathbb{Z}} |r(n)|$, denoted by $\|r\|$. It is easy to see that for $a, b \in \mathcal{R}$, $\|a + b\| \leq \|a\| + \|b\|$ and $\|ab\| \leq \|a\| \|b\|$. Surprisingly, by using computer we find that for $t \leq 7$, $\|\hat{f}(8, \alpha)\| < 2^{-7.999979}$ for all nonzero t -binary polynomial α and $\mu = 6, 8, 10$, and thus $\|\hat{f}(256, \alpha)\| < 2^{-255.999}$. We define $\varepsilon = 2^{-255.999}$ in the remaining of this subsection.

In the following we define $r_t \in \mathcal{R}$ to be the distribution of $\delta_{255}(e_1^T s'_1)$ conditioned on $e_1 = \mathcal{U}(I^t)$ for $t < 256$. By the previous paragraph, for $t \leq 7$ we have $r_t - r_0 =$



$\sum_{\alpha \neq 0: t\text{-binary}} \hat{f}(256, \alpha)$, hence $\|r_t - r_0\| \leq \sum_{\alpha \neq 0: t\text{-binary}} \|\hat{f}(256, \alpha)\| \leq (2^t - 1)\epsilon$. We further define $\mathbf{r}_t \in \mathcal{R}$ to be the distribution of $\delta_{255}(\mathbf{e}^T \mathbf{s}')$ conditioned on $\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})$. We have

$$\begin{aligned} \|\mathbf{r}_t - \mathbf{r}_0\| &= \|r_t^l - r_0^l\| \\ &\leq \|r_t - r_0\| \left\| \sum_{i=0}^{l-1} r_t^i r_0^{l-1-i} \right\| \\ &\leq \|r_t - r_0\| \left(\sum_{i=0}^{l-1} \|r_t^i r_0^{l-1-i}\| \right) \\ &= \|r_t - r_0\| \cdot \sum_{i=0}^{l-1} 1 \\ &\leq (2^t - 1)l\epsilon \end{aligned}$$

For $8 \leq t \leq 256$, we note that by Theorem 4.2(iv) we may conclude that for each $\mathbf{e}_0 \in \mathcal{U}((I^t)^{l \times 1})$, we have $P(\mathbf{e} = \mathbf{e}_0 | \mathbf{e} \in \mathcal{U}((I^t)^{l \times 1})) = 2^{-tl} = 2^l P(\mathbf{e} = \mathbf{e}_0 | \mathbf{e} \in \mathcal{U}((I^{t-1})^{l \times 1}))$.

Therefore $\mathbf{r}_t \leq 2^l \mathbf{r}_{t-1}$, where for $a, b \in \mathcal{R}$, $a \leq b$ means that $b(n) - a(n) \geq 0$ for all $n \in \mathbb{Z}$.

Therefore, $\mathbf{r}_t \leq 2^{(t-7)l} \mathbf{r}_7$ for all $t \geq 7$.

For $d \leq 256$ let ϵ_d be the distribution of $\delta_{255}(\mathbf{e}^T \mathbf{s}') | (s_1, \dots, s_l) \subseteq I^d$ multiplying $P((s_1, \dots, s_l) \subseteq I^d)$, which is 2^{-ld} . Let $\mathbf{r} \in \mathcal{R}$ be the distribution of $\delta_{255}(\mathbf{e}^T \mathbf{s}')$. For $d \leq 256$ we have

$$\begin{aligned} \mathbf{r} &= \sum_{t=0}^{d-1} P(\mathbf{e} = \mathcal{U}((I^t)^{l \times 1})) \mathbf{r}_t + \epsilon_d \\ &= \sum_{t=0}^{d-1} (2^{-tl} - 2^{-(t+1)l}) \mathbf{r}_t + \epsilon_d \text{ (by Lemma 4.12)} \\ &\leq \sum_{t=0}^6 (2^{-tl} - 2^{-(t+1)l}) \mathbf{r}_t + \sum_{t=7}^{d-1} (2^{-tl} - 2^{-(t+1)l}) (2^{(t-7)l}) \mathbf{r}_7 + \epsilon_d \\ &= \sum_{t=0}^6 (2^{-tl} - 2^{-(t+1)l}) \mathbf{r}_t + 2^{-7l} \sum_{t=7}^{d-1} (1 - 2^{-l}) \mathbf{r}_7 + \epsilon_d \\ &= (1 - 2^{-7l} + 2^{-7l}(d-7)(1 - 2^{-l})) \mathbf{r}_0 + E_d \end{aligned}$$

where

$$E_d = \sum_{t=1}^6 (2^{-tl} - 2^{-(t+1)l})(\mathbf{r}_t - \mathbf{r}_0) + 2^{-7l}(d-7)(1-2^{-l})(\mathbf{r}_7 - \mathbf{r}_0) + \boldsymbol{\epsilon}_d \in \mathcal{R}$$



We may estimate $\|E_d\|$ in the following way:

$$\begin{aligned} \|E_d\| &\leq \sum_{t=1}^6 (2^{-tl} - 2^{-(t+1)l})\|\mathbf{r}_t - \mathbf{r}_0\| + 2^{-7l}(d-7)(1-2^{-l})\|\mathbf{r}_7 - \mathbf{r}_0\| + \|\boldsymbol{\epsilon}_d\| \\ &\leq \sum_{t=1}^6 \frac{(1-2^{-l})(2^t-1)l\epsilon}{2^{tl}} + 2^{-7l}(d-7)(1-2^{-l}) \cdot 127l\epsilon + 2^{-dl} \\ &\leq \sum_{t=1}^6 \frac{2^t l \epsilon}{2^{tl}} + 2^{-7l}(d-7) \cdot 127l\epsilon + 2^{-dl} \\ &\leq 6l\epsilon + dl\epsilon + 2^{-dl} \end{aligned}$$

Set $d = 101$, since $l \geq 2$ we may see that $\|E_d\| < 2^{-202}$. Thus we define $c_l = (1 - 2^{-7l} + 2^{-7l} \cdot 94(1 - 2^{-l}))$ and our previous works give the following results:

$$\mathbf{r} \leq c_l \mathbf{r}_0 + \mathbf{e}_r, \quad (4.1)$$

where $\|\mathbf{e}_r\| \leq 2^{-202}$.

For $a \in \mathcal{R}$, we define $B_k^+(a) = \sum_{i \geq k} a(i)$ and $B_k^-(a) = \sum_{i \leq k} a(i)$. Let $X = \delta_i(\mathbf{e}^T \mathbf{s})$, $Y = \delta_i(-\mathbf{e}^T \mathbf{s}' + \frac{q}{p} \mathbf{e}_r)$, and let $\mathbf{r}'_t = \mathbf{r}_t \mathbf{a}$ and $\mathbf{r}' = \mathbf{r} \mathbf{a}$, where $\mathbf{a} \in \mathcal{R}$ is the distribution of $\frac{q}{p} \mathbf{e}_r$. The estimated error rates conditioned on $\text{Inv}(\mathbf{s}, \mathbf{s}_1)$ is in fact $B_{k_1}^+(\mathbf{r}_0) + B_{q/4-k_1}^+(\mathbf{r}'_0) + B_{k_2}^-(\mathbf{r}_0) + B_{-q/4-k_2-1}^-(\mathbf{r}'_0)$ for some $k_1, k_2 \in \mathbb{Z}$.

Let k_1, k_2 be the integers such that $\mathcal{P}_{k_1}^+ + \mathcal{P}_{k_2}^-$ possesses its minimal value. It suffices to

prove that

$$(\mathcal{P}'_{k_1} + \mathcal{P}'_{k_2}) \leq 1.01 (\mathcal{P}_{k_1} + \mathcal{P}_{k_2}) + 2^{-200}. \quad (4.2)$$

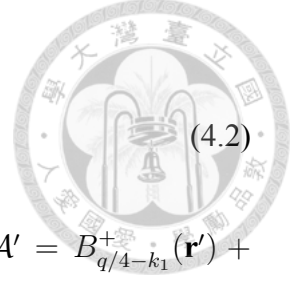
Define $\mathcal{A} = B_{q/4-k_1}^+(\mathbf{r}'_0) + B_{-q/4-k_2-1}^-(\mathbf{r}'_0)$, $\mathcal{B} = B_{k_1}^+(\mathbf{r}_0) + B_{k_2}^-(\mathbf{r}_0)$, $\mathcal{A}' = B_{q/4-k_1}^+(\mathbf{r}') + B_{-q/4-k_2-1}^-(\mathbf{r}')$, and $\mathcal{B}' = B_{k_1}^+(\mathbf{r}) + B_{k_2}^-(\mathbf{r})$. We have

$$\begin{aligned} \mathcal{A}' &= B_{q/4-k_1}^+(\mathbf{r}') + B_{-q/4-k_2-1}^-(\mathbf{r}') \\ &= B_{q/4-k_1}^+(\mathbf{r}\mathbf{a}) + B_{-q/4-k_2-1}^-(\mathbf{r}\mathbf{a}) \\ &\leq B_{q/4-k_1}^+(c_l \mathbf{r}_0 \mathbf{a}) + B_{-q/4-k_2-1}^-(c_l \mathbf{r}_0 \mathbf{a}) + 2^{-201} \|\mathbf{a}\| \text{ (by (4.1))} \\ &= c_l (B_{q/4-k_1}^+(\mathbf{r}'_0) + B_{-q/4-k_2-1}^-(\mathbf{r}'_0)) + 2^{-201} \\ &= c_l \mathcal{A} + 2^{-201} \end{aligned}$$

Use the same argument, we have $\mathcal{B}' \leq c_l \mathcal{B} + 2^{-201}$, and we may take summation to get

$$\mathcal{A}' + \mathcal{B}' \leq c_l (\mathcal{A} + \mathcal{B}) + 2^{-200}. \quad (4.3)$$

For $l \geq 2$ we have $c_l = (1 - 2^{-7l} + 2^{-7l} \cdot 94(1 - 2^{-l})) \leq 1 + 141/32768 < 1.01$, thus we may conclude (4.2) directly from (4.3) and finish our proof.





Chapter 5

NTT-friendly Variance of Saber

5.1 Motivation

As we have seen in the previous chapters, the moduli of Saber are all chosen to be power-of-2. Thus, rather than NTT, other fast multiplication algorithms are adopted to implement the polynomial multiplications in Saber. However, Chung, Hwang, Kannwischer, Seiler, Shih, and Yang found that using NTT can still profoundly improve the speed of the polynomial multiplications of Saber [5]. This inspires us to modify the moduli of Saber to create a variance that has enormous potential for high implementation speed. We will call this "NTT-friendly Saber" throughout this chapter.

5.2 Number theoretic transform (NTT)

This section introduces the theoretical foundations for NTT and defines some terms for later use. We do not deal with technical details of implementation here.

We first introduce the "standard" NTT. Let $n = 2^k$ be a power-of-2 and $p \equiv 1 \pmod{2n}$ be a prime. We know that in \mathbb{Z}_p^\times there exists a primitive $(2n)$ -th root of unity ζ_{2n} . Furthermore, the map φ defined as follows is a ring isomorphism by the Chinese remainder theorem.

$$\varphi : \mathbb{Z}_p[x]/(x^n + 1) \rightarrow \prod_{i=0}^{n-1} \mathbb{Z}_p[x]/(x - \zeta_{2n}^{2i+1})$$

$$\bar{f} \mapsto (\bar{f}, \bar{f}, \dots, \bar{f}) \text{ (} n\text{-tuple)}$$

We note that each coefficient of the n -tuple in the RHS is indeed just an element of \mathbb{Z}_p . Thus the multiplication of elements in the RHS is just the coefficient-wise multiplication, which is very fast. Applying NTT means to find the image of a given $f \in \mathbb{Z}_p[x]/(x^n + 1)$ under φ and applying inverse NTT is to find the preimage of φ . There exist fast algorithms to solve both problems. Hence NTT can be used to accelerate polynomial multiplications in $\mathbb{Z}_p[x]/(x^n + 1)$.

In the following, the range of NTT is called the **NTT domain**. If we say that \hat{f} is in the NTT domain, we mean that the value of \hat{f} is stored as an n -tuple.

Now we see a variance of NTT called incomplete NTT. Assume that $p \equiv n+1 \pmod{2n}$, then a primitive $(2n)$ -th root of unity does not exist in \mathbb{Z}_p^\times . However, we may still find a primitive n -th root of unity ζ_n and define the following isomorphism by the Chinese remainder theorem:

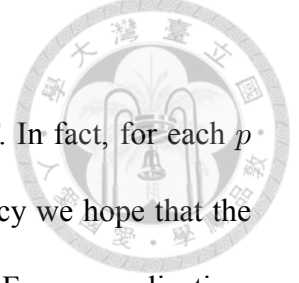
$$\varphi : \mathbb{Z}_p[x]/(x^n + 1) \rightarrow \prod_{i=0}^{n/2-1} \mathbb{Z}_p[x]/(x^2 - \zeta_n^{2i+1})$$

$$\bar{f} \mapsto (\bar{f}, \bar{f}, \dots, \bar{f}) \text{ (} n/2\text{-tuple)}$$

Under this circumstances, computing multiplication by NTT is still fairly efficiently since computing multiplication of $n/2$ pairs of degree-2 polynomials is still much faster than com-

puting a pair of degree- n polynomials by the naive approach.

NTT domain is similarly defined when we use the incomplete NTT. In fact, for each p we may define a corresponding incomplete-NTT. However, for efficiency we hope that the degree of each component of the tuple in the NTT domain is not too high. For our application, $n = 256$ is fixed and we will set $p \equiv 1 \pmod{128}$.



5.3 Notations and parameters

The notations \mathbb{Z}_q , R_q , conventions, and the notations for probability distributions and sampling are the same as listed in section 3.2. The major difference is that q, p are now primes rather than power-of-2, and $q, p \equiv 1 \pmod{128}$.

For $x \in \mathbb{Z}_q$ we define $\text{Round}_{q,p}(x) = \left\lfloor \frac{px}{q} \right\rfloor \in \mathbb{Z}_p$ to be the normal rounding from \mathbb{Z}_q to \mathbb{Z}_p , where $\lfloor x \rfloor = \lfloor x + 0.5 \rfloor$ for $x \in \mathbb{R}$.

Another major difference is that we want to use NTT to enhance the implementation speed as much as possible. Inspired by the implementation of Crystal Kyber [6], rather than performing the transformation and inverse transformation each time when computing the polynomial multiplications, we let some variables remain the transformed form. Furthermore, if a transformation is to do, then it is better to do it in the key generation phase rather than the encryption phase, as encryption speed is usually much more important.

In the remaining part of this chapter, a variable in R_q with a hat on it as shown above means that it is in the NTT domain. For a variable in $R_q^{a \times b}$ with a hat on it, it is a matrix or vector (if

$b = 1$) such that all entries of it are in the NTT domain. For $s \in R_q$, $\text{NTT}(s)$ denotes applying NTT on s . On the contrary, for $\hat{s} \in R_q$ in the NTT domain, $\text{NTT}^{-1}(\hat{s})$ denotes applying inverse NTT on \hat{s} . When this transformation is applied on a matrix or vector of R_q , it means that we transform each entries of the matrix or vector. It is similar for the inverse transform.

The parameter T is abolished to further enhance implementation speed and simplify our analysis. For exchange, the size of the ciphertext becomes larger.

$\text{Inv}(\mathbf{s})$ denotes that for any $\mathbf{y} \in R_q^{l \times 1}$, there exists $\mathbf{A} \in R_q^{l \times l}$ such that $\mathbf{A}\mathbf{s} = \mathbf{y}$. Let $\hat{\mathbf{s}} = \text{NTT}(\mathbf{s})$. As we will see later, $\text{Inv}(\mathbf{s})$ holds if and only if for each component of the NTT domain there exists i such that \hat{s}_i has nonzero value at that component. Therefore, it is very fast to check whether $\text{Inv}(\mathbf{s})$ holds. We will also see why we need this Inv condition.

5.4 Algorithm

We describe the algorithm of public-key encryption of NTT-friendly Saber. We assume that Alice generates private and public keys, and Bob sends the message to Alice following the convention.

5.4.1 Key generation

- (1) Alice picks $\text{seed}_{\mathbf{A}} \leftarrow \{0, 1\}^{256}$ and generates $\hat{\mathbf{A}} = \text{gen}(\text{seed}_{\mathbf{A}}) \in R_q^{l \times l}$.
- (2) Alice picks secret key $\mathbf{s} \leftarrow \beta_{\mu}(R_q^{l \times 1})$ and computes $\hat{\mathbf{s}} = \text{NTT}(\mathbf{s}) \in R_q^{l \times 1}$.
- (3) Check that whether $\text{Inv}(\mathbf{s})$ holds. If not, go back to step (2).
- (4) Compute $\mathbf{b} = \text{Round}_{q,p}(\text{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{s}})) \in R_p^{l \times 1}$ and $\hat{\mathbf{b}} = \text{NTT}(\mathbf{b})$.



- (5) Alice computes $\hat{\mathbf{s}}_p = \text{NTT}(\mathbf{s} \bmod p) \in R_p^{l \times 1}$ and stores the value as private key for later decryption.
- (6) Alice publishes public key $(\text{seed}_{\mathbf{A}}, \hat{\mathbf{b}})$.

5.4.2 Encryption

- (1) Bob encodes his 256-bit message as $m \in R_2$.
- (2) Bob generates $\hat{\mathbf{A}} = \text{gen}(\text{seed}_{\mathbf{A}}) \in R_q^{l \times l}$.
- (3) Bob picks $\mathbf{s}' \leftarrow \beta_{\mu}(R_q^{l \times 1})$ and computes $\hat{\mathbf{s}}' = \text{NTT}(\mathbf{s}') \in R_q^{l \times 1}$.
- (4) Check whether $\text{Inv}(\mathbf{s}')$ holds. If not, go back to step (3).
- (5) Compute $\hat{\mathbf{s}}'_p = \text{NTT}(\mathbf{s}' \bmod p) \in R_p^{l \times 1}$, $\mathbf{b}' = \text{Round}_{q,p}(\text{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{s}}')) \in R_p^{l \times 1}$, and $v' = \text{NTT}^{-1}(\hat{\mathbf{b}}^T \circ \hat{\mathbf{s}}'_p) \in R_p$.
- (6) Bob computes ciphertext $c_m = v' - \frac{p-1}{2}m \in R_p$.
- (7) Bob sends (c_m, \mathbf{b}') to Alice.

5.4.3 Decryption

- (1) Compute $\hat{\mathbf{b}}' = \text{NTT}(\mathbf{b}')$.
- (2) Compute $v = \text{NTT}^{-1}(\hat{\mathbf{b}}'^T \circ \hat{\mathbf{s}}_p) \in R_p$.
- (3) Alice computes $m' = \text{Round}_{p,2}(v - c_m) \in R_2$ as decrypted message.

5.4.4 Parameter sets

First we pick p , which directly affect the size of the public key. The size of public key is determined by $\lceil \log_2 p \rceil$. On the other hand, we hope that the public key space is as large as possible. Therefore, to make the size of public key be the same as the original Saber, we shall pick $p = 769$, which is the largest prime below 1024 that is of the form $128k + 1$. We also consider a slightly larger public key. When $\lceil \log_2 p \rceil = 11$, the optimal choice is $p = 1409$.

We then pick q according to the choose of p . In general, for larger q/p the computation takes longer time. However, if q/p is too small then the security may be compromised. We choose p such that q/p is around 8, which is the ratio of the q to p in the original Saber. Hence, we consider $q=6529, 7297, 7937, 9473, 9857, 10369, 10753$, or 11777 for these two values of p .

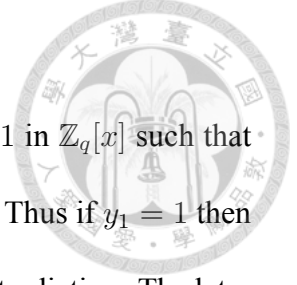
At last, we pick $(\mu, l) = (10, 2), (8, 3)$ like LightSaber and Saber.

5.5 Ciphertext distribution

We first establish a theorem to help us analyze the ciphertext distribution.

Theorem 5.1. Let $\mathbf{s} \in R_q^{l \times 1}$. The followings are equivalent:

- (i) $\text{Inv}(\mathbf{s})$ holds. That is, for any $\mathbf{y} \in R_q^{l \times 1}$ there exists $\mathbf{A} \in R_q^{l \times l}$ such that $\mathbf{A}\mathbf{s} = \mathbf{y}$.
- (ii) For each $f(x) | x^{256} + 1$ in $\mathbb{Z}_q[x]$ and $\deg f(x) \geq 1$, there exists i such that $f(x) \nmid s_i$. Hence, if we let $\hat{\mathbf{s}} = \text{NTT}(\mathbf{s})$, then for each component of the tuple in the NTT domain there exists i such that the value of \hat{s}_i at that component is nonzero.
- (iii) Let $\mathbf{A} = \mathcal{U}(R_q^{l \times l})$. Then $\mathbf{A}\mathbf{s} = \mathcal{U}(R_q^{l \times 1})$.



Proof. (i) \Rightarrow (ii): We prove by contradiction. If there exists $f(x)|x^{256} + 1$ in $\mathbb{Z}_q[x]$ such that $f(x)|s_i$ for all i , then we find that each entry of $\mathbf{A}\mathbf{s}$ is a multiple of $f(x)$. Thus if $y_1 = 1$ then since 1 is not a multiple of $f(x)$ in R_q , it is impossible that $\mathbf{A}\mathbf{s} = \mathbf{y}$, a contradiction. The later part of (ii) is just a straightforward algebraic interpretation.

(ii) \Rightarrow (iii): By Theorem 4.3, it suffices to prove that $(s_1, \dots, s_l) = R_q$. Since q is an odd prime, $x^{256} + 1$ is a separable polynomial in $\mathbb{Z}_q[x]$. Say that $x^{256} + 1 = f_1(x)f_2(x) \dots f_k(x)$ be the polynomial factorization in \mathbb{Z}_q , then by the Chinese remainder theorem we may construct the following isomorphism φ .

$$\varphi : R_q \rightarrow \prod_{i=1}^k \mathbb{Z}_q[x]/(f_i(x))$$

$$\bar{g}(x) \mapsto (\bar{g}(x), \dots, \bar{g}(x))$$

Let $I = (s_1, \dots, s_l) \subseteq R_q$. Since I is an ideal and φ is an isomorphism, $\varphi(I)$ is also an isomorphism. By the property of ideal we know that $\varphi(I)$ is of the form $\prod_{i=1}^k I_i$, where I_i is an ideal of $\mathbb{Z}_q[x]/(f_i(x))$ for each i . Furthermore, since $f_i(x)$ is irreducible for each i , $\mathbb{Z}_q[x]/(f_i(x))$ is a field thus the ideal of it is either the zero ideal or itself.

If $I_i = (0)$ for some i , then we may conclude that $f_i(x)|s_j$ for all $j \leq l$, and since $f_i(x)$ appears in the factorization of $x^{256} + 1$ we also have $f_i(x)|x^{256} + 1$ and $\deg f(x) \geq 1$, which leads to a contradiction. Therefore $I_i = \mathbb{Z}_q[x]/(f_i(x))$ for each i , which implies that $I = R_q$.

(iii) \Rightarrow (i): This is clear.



By Theorem 5.1, we may see that $\mathbf{A}^T \mathbf{s} = \mathcal{U}(R_q^{l \times 1})$ and $\mathbf{A} \mathbf{s}' = \mathcal{U}(R_q^{l \times 1})$. However, \mathbf{b} and \mathbf{b}' are not uniformly distributed since $p \nmid q$ in this scenario and thus the rounding from \mathbb{Z}_q to \mathbb{Z}_p cannot have uniform output. For a certain coefficient of \mathbf{b} , some elements in \mathbb{Z}_p may appear with probability $1/\lfloor q/p \rfloor$ while the others appear with probability $1/(\lfloor q/p \rfloor + 1)$. Thus, the distribution of the ciphertext cannot be uniform. Since the difference is not large, we believe this will not threaten security too much.

5.6 Error rate estimation

We will adopt a similar approach with section 4.3. First we define

$$\begin{cases} \mathbf{e} = q \left(\mathbf{b} - \frac{p}{q} \mathbf{A}^T \mathbf{s} \right) = q\mathbf{b} - p\mathbf{A}^T \mathbf{s} \\ \mathbf{e}' = q \left(\mathbf{b}' - \frac{p}{q} \mathbf{A} \mathbf{s}' \right) = q\mathbf{b}' - p\mathbf{A} \mathbf{s}' \end{cases}$$

It is easy to see that \mathbf{e} has integer coefficients and are in the range $[-\frac{q}{2}, \frac{q}{2})$ by the definition of rounding. As discussed in the previous sections we know that $\mathbf{A}^T \mathbf{s} = \mathcal{U}(R_q^{l \times 1})$ and we may find that $\mathbf{e} \bmod q$ is uniquely determined by $\mathbf{A}^T \mathbf{s}$. Since p, q are both primes here, $p\mathbf{A}^T \mathbf{s} \bmod q = \mathcal{U}(R_q^{l \times 1})$ also and we may conclude that each coefficient of each entry of \mathbf{e} are uniformly distributed in the range $[-\frac{(q-1)}{2}, \frac{q-1}{2}]$. The same holds for \mathbf{e}' .

On the other hand, the decryption is success when $\|v - v'\| < (p-1)/4$. Therefore we will



bound the error rate by $P(\|v - v'\|) \geq (p - 1)/4$. We have

$$\begin{aligned} v - v' &= \mathbf{b}'^T \mathbf{s} - \mathbf{b}^T \mathbf{s}' \\ &= \frac{1}{q} ((\mathbf{e}' + p\mathbf{A}\mathbf{s}')^T \mathbf{s} - (\mathbf{e} + p\mathbf{A}^T \mathbf{s})^T \mathbf{s}') \\ &= \frac{1}{q} (\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}') \end{aligned}$$

Therefore, our goal is to estimate

$$P\left(\|\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}'\| \geq \frac{q(p-1)}{4}\right)$$

For $a \in \mathbb{Z}[x]/(x^{256}+1)$ let $\delta_i(a)$ denote the coefficient of x^i of a . We have $P\left(\|\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}'\| \geq \frac{q(p-1)}{4}\right) \leq 256P(|\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}')| \geq \frac{q(p-1)}{4})$. Furthermore, as discussed before, the distributions of $\delta_i(\mathbf{e}'^T \mathbf{s})$ and $\delta_i(\mathbf{e}^T \mathbf{s}')$ are the same, but they are dependent. Hence we have

$$P(|\delta_i(\mathbf{e}'^T \mathbf{s} - \mathbf{e}^T \mathbf{s}')| \geq \frac{q(p-1)}{4}) \leq 2P(|\delta_i(\mathbf{e}'^T \mathbf{s})| \geq \frac{q(p-1)}{8})$$

To compute the term $P(|\delta_i(\mathbf{e}'^T \mathbf{s})| \geq \frac{q(p-1)}{8})$, we may use convolution as before. However, there are much more terms than before, since we have $\delta_i(\mathbf{e}'^T \mathbf{s}) \leq nl \frac{\mu(q-1)}{4} \sim 2^{23}$. Thus using the naive approach to compute the distribution is not computationally feasible.

To solve this problem, we use the following approaches:

- (1) Use the fast Fourier transform (FFT) to compute the convolutions efficiently.
- (2) There is an issue of precision when we use a program to compute the FFT. Hence we need to use a programming language or package that supports multiple-precision arithmetic. We use C++ with GMP.

(3) To speed up our computation and decrease the memory usage, we compute fewer terms of FFT and use approximation to normal distribution to estimate the remaining terms. By using approximation to normal distribution, we have $P(|X - E(X)| > t) \sim 1 - \text{erf}\left(\frac{t}{\sqrt{2\text{Var}(Y)}}\right)$.

Using these approaches makes it possible to estimate the error rates in about 10 minutes with a personal laptop for each parameter set. The C++ code is attached in Appendix B.

Our results are listed on the next page. The first row of the following tables has different values of p , and the first column of the following tables has different values of q .

- $(\mu, l) = (10, 2)$

$q \backslash p$	769	1409
6529	$2^{-55.44}$	$2^{-105.04}$
7297	$2^{-55.44}$	$2^{-105.21}$
7937	$2^{-55.44}$	$2^{-105.37}$
9473	$2^{-55.45}$	$2^{-105.94}$
9857	$2^{-55.44}$	$2^{-106.10}$
10369	$2^{-55.44}$	$2^{-106.35}$
10753	$2^{-55.44}$	$2^{-106.61}$
11777	$2^{-55.45}$	$2^{-109.45}$



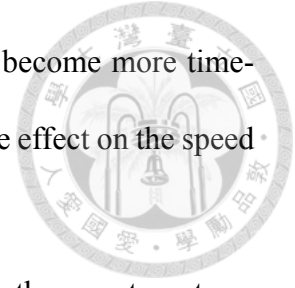
- $(\mu, l) = (8, 3)$

$q \backslash p$	769	1409
6529	$2^{-45.76}$	$2^{-104.49}$
7297	$2^{-45.76}$	$2^{-104.68}$
7937	$2^{-45.76}$	$2^{-104.83}$
9473	$2^{-45.76}$	$2^{-105.39}$
9857	$2^{-45.76}$	$2^{-99.53}$
10369	$2^{-45.76}$	$2^{-99.57}$
10753	$2^{-45.76}$	$2^{-99.61}$
11777	$2^{-45.76}$	$2^{-99.76}$

5.7 Comparison with the original Saber

As shown in [5], applying NTT on the original Saber may accelerate the multiplications. For NTT-friendly Saber, we have an added advantage that we may directly sample the matrix \mathbf{A} in the NTT domain. Therefore, changing the moduli to be prime can further accelerate the multiplications.

On the contrary, some operations like sampling and taking modulo become more time-consuming and less resistant to side-channel attacks. We will focus on the effect on the speed of implementation and discuss the strategy to choose the parameters.



For sampling in \mathbb{Z}_q , we shall use rejection sampling as Kyber [6] and other cryptosystems using prime moduli do. That is, let $N = 2^{\epsilon_n}$ be a power-of-2 larger than q and let $B = \left\lfloor \frac{N}{q} \right\rfloor$. To sample $x \in \mathbb{Z}_q$, we first sample a ϵ_n bits long $s \in \mathbb{Z}_N$. Then if $s < Bq$, we take $s \bmod q$ as our sampling result; otherwise, we sample another $s \in \mathbb{Z}_N$ until the condition $s < Bq$ is satisfied. To reduce the need of taking mod q , we usually let N to be the smallest possible power-of-2 larger than q , and the sampling results become s as $s < q$ is satisfied.

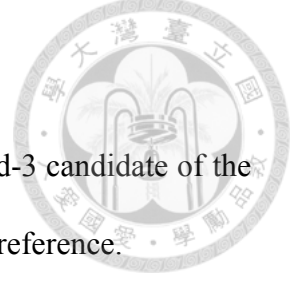
Considering the sampling speed, $q = 7937$ is a good choice since we may take $N = 8192$ and then each sampling is rejected with the probability of only about $1/32$, which is the best among the various moduli q considered in the previous section. It turns out that taking modulo in \mathbb{Z}_{7937} is also supposed to be fast since $7937 = 2^{13} - 2^8 + 1$, since we may compute $x \bmod 7937$ by a few bit-wise operations, additions, and subtractions as the following pseudo code shows:

```

1 mod_7937(x) { // compute x modulo 7937
2   while (x >= 8192) {
3     y := x >> 13;
4     x ← x - (y << 13) + (y << 8) - y;
5   }
6   if (x >= 7937)
7     x = x - 7937;
8   return x;
9 }

```

5.8 Comparison with Kyber



Similar to Saber, Kyber is also a lattices-based cryptosystem and a round-3 candidate of the NIST post-quantum cryptography competition. Readers may see [6] for reference.

Contrary to Saber and our NTT-friendly Saber, Kyber is based on the MLWE problem. Hence, while Kyber needs one more sampling to generate the small errors, our NTT-friendly Saber replaces the sampling with rounding, which is usually faster. Furthermore, if the implementation of sampling is insecure, our NTT-friendly Saber is less likely to be broken than Kyber. However, since we use two different prime moduli for rounding, when we directly follow the algorithm described in Section 5.4, we need to compute the NTT of \mathbf{s}, \mathbf{s}' in both \mathbb{Z}_q and \mathbb{Z}_p . This causes one more computation of NTT in the encryption phase than Kyber, which offsets the advantage of replacing one sampling with one rounding.

If we can compute NTT in \mathbb{Z}_q and \mathbb{Z}_p simultaneously, our NTT-friendly Saber may be preferable to Kyber. In fact, NTT may be computed in \mathbb{Z}_{pq} directly. Hence if computing multiplication and taking modulo of slightly larger numbers does not affect the implementation speed too dramatically, we may use only one NTT and some operations of taking modulo to compute NTT in \mathbb{Z}_q and \mathbb{Z}_p simultaneously.

We also note that it is worth a try to set $q = rp$ rather than a prime, where $r \equiv 1 \pmod{128}$ is a prime. Under this circumstance, the result of NTT in \mathbb{Z}_q may be regarded as the result of NTT in \mathbb{Z}_p directly. However, since $r \geq 257$, this will cause q to be much larger than we have chosen. Detailed analysis and implementation are left for future work.



Chapter 6

Conclusion

In this paper, we first surveyed the distribution of ciphertext and the error rate of Saber. By investigating the algebraic structure of the ring used in Saber, we found some flaws in the original paper of Saber and then figured out some method to remedy them. We proved that the ciphertext distribution is indeed uniform in Saber. However, the estimation of the error rates is questionable, and we proposed an upper bound for the error rates of Saber. Second, we explored the possibility of changing the moduli used in Saber. We presented the algorithm of the NTT-friendly Saber and analyzed the ciphertext distribution and error rate for some parameter sets. We believe that these results help future development of lattice-based cryptosystems.

In future work, it is desirable to develop a tighter and more accurate bound for the error rates of Saber. For NTT-friendly Saber, the computation effectiveness may be tested on various platforms. Discussion on more different parameter sets and a rigorous security proof are also worthwhile to do.



Bibliography

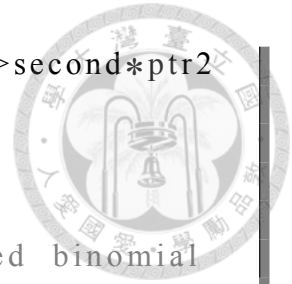
- [1] J.-P. D’Anvers, A. Karmakar, S. Sinha Roy, F. Vercauteren: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure kem. *Progress in Cryptology — AFRICACRYPT 2018*. 282—305 (2018).
- [2] A. Basso, J.M. Bermudo Mera, J.-P. D’Anvers, A. Karmakar, S. Sinha Roy, M. Van Beirendonck, F. Vercauteren: SABER: Mod-LWR based KEM (Round 3 Submission), (2020).
- [3] T.W. Cusick, P. Stanica: Fourier analysis of boolean functions. *Cryptographic Boolean Functions and Applications*. 7—29 (2017).
- [4] Z. Jin, Y. Zhao: Optimal key consensus in presence of noise, <https://arxiv.org/abs/1611.06150>.
- [5] C.-M.M. Chung, V. Hwang, M.J. Kannwischer, G. Seiler, C.-J. Shih, B.-Y. Yang: NTT multiplication for NTT-unfriendly rings. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. 159—188 (2021).
- [6] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J.M. Schanck, P. Schwabe, G. Seiler, D. Stehle: Crystals - Kyber: A CCA-secure module-lattice-based KEM. 2018 IEEE European Symposium on Security and Privacy (EuroS&P). (2018).



Appendix A

Code to Estimate the Error Rates of Saber

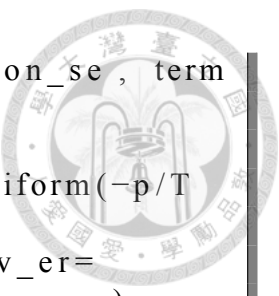
```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int q=8192;
5  const int p=1024;
6  const int n=256;
7  const int e_n=8;
8  const double epsilon=pow(2, -320);
9
10 unordered_map<int, double> add_convolution(unordered_map<int,
    double>& a, unordered_map<int, double>& b){
11     unordered_map<int, double> ans;
12     for(auto ptr1=a.begin(); ptr1!=a.end(); ptr1++)
13         for(auto ptr2=b.begin(); ptr2!=b.end(); ptr2++)
14             ans[ptr1->first + ptr2->first] += ptr1->second * ptr2
                ->second;
15     for(auto ptr=ans.begin(); ptr!=ans.end();){ // 減少計算時間
16         if(ptr->second < epsilon)
17             ptr=ans.erase(ptr);
18         else
19             ptr++;
20     }
21     return ans;
22 }
23
24 unordered_map<int, double> product_convolution(unordered_map<
    int, double>& a, unordered_map<int, double>& b){
25     unordered_map<int, double> ans;
26     for(auto ptr1=a.begin(); ptr1!=a.end(); ptr1++)
27         for(auto ptr2=b.begin(); ptr2!=b.end(); ptr2++)
```



```
28         ans[ptr1->first *ptr2->first]+=ptr1->second*ptr2
           ->second;
29     return ans;
30 }
31
32 unordered_map<int, double> CBD(int mu){ //centered binomial
distribution
33     vector<int> BD(1,1);
34     for(int i=0;i<mu*2;i++){
35         vector<int> new_BD(1,1);
36         for(int j=0;j<i;j++){
37             new_BD.push_back(BD[j]+BD[j+1]);
38         }
39         BD=new_BD;
40         BD.push_back(1);
41     }
42     unordered_map<int, double> ans;
43     int N=1<<(mu*2);
44     for(int i=0;i<=mu*2;i++){
45         ans[i-mu]=(double)BD[i]/N;
46     }
47
48 unordered_map<int, double> uniform(int a, int b){
49     unordered_map<int, double> ans;
50     if(a>b) swap(a, b);
51     double p=1.0/(b-a+1);
52     for(int i=a;i<=b;i++){
53         ans[i]=p;
54     }
55     return ans;
56 }
57 unordered_map<int, double> uniform(int a, int b, int m){ // 若
a<b 則是am, (a+1)m, ..., bm的均匀分布
58     unordered_map<int, double> ans;
59     if(a>b) swap(a, b);
60     double p=1.0/(b-a+1);
61     for(int i=a;i<=b;i++){
62         ans[i*m]=p;
63     }
64     return ans;
65 }
66 double estimate_p_overbound(unordered_map<int, double>& d1,
unordered_map<int, double>& d2, int B, int bound){ // 估計
分布為d1, d2的X,Y, P(X+Y>=B) || P(X+Y<=-B)之bound
67     double ans=0;
```



```
68     int counter=bound;
69     int counter_n=-bound;
70     while(counter>B){
71         if(d1.count(counter)) ans+=d1[counter];
72         counter--;
73     }
74     while(counter_n<=-B){
75         if(d1.count(counter_n)) ans+=d1[counter_n];
76         counter_n++;
77     }
78     while(!d1.count(counter) || d1[counter]<d2[B-counter]){
79         if(d1.count(counter)) ans+=d1[counter];
80         counter--;
81     }
82     while(!d1.count(counter_n) || d1[counter_n]<d2[-B-1-
83         counter_n]){
84         if(d1.count(counter_n)) ans+=d1[counter_n];
85         counter_n++;
86     }
87     for(auto ptr=d2.begin(); ptr!=d2.end(); ptr++){
88         if(ptr->first>=B-counter || ptr->first<=-B-1-
89             counter_n)
90             ans+=ptr->second;
91     }
92     return ans;
93 }
94
95 main(){
96     int N;
97     cout << "請輸入中央二項分布之極值：";
98     cin >> N;
99     unordered_map<int, double> S=CBD(N);
100    cout << "請輸入SABER之矩陣大小：";
101    int L;
102    cin >> L;
103    cout << "請輸入epsilon_T值：";
104    int epsilon_T;
105    cin >> epsilon_T;
106    int T=1<<epsilon_T;
107    unordered_map<int, double> E=uniform(-3, 4);
108    unordered_map<int, double> term=product_convolution(S,E);
109    for(int i=0;i<e_n;i++){
110        term=add_convolution(term, term);
111    }
112    unordered_map<int, double> distribution_se=term;
113    for(int i=1;i<L;i++){
```



```

111         distribution_se=add_convolution(distribution_se , term
112         );
113     }
114     unordered_map<int , double> distribution_er=uniform(-p/T
115         /2 , p/T/2-1 , 8);
116     unordered_map<int , double> distribution_se_cov_er=
117         add_convolution(distribution_se , distribution_er);
118     int bound=N*4*256*L; // distribution_se 中key值絕對值的上界
119     double error_rate=estimate_p_overbound(distribution_se ,
120         distribution_se_cov_er , q/4 , bound);
121     cout << "error□rate=" << error_rate*256 << endl;
122     cout << "log_2□value□of□error□rate=" << log2(error_rate)
123         +8 << endl;
124     return 0;
125 }

```



Appendix B

Code to Estimate the Error Rates of NTT-Friendly Saber

```
1 #include <iostream>
2 #include <stdlib.h>
3 #include <gmp.h>
4 #include <gmpxx.h>
5 #include <complex>
6 #include <math.h>
7
8 using namespace std;
9 const int PREC=200; // 確保誤差在 $2^{(-PREC)}$ 數量級
10 mpf_class Pi;
11 mpf_class prec_unit;
12 complex<mpf_class> zeta;
13 complex<mpf_class>* zeta_major_pow;
14 complex<mpf_class>* zeta_minor_pow;
15 complex<mpf_class> two; // 2+0i
16 int size_FFT;
17 int size_minor_pow;
18
19
20 mpf_class sin(mpf_class theta){
21     mpf_class ans(0, PREC);
22     mpf_class term(theta);
23     int round=0;
24     mpf_class threshold=theta*prec_unit;
25     while(term>threshold){
26         if(round%2==0) ans+=term;
27         else ans-=term;
28         round++;
29         term/=((round*2+1)*(round*2));
30         term*=theta*theta;
```



```
31     }
32     return ans;
33 }
34 mpf_class cos(mpf_class theta){
35     mpf_class ans(0, PREC);
36     mpf_class term(1, PREC);
37     int round=0;
38     mpf_class threshold(prec_unit);
39     while(term>threshold){
40         if(round%2==0) ans+=term;
41         else ans-=term;
42         round++;
43         term/=((round*2-1)*(round*2));
44         term*=theta*theta;
45     }
46     return ans;
47 }
48 void initialize(){
49     mpf_set_default_prec(PREC);
50     Pi=mpf_class("3.141592653589793238462643383279502
51 □□□□8841971693993751058209749445923078164062862089986
52 □□□□280348253421170679", PREC);
53     prec_unit=mpf_class("1", PREC);
54     for(int i=0;i<PREC;i++) prec_unit/=2;
55     {
56         mpf_class temp1("2");
57         mpf_class temp2("0");
58         two=complex<mpf_class>{temp1, temp2};
59     }
60 }
61 complex<mpf_class> zeta_power(int n){
62     return zeta_major_pow[n/size_minor_pow]*zeta_minor_pow[n%
        size_minor_pow];
63 }
64 complex<mpf_class> complex_power(complex<mpf_class> base, int
    pow){
65     if(pow==1) return base;
66     complex<mpf_class> temp=complex_power(base, pow/2);
67     if(pow%2==0) return temp*temp;
68     return temp*temp*base;
69 }
70
71 void power_by_FFT_recur(complex<mpf_class>* arr, int length,
    int hint, int power){
72     int half=length/2;
```



```
73 complex<mpf_class> trans_coeffi=zeta_power(hint/2);
74 for(int i=0;i<half;i++){
75     complex<mpf_class> temp=arr[i]+trans_coeffi*arr[i+
76         half];
77     arr[i+half]=arr[i]-trans_coeffi*arr[i+half];
78     arr[i]=temp;
79 }
80 if(half>1){
81     power_by_FFT_recur(arr, half, hint/2, power);
82     power_by_FFT_recur(arr+half, half, (hint+size_FFT)/2,
83         power);
84 }
85 else {
86     arr[0]=complex_power(arr[0], power);
87     arr[1]=complex_power(arr[1], power);
88 }
89 for(int i=0;i<half;i++){
90     arr[i]=(arr[i]+arr[i+half])/two;
91     arr[i+half]=(arr[i]-arr[i+half])/trans_coeffi;
92 }
93 }
94 void power_by_FFT(complex<mpf_class>* arr, int N, int power){
95     cout << "call power_by_FFT" << endl;
96     mpf_class zeta_real=cos(Pi*2/N);
97     mpf_class zeta_imag=sin(Pi*2/N);
98     zeta=complex<mpf_class>{zeta_real, zeta_imag};
99     size_FFT=N;
100     size_minor_pow=1;
101     while(N>1){
102         N>>=2;
103         size_minor_pow<=1;
104     }
105     zeta_minor_pow=new complex<mpf_class>[size_minor_pow];
106     zeta_major_pow=new complex<mpf_class>[size_FFT/
107         size_minor_pow];
108     {
109         mpf_class temp1("1");
110         mpf_class temp2("0");
111         zeta_minor_pow[0]=complex<mpf_class>{temp1, temp2};
112         zeta_major_pow[0]=complex<mpf_class>{temp1, temp2};
113         mpf_class temp3=cos(Pi*2/(size_FFT/size_minor_pow));
114         mpf_class temp4=sin(Pi*2/(size_FFT/size_minor_pow));
115         zeta_major_pow[1]=complex<mpf_class>{temp3, temp4};
116     }
```



```
115     zeta_minor_pow[1]=zeta;
116     for(int i=2;i<size_minor_pow;i++)
117         zeta_minor_pow[i]=zeta_minor_pow[i-1]*zeta;
118     for(int i=2;i<size_FFT/size_minor_pow;i++)
119         zeta_major_pow[i]=zeta_major_pow[i-1]*zeta_major_pow
120             [1];
121     // 以上為初始設定
122     power_by_FFT_recur(arr, size_FFT, 0, power);
123 }
124 void power_by_FFT(mpf_class* arr, int N, int power){
125     complex<mpf_class>* complex_arr=new complex<mpf_class>[N
126         ];
127     mpf_class zero(0);
128     for(int i=0;i<N;i++)
129         complex_arr[i]=complex<mpf_class>{arr[i], zero};
130     power_by_FFT(complex_arr, N, power);
131     for(int i=0;i<N;i++)
132         arr[i]=complex_arr[i].real();
133 }
134 int main() {
135     initialize();
136     //
137     const int DEGREE_POLY=256;
138     int p,mu,l;
139     cout << "請輸入使用之質數p:";
140     cin >> p;
141     cout << "請輸入mu(二倍極值):";
142     cin >> mu;
143     cout << "請輸入矩陣大小l:";
144     cin >> l;
145     int candidate_q[2]={769, 1409};
146     int N=1;
147     int Bound=(p-1)/2*mu*DEGREE_POLY*l/7;
148     while(N<Bound) N<<=1;
149     cout << "N=" << N << endl;
150     mpf_class* arr=new mpf_class[N];
151     // start: 初始化 arr
152     for(int i=0;i<N;i++) arr[i]=mpf_class(0);
153     {
154         int incrementor=1;
155         for(int i=mu/2;i>0;i--){
156             arr[0]+=incrementor*2;
157             for(int j=1;j*2<p;j++){
```




```
158         arr[j*i] += incrementor * 2;
159         arr[N-j*i] += incrementor * 2;
160     }
161     incrementor = incrementor * (mu/2 + i) / (mu/2 - i + 1);
162 }
163 arr[0] += incrementor * p;
164 }
165 int divisor = p * (1 << mu);
166 for (int i = 0; i < N; i++)
167     arr[i] /= divisor;
168 // end: 初始化 arr
169 power_by_FFT(arr, N, DEGREE_POLY * 1);
170 double stv = sqrt((double) mu * (p * p - 1) / 48 * 256 * 1);
171 for (int i = 0; i < 2; i++) {
172     int q = candidate_q[i];
173     int Bound = (p - 1) * q / 8;
174     mpf_class sum(0);
175     for (int j = Bound; j < N - Bound; j++)
176         sum += arr[j];
177     double n_stv = (double) (N - Bound) / stv;
178     sum *= 512;
179     gmp_printf("estimated error rate for q=%d: %.60Ff\n",
180               q, sum);
181     cout << "plus an error of E(" << n_stv << ") " << endl;
182 }
183 return 0;
```