Physical attacks on CCA-Secure Lattice-based KEM SABER

Submitted by

Puja Mondal

ROLL NO. CRS1905

M.TECH IN CRYPTOLOGY & SECURITY

INDIAN STATISTICAL INSTITUTE

KOLKATA

Primary Supervisor

Dr. Bart Preneel

ELECTRICAL ENGINEERING DEPARTMENT

KATHOLIEKE UNIVERSITEIT LEUVEN

BELGIUM

Secondary Supervisor

Dr. Bimal Kumar Roy

APPLIED STATISTICS UNIT

INDIAN STATISTICAL INSTITUTE

KOLKATA

Mentors

Angsuman Karmakar

Jose M. Bermudo Mera

Suparna Kundu

Arther Beckers

KATHOLIEKE UNIVERSITEIT LEUVEN

BELGIUM

15'th June, 2021

Declaration

I hereby declare that the project entitled **"Physical attack on CCA Secure Lattice-based KEM Saber"** submitted in partial fulfillment for the award of the degree of Master of Technology in Cryptology and Security completed under the supervision of Prof. Dr. Ir. Bart Preneel and Prof. Dr. Bimal Kumar Roy, at ISI Kolkata is an authentic work. Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student with date

It is certified that the above statement made by the student is correct to the best of my knowledge.

Signature and designation with date

(Primary supervisor)

(Secondary supervisor)

Abstract

Nowadays the security of most used public-key algorithms are based on the hardness of one of the following problems :

1. The integer factorization,
2. The elliptic-curve discrete logarithm problem.

But these problems can be solved by *Shor's algorithm* [38] and *Proos.Zalka's algorithm* [31] on a powerful quantum computer.

The relief is that yet there is no quantum computer available. But from the continuous improvement of computer science, we can say that the quantum computer is coming within a few decades. Then to secure the communication, we need many cryptographic schemes, which are quantum secure. That is are not attacked by a powerful quantum computer. For this reason, post-quantum cryptographic schemes are needed.

The lattice-based public-key cryptographic schemes Saber[15], Kyber[7], NTRU are secure against attacks from a quantum computer.
These schemes are selected in the 3rd round by NIST in the Post Quantum Cryptography standardization program. The security of Saber is based on the *Module Learning with Rounding (MLWR)* problem [15], which is assumed to be computationally hard problem[2]. Saber.PKE is an IND-CPA secure scheme and can be transformed to a secure against chosen-ciphertext attacks(IND CCA-secure) by applying well known CCA conversions such as the *Fujisaki Okamoto* transform[19] .

Now the remaining important task is to check the security of implementation of the scheme SABER. Because a perfectly secure scheme is broken if not implemented correctly.

For example: RSA is a public-key encryption[17] whose security is based on the hardness of the prime factorization of a large number. We assume that the factorization of a large integer is a hard problem. Till now, there is no efficient factorizing method. But at the RSA Data Security and CRYPTO conferences in 1996, Kocher presented the "Timing Attack" on RSA [17].

To secure a cryptographic scheme, we have to protect it from any possible attacks. Now for the protection, first of all, we have to analyze the algorithm. And if we see that the scheme is mathematically secure, then we have to analyze the implemented scheme and find all such

possible points, where we can inject a fault. Then we have to see that whether the injected fault leak some information about the secret. If some information leaks after injecting a fault in the implementation, then we have to put a countermeasure to prevent this fault attack.

In this project, first, we inject a fault in the decapsulation of the CCA secure scheme SABER. After that, we query the decapsulation oracle with constructed dummy ciphertexts (which may not be valid ciphertext), then using attack models, we recover the whole secret. To recover the secret, we need to query atmost 3072 number of constructed ciphertext to the decapsulation oracle for the parameter set $(n = 256, l = 3, q = 2^{13}, p = 2^{10}, \mu = 8)$ of SABER.

The List of Abbreviation

BKZ Block Korkine-Zolotarev.

CCA Chosen-ciphertext attack.

CPA Chosen-plaintext attack.

CVP Closest vector problem.

DPA Differential power analysis.

SPA Simple power analysis.

ECC Elliptic curve cryptography.

FO Fujisaki-Okamoto.

KEM Key encapsulation mechanism.

LLL Lenstra-Lenstra-Lovász.

LSB Least significant bit.

LWE Learning with errors.

LWR Learning with rounding.

RLWE Ring-learning with errors.

RLWR Ring-learning with rounding.

MLWE Module learning with errors.

MLWR Module learning with rounding.

MSB Most significant bit.

NIST National institute of standards and technology.

PKC Public-key cryptography.

PKE Public-key encryption.

PRNG Pseudo-Random Number Generators.

RSA Rivest–Shamir–Adleman.

DSS Digital Signature Standard

SIS Shortest integer solution.

SVP Shortest vector problem.

PQC Post Quantum CRyptography

The List of Symbols:

$\mathbb{Z}$	The set of integers.
$\mathbb{R}$	The set of real numbers.
$\mathbb{N}$	The set of natural numbers.
$\mathbb{Z}_q$	The ring of integers modulo q.
$\mathbb{Z}_q[x]$	The polynomial ring of integers modulo q.
$\mathbf{R}_q$	The ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$, where x^n+1 is a polynomial.
$\mathbf{A}$	The matrices are represented by bold capital letter
$\mathbf{a}$	The vectors of polynomials are represented by bold small letter.
a	The polynomials are represented by normal small letter.
$a[i]$	The i^{th} coefficient of the polynomial a.
$\|S\|$	Cardinality of the set S.
$\|\|\mathbf{v}\|\|$	Euclidean norm of the vector $\mathbf{v}$.
$\langle \mathbf{s}, \mathbf{v}\rangle$	inner product of two vectors $\mathbf{s}$ and $\mathbf{v}$
$\lfloor r \rfloor$	The largest integer that does not exceed r.
$\lfloor r \rceil$	The rounding of r. *i.e.*, equal to $\lfloor r+\frac{1}{2}\rfloor$.
$\lfloor r_1 \rceil$	Each coefficients of r_1 are rounded for the polynomial r_1.
$x \leftarrow \mathcal{X}$	x is sampled from the distribution $\mathcal{X}$.
$x \leftarrow \mathcal{U}(S)$	x is sampled uniformly from the set S.
β_μ	The central binomial distribution with parameter μ.
$\mathbf{s} \leftarrow \beta_\mu(\mathbf{R}_q^{l\times 1})$	$\mathbf{s} \in \mathbf{R}_q^{l\times 1}$, and each coefficicient of a polynomial are sampled from β_μ.
$r >> x$	r is shifted right x positions.
$r << x$	r is shifted left x positions.
$\lfloor \mathbf{s} \rceil_{q\to p}$	We apply the operation $\lfloor s_i[j] \rceil >> (\epsilon_q - \epsilon_p)$ for all i, j, where $p = 2^{\epsilon_p}$ and $q = 2^{\epsilon_q}$.

Contents

List of Figures

Chapter 1

Introduction

We are entering into the digital world day by day. Most of the time in our life is being occupied by conversation via mobile, e-mail, online transactions. Nowadays in the pandemic situation, most of the meeting is organized at the digital platform. So, it is important to maintain that privacy over the digital system of this world. It should maintain privacy in such a way that only legitimate users can see the message. To complete this job, we need cryptography. A cryptographer creates a scheme to secure the data transaction. An attacker started searching the crack of this scheme and whether she(/he) finds the crack of the scheme, then the attacker tries to break this scheme. In such a way, cryptology continues to evolve. Mainly cryptography has two parts, one is Private-key cryptography or symmetric key cryptography and another part is Public-key cryptography. Public-key cryptography has two keys, one is a private key or a secret key and another one is the public key. We use the public key to encrypt the message and a private key to decrypt the message. In 1976, Diffie and Hellman [13] presents a public-key cryptosystem. The security of this cryptosystem depends on the Discrete logarithm problem. Another public-key cryptographic scheme RSA is proposed by Rivest et al. [37]. The security of RSA depends on the hardness of the problem prime factorization. RSA and Diffie-Hellman are the most used public-key cryptographic schemes. We can't solve the underlying problems of these schemes in classical computers in polynomial time.

In 1981, at the first conference on the physics of computation, held at MIT in May, Famous physicist Richard Feynman gives a talk on quantum computing and he delineated the model for a quantum machine [16]. In October 2019 IBM reveals its biggest 53 qubits quantum computer yet and they also promise to create a 1000-qubit quantum computer by 2023. Most of the currently used Public-key cryptography (PKC) protocols are based on the integer factorization problem and elective curve discrete logarithm problem. The prime factorization and discrete logarithm problems can be solved in polynomial time algorithms Shor [38] and Proos and Zalka [31] algorithms in a larger quantum computer. But these algorithms require a large number of qubits to solve the problem. The research on quantum computation is increasing rapidly. It is expected that a powerful quantum machine is coming very soon. For this reason, we need PKC schemes that will survive in the quantum world.

NIST organized Post Quantum Cryptography standardization program in PQCrypto

2016 [1]. After 3rd round, four KEM schemes were selected. Three of them CRYSTALS-KYBER, NTRU, SABER are latticed-based cryptography. The security of CRYSTALS-KYBER is based on the RLWE [25] and the security of SABER is based on the MLWR [26]. The LWE and RLWE are reduced from the lattice problem SVP and α-SVP problem. Till now there does not exist any polynomial-time algorithm to solve the SVP problem. So we can believe that the public-key cryptography based on these hard problems will survive in presence of quantum computers. So now the remaining part is to make the implementation side-channel resistance for all these schemes. Therefore improving the side-channel analysis is very important.

The first side-channel attack on RSA, DSS was published in 1995 by Kocher [20]. This attack is called Timing Analysis and this requires predicting the timing behavior of the target device. Fault attack is a special type of side-channel attack that was introduced by Boneh et al. [5] on the public key cryptosystem RSA. In 1999 Paul Kocher introduces another efficient side-channel analysis with help of power consumption, which is known as Power Analysis [22]. After that in 2000 Electromagnetic side-channel attack was introduced by Jean-Jacques[32]. Before this attack, the security of the cryptographic scheme was considered only on the hardness problem of the underlying mathematical problem. But after this attack side-channel analysis is considered as a part of the security of cryptographic schemes.

1.1 Motivation

SABER, a lattice-based post-quantum key-encapsulation mechanism, is entered in the final round of NIST's ongoing post-quantum standardization program. So, now analysis of the implementation of SABER is required. In this thesis, we explain three attack models on SABER by EM side-channel analysis and fault attack.

1.2 Our contribution

The work in this thesis is focused on finding the weakness of the implementation of the lattice-based KEM scheme SABER. In each case, we make some assumptions. Then applying the attack method that we describe in chapter 4, we find the secret. We are giving a summary for each attack model as follows

Model 1 This model assumes that i. we can inject a fault in such a way that we can skip one instruction for one coefficient in decryption, which runs in the decapsulation process. ii. We can distinguish particular two decrypted messages $m = 0$ (all bits are zero) and $m = 1$ (LSB is 1 and other bits are 0) by EM side-channel analysis. Then we construct some dummy ciphertexts in such a way that each faulted decrypted message bit depends on a secret coefficient and decryption of that ciphertexts will either 0 or 1 based on the secret coefficients. Then querying the dummy ciphertexts to decapsulation oracle and observing whether $m = 0$ or 1, we find the whole secret key. We briefly describe the attack method in Chapter 4. To recover the secret using this method, we need 3072 number of queries to make to decapsulation oracle for the parameter set $(n = 256, l = 3, q = 2^{13}, p = 2^{10}, \mu = 8)$.

Model 2 This model assumes that we can inject a fault in decapsulation in such a way that for each decapsulation query we can see only 0^{th} coefficient of decrypted message. Like model 1, we construct some dummy ciphertexts in such a way that 0^{th} bit decrypted message bit depends on a secret coefficient. Then querying the dummy ciphertexts to decapsulation oracle and observing whether $m[0] = 0$ or 1. By this method we find a vector $\mathbf{s}'$ such that each coefficient $s'[j][k] = s[j][k]$ or $-s[j][k]$, where $\mathbf{s}$ is secret key. Then from $\mathbf{s}'$, we can find $\mathbf{s}$ efficiently. To recover the secret using this method, we need 3072 number of queries to make to decapsulation oracle. We make the assumption strong by assuming that we can inject a fault in decapsulation in such a way that for each decapsulation query we can see only fixed i^{th} coefficient of the decrypted message but we don't know the value of i. We solve it by a similar approach but this time it requires at most 256 extra operation to find $\mathbf{s}$ from $\mathbf{s}'$.

Model 3 The previous models don't bother about the output of decapsulation oracle. We describe an attack model which uses the result of decapsulation oracle. In this model, we assume that we can skip one instruction in the decryption method, which is running in the decapsulation algorithm. Since the decryption method is required for removing the noise, therefore after injecting this fault, either the decryption will be not able to remove the noise or can compute the actual message. If the decryption method computes the actual message for a ciphertext say c, then decapsulation oracle will return the valid shared key in this case we will call the fault as ineffective fault for the ciphertext c, and otherwise it will return random shared key, then we will call as the effective. For each query with a ciphertext to faulted decapsulation oracle, we get an inequality on the secret by observing that the fault is effective or not. Querying with multiple ciphertexts, we will get a system of linear inequalities. Since the secret key satisfies this system of inequalities, therefore if we solve the inequality the secret will be recovered. But in our thesis, we have done up to generate the system of inequalities on secret key. In the future, we are planning to solve this system of inequalities.

1.3 Thesis Outline

The chapters in this thesis are organized in the bottom-up manner

Chapter 2 In this chapter, we define the lattice and its basis, shortest vectors, etc. After that, we describe the lattice problems and their hardness. We also describe various types of side-channel attacks that are affecting the cryptosystems.

Chapter 3 This chapter describes the CPA-secure and CCA-secure scheme SABER and gives their corresponding parameters security.

Chapter 4 In this chapter, first we describe some previous fault attacks on lattice-based schemes. After that, I briefly describe the decapsulation algorithm of CCA-secure KEM SABER. After that, we make three attack models and describe the attacks very briefly.

Chapter 5 This chapter summarizes the contribution of this thesis and also describes some directions for future research.

Chapter 2

Preliminaries:

In the previous chapter, we have mentioned that the security of any public-key crypto-system depends on the hardness of some underlying computational problems. A problem like prime factorization, Discrete logarithm are computationally hard in classical computers but these problems become easy or solvable in polynomial time with a quantum computer with a sufficiently large number of qubits.

Also, there exist some problems which are assumed to be hard even against quantum computers. The *shortest vector problem* (SVP), *closest vector problem* (CVP) or *short integer solutions* (SIS) are such problems. The security lattice-based cryptography is based on these problems.

Our main target in this thesis is to attack the implementation of lattice-based crypto-system SABER using faults. In this chapter, we discuss the basics of lattice and the underlying hard problems of lattice-based cryptography, types of side-channel attacks, fault attacks.

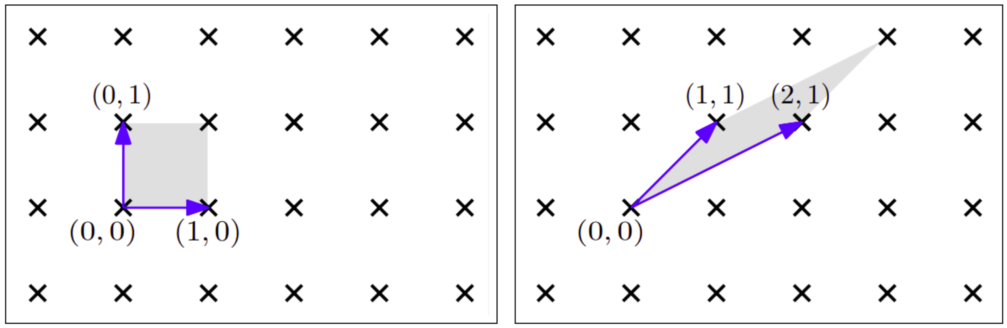
2.1 Lattice

Definition 2.1.1 (Lattice[36]). *Let $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subset \mathbf{R}^m$ be a set of n linearly independent vectors. Then the lattice generated by $\mathcal{B}$ is denoted by $\mathcal{L}(\mathcal{B})$ and defined by the set of all integer linear combination of $\mathcal{B}$ i.e.,*

$$\mathcal{L}(\mathcal{B}) = \left\{ \sum_{i=1}^{m} a_i \alpha_i \mid a_i \in \mathbb{Z} \right\}$$

Here $\mathcal{B}$ is called a basis of the lattice $\mathcal{L}(\mathcal{B})$. The cardinality of a basis is called the rank of the lattice, and let the lattice $\mathcal{L}(\mathcal{B})$ is of dimension m. If $m = n$, then we say that the lattice $\mathcal{L}(\mathcal{B})$ is of full rank. The basis $\mathcal{B}$ can be expressed by the matrix B, whose columns are $\alpha_1, \alpha_2, \ldots, \alpha_n$. Then $\mathcal{L}(\mathcal{B}) = \{B \cdot x \mid x \in \mathbb{Z}^n\}$.

Example The Figure 2.1a contains a full rank Lattice generated by the linearly independent set $\{(0,1),(1,0)\}$. The lattice generated by $\{(0,1),(1,0)\}$ is $\mathbb{Z}^2$. Both rank and



(a) Lattice generated by $\{(0,1),(1,0)\}$ (b) Lattice generated by $\{(1,1),(2,1)\}$

Figure 2.1: different basis of $\mathbb{Z}^2$

dimension of the lattice are 2. Also $\{(1,1),(2,1)\}$ is a basis of this lattice in Figure 2.1b. So basis of a lattice is not unique.

Definition 2.1.2 (Span Of Lattice [36])**.** *The span of the lattice $\mathcal{L}(\mathcal{B})$ is denoted by span($\mathcal{L}(\mathcal{B})$) and defined by:*

$$span(\mathcal{L}(\mathcal{B})) = \{B \cdot y \mid y \in \mathbb{R}^n\}$$

Definition 2.1.3 (Determinant Of Lattice[36])**.** *The determinant of the lattice $\mathcal{L}(\mathcal{B})$ is denoted by det($\mathcal{L}(\mathcal{B})$) and is defined by*

$$det(\mathcal{L}(\mathcal{B})) = \sqrt{B^T B}$$

where B is the matrix corresponding to the basis $\mathcal{B}$ of lattice $\mathcal{L}$.

Definition 2.1.4 (i^{th} successive minimum[36])**.** *Let $\mathcal{L}(\mathcal{B})$ be a lattice of rank n. Then the i^{th} successive minimum is denoted by $\lambda_i(\mathcal{L}(\mathcal{B})$ and is defined by*

$$\lambda_i(\mathcal{L}(\mathcal{B})) = \inf \left\{ r \mid \dim\left(span(\mathcal{L}(\mathcal{B}) \cap \bar{B}(0,r))\right) \geq i \right\}$$

where $\bar{B}(0,r) = \{x \in \mathbb{R}^n \ : \ ||x|| \leq r\}$

We denote the shortest vector of a lattice $\mathcal{L}$ by $\lambda(\mathcal{L})$. In the above lattice 2.1a the length of the shortest vector of lattice $\mathcal{L}$ is $\lambda = 1$ and $\lambda_2(\mathcal{L}) = 2$.

Lattice Problems:

Let $\mathcal{L}$ be a lattice with basis $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. Let $M = \left\{ \sum_{i=1}^{m} x_i\alpha_i : x_i \in \mathbf{R} \right\}$. Then M is a vector space over $\mathbf{R}$ and $|| \cdot ||$, be euclidean norm on M. Given the basis $\mathcal{B}$ of the lattice $\mathcal{L}$, we define the following problems.

2.1.1 Shortest Vecor Problem (SVP)

[36] There are two variants of the SVP.

Definition 2.1.5 (Search SVP Problem)**.** *To find a non zero vector v such that $||v|| \leq ||u||$, for all $u \in \mathcal{L}(\mathcal{B}) - \{0\}$.*
i.e., To find a non zero vector v such that $||v|| = \lambda(\mathcal{L})$.

Definition 2.1.6 (Decisional SVP problem)**.** *Given a rational $r \in \mathbb{Q}$, determine whether $\lambda(\mathcal{L}) \leq r$ or not.*

Till now there are no efficient algorithm to solve the shortest vector problem for a lattice. However from Minkowski's first theorem we can that any lattice $\mathcal{L}$ of rank n contains a nonzero vector of length at most $\sqrt{n}(det(\mathcal{L}))^{\frac{1}{n}}$.

One other variant of SVP is the approximate SVP. In this problem, we are interested in finding an approximation of the shortest vector. The approximation factor is given by some parameter $\alpha \geq 1$. Similar to the SVP problem this has also two variants.

Definition 2.1.7 (Search α-SVP Problem)**.** *Given a real number $\alpha \geq 1$. To find a non zero vector v such that $||v|| \leq \alpha\lambda(\mathcal{L})$.*

Definition 2.1.8 (**Decisional-α-SVP Problem:**)**.** *Given a rational $r \in \mathbb{Q}$, determine whether $\lambda(\mathcal{L}) \leq \alpha r$ or not.*

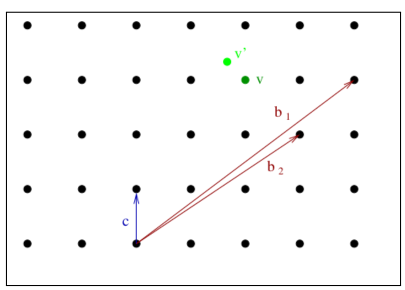


Figure 2.2: A 2-dimensional lattice with basis vectors (b_1, b_2). The shortest vector of this lattice is c. Given a point v', the closest vector in the lattice is v.

2.1.2 Closest Vector Problem (CVP)[36]

Another fundamental lattice problem is closest vector problem or CVP.

Definition 2.1.9 (**closest vector problem (CVP)**)**.** *Given a vector $t \in M$.*
To find: *a vector v such that $||v - t|| \leq ||u - t||$ $\forall$ $u \in \mathcal{L}$. i.e., To find a vector v such that $||v - t|| \leq dist(t, \mathcal{L})$, where $dist(t, \mathcal{L}) = inf\{||v - t|| : v \in \mathcal{L}\}$.*

There is another varient of the CVP problem which is approximate CVP. As before for an approximation factor $\alpha \geq 1$ there are two variants of approximate CVP.

Definition 2.1.10 (**Search** CVP_α **problem).** *Given a vector $t \in M$ and a real number α. To find: a vector v such that $||v - t|| \leq \alpha.dist(t, \mathcal{L})$.*

Definition 2.1.11 (**Decissional** CVP_α **problem).** *Given a vector $t \in M$ and $r \in \mathbb{Q}$, determine whether $dist(t, \mathcal{L}) \leq \alpha r$ or not.*

2.1.3 Relation between the above lattice problems:

We can reduce $\alpha-$CVP from $\alpha-$SVP: Let $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ be a basis of a lattice and we can find closed vector say a_i of α_i for the basis $B^i = \{2\alpha_1, \alpha_2, \cdots, 2\alpha_i, \cdots, \alpha_m\}$. Then $\min \{a_i - \alpha_i \mid i = 1, \cdots, m\}$ is the shortest vector of α-SVP for the basis $\mathcal{B}$. If we choose $\alpha = 1$, then we can say that CVP can reduce from SVP[18].

2.1.4 Algorithm for solving the SVP problem:

Let $\mathcal{B} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a basis of a lattice $\mathcal{L}$. Using $\delta - LLL$ reduction[10] we construct a basis $\{b_1, \ldots, b_n\}$ such that

1. $\forall\ 1 \leq i \leq n$ and $j < i$,$|\mu_{i,j}| \leq \frac{1}{2}$.
2. $\forall\ 1 \leq i < n$ $\delta||b_i'||^2 \leq ||\mu_{i+1,j}b_i' + b_{i+1}'||^2$, where $\{b_1', b_2', \ldots, b_n'\}$ is orthogonal basis reduced from $\{b_1, \ldots, b_n\}$ using Gram-Schmidt orthogonalization[28] and $\mu_{i,j} = \frac{\langle b_i, b_j' \rangle}{\langle b_j', b_j' \rangle}$.

From the 2nd property we get the relation

$$||b_1|| \leq (\frac{2}{\sqrt{4\delta - 1}})^{n-1}\lambda(\mathcal{L})$$

For $\delta = \frac{3}{4}$, we get $||b_1|| \leq 2^{(\frac{n-1}{2})}\lambda(\mathcal{L})$. So given any basis of a lattice we can find a $\frac{3}{4}$-LLL reduced basis, whose b_1 is non zero δ shortest vector.

The algorithm runs in polynomial time but it has an exponential approximation factor. There is an algorithm BKZ[11] which has a small exponential approximation factor but it runs in exponential time.

Hardness of the lattice problems: Till today, there is no such algorithm, which takes polynomial time to solve these problems in quantum computers. So finding a good basis of lattice, a computationally hard problem. For this reason, lattice-based cryptography survives in the post-quantum world.

2.1.5 Learning with Error (LWE) Problem and it's varients:

There are two types of *Learning with error* (LWE) [35] problem . One is *search LWE problem* and another one is *Decisional LWE problem.* The problems are stated below.

Let ℓ, k, n be positive integers and χ be a distribution over $\mathbb{Z}$.

Instance: $\big(\mathbf{A} \in \mathbb{Z}_q^{\ell\times k}, \mathbf{b} = \mathbf{As} + \mathbf{e} \in \mathbb{Z}_q^{\ell\times 1}\big)$, where $A \leftarrow \mathcal{U}(\mathbb{Z}_q^{\ell\times k})$ and $e \leftarrow \chi^{\ell\times 1}$ and $s \leftarrow \chi^{k\times 1}$. ($\mathcal{U}$ denotes the uniform distribution).

Now the *Search LWE Problem* for the parameter (ℓ, k, n, χ) is to find the secret $\mathbf{s}$. whereas the *Decisional LWE Problem* for the parameter (ℓ, k, n, χ) is to distinguish the given pair $(\mathbf{A}, \mathbf{b})$ from a pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{U}(\mathbb{Z}^{\ell\times\mathbf{k}} \times \mathbb{Z}^{\ell\times\mathbf{1}})$

If we use the polynomial ring $\mathbf{R}_q = \mathbb{Z}_q[X]/\Phi(X)$, (where $\Phi(X)$ is *irreducible polynomial*) instead of $\mathbb{Z}_q$ and $\ell = k = 1$, then we call the problem as *Ring learning with error problem (RLWE)*[27] and when $\gcd(\ell, k) > 1$, then we call the problem as *Module Learning with error problem (MLWE)*[8].

The hardness of these LWE problems are based on the computational hardness of lattice problems α-SVP and shortest integer vector problem[35]. The security of NIST's ongoing Post-Quantum Cryptography candidate KYBER is based on the M-LWE problem.

2.1.6 Learning with rounding (LWR) and its variants:

If we scale down the polynomial from $\mathbf{R}_q$ to $\mathbf{R}_p$, where $p < q$ then the RLWE instance becomes $(a \in \mathbf{R}_q, b = \left\lfloor \frac{p}{q} as \right\rceil \in \mathbf{R}_p)$. This instance is called *Ring learnig with Rounding* (RLWR) [4] and the instance $(\mathbf{A} \in \mathbf{R}_q^{l\times k}, \mathbf{b} = \left\lfloor \frac{p}{q} \mathbf{As} \right\rceil \in \mathbf{R}_p^{l\times 1})$ is called *Module Learning with Rounding* Problem [35].

The hardness of LWR depends on the hardness of the lattice SVP problem. SABER is one of the NIST's post-quantum cryptography standardization finalists and the security of SABER is based on the MLWR problem. Till now in a classical and quantum computer, solving the MLWR problem is known hard problem.

2.2 Side Channel Attacks:

The security of a cryptographic scheme can be categorized into two parts.

1. The security of a cryptographic scheme is always based on a computationally hard problem.
2. The security also depends on the implementation of the scheme. That means, if there is a flaw in the implementation, then an attacker might recover some secret data and break the scheme.

A side-channel attack is an attack that gathers the information from a weak implementation of a scheme by affecting the system hardware and find the secret information. We describe many types of Side-channel attacks here.

2.2.1 Electromagnetic Attack

In cryptography, an EM attack is a side-channel attack. By measuring electromagnetic radiation ejected from the device, an attacker can find information without defecting the device.[23]. In the paper [34], they propose a practical EM-side channel attack on Lattice-based post-quantum KEMS.

2.2.2 Fault Attacks

By putting an electronic device in an abnormal condition, we force the device to stop working correctly. Now if a crypto-system is running on the damaged device, then sometimes it leaks the information of secret key. [24]

Type Of Faults

Permanent : [24] This fault damage the cryptographic device permanently. i.e., in the future, the device always works incorrectly. Example: freezing a memory cell to a constant value, cutting a data bus wire.

Transient : [24] This fault disturbs the device only when a particular algorithm is running. Example: abnormally high or low clock frequency, an abnormal voltage in power supply.

Error location : [24] This kind of fault attack only requires imposing an error in a very specific location in the memory cell.

Time of occurrence : [24] This kind of fault damages the device at a specific time of computation.

Error type : [24] We consider many types of errors. For example:

1. We introduce flips in memory, but only in one direction.
2. disables instruction decoder.
3. flip the value of some bit or some byte,

Fault attack is a real and practical threat to any cryptographic scheme. In a fault attack, there are two steps,

1. The way of injecting fault in the cryptographic device.
2. Assuming the fault model, break the cryptosystem.

In our thesis, we have assumed a fault model and after the fault injection, we have retrieved the secret. In the next subsection, we discuss the processes of fault injection.

2.2.3 Fault injection techniques

Practical fault in a device are introduced by putting the device in a abnormal condition. Many process are abvailable to the attacker to make that condition[24]. For example:

- High or low voltage may effect a device's behavior
- There may be occure a error by changing with high or low clock frequency
- Having the device process in extreme temperature conditions is also a potential way to induce faults.

2.2.4 Example of Fault Attack

Attack on RSA with CRT: RSA [17] is a one of the public key cryptosystem that we used for security. To improve the performence, RSA uses Chinese Remainder Theorem for signature scheme. Let $n = pq$, where p and q are prime numbers and d and e are secret and public key described in [17]. Then the signature scheme is given in Algorithm 1.

Algorithm 1: RSA Signature

Data: Given a message m and $n = pq$
Result: The signature s
$m_p = m \mod p$;
$m_q = m \mod q$;
$d_p = d \mod (p-1)$;
$d_q = d \mod (q-1)$;
$x_p = m_p^{d_p} \mod p$;
$x_q = m_q^{d_q} \mod q$;
$s = q(q^{-1} \mod p)x_p + p(p^{-1} \mod q)x_q \mod n$;
return s;

Let inject a fault in the above scheme1 in such a way that x_p is compute incorrectly with high probability. Let the faulted value of x_p is x'_p and after faulted signature is s'. Then $s'^e \neq m \mod p$ but $s'^e = m \mod q$. So $s'^e - m$ is divisible by q but not p. So $gcd(s'^e - m, n)$ is a factor q of n. So from this fractorization, the attacker can compute the secret exponent d. This is a straight forward attack.

2.2.5 Timing Attack

If the running time of a program is not constant (i.e., the running time differs for distinct inputs), then it may leak the information about the secret. i.e., depending on the running time of the program for different input values, an attacker can guess the secret or get more information about the secret. This kind of attack is called a timing attack.[24] This attack was first introduced by Kocher[21].

Principle of this attack: first an attcker run the program with different type of message and note down the run time corresponding to each message. Then try to find the secret from the time set.

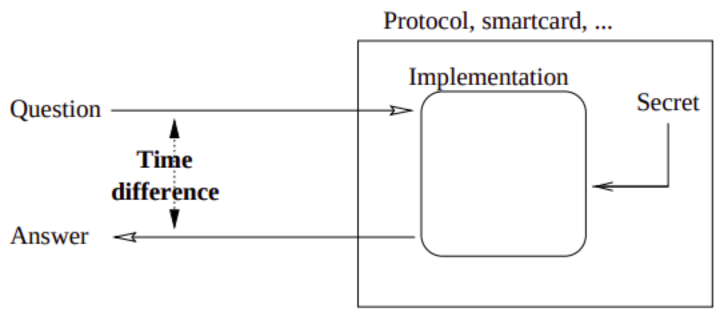


Figure 2.3: Principle of timing attack

2.3 Conclusion

In this chapter, we describe the lattice problem, because the security of our target scheme SABER is depending on the hardness of solving the SVP problem and we describe some side-channel attacks, fault attacks. In the next chapter, we describe the scheme SABER and after this, we will describe how fault attacks and EM-side channel attacks break the security of our scheme SABER.

Chapter 3

Description of SABER

SABER [14] is an IND-CCA2 secure Key Encapsulation Mechanism (KEM) whose security relies on the hardness of the Module Learning With Rounding problem (MLWR). This is secure against quantum computers. SABER is one of the finalists of the NIST Post-Quantum Cryptography Standardization effort.

As the stated introduction, the object of this thesis is to analyze the implementation of the scheme carefully and find the weakness. As we have selected SABER as our target so we need to understand the basics of SABER. In this chapter, We have described the scheme SABER and maintain the security based on the parameters.

3.1 Saber.PKE

3.1.1 Construction

Saber.PKE=($KeyGen, Enc, Dec$) is a public key encryption scheme, it consists of three algorithms which are described below.

Saber.PKE.KeyGen

Algorithm 2: Saber.PKE.KeyGen()

Output: A *public key, secret key* pair (pk, sk)

$seed_{\mathbf{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$;
$\mathbf{A} = gen(seed_{\mathbf{A}}) \in \mathbf{R}_q^{\ell\times\ell}$;
$r \leftarrow \mathcal{U}(\{0,1\}^{256})$;
$\mathbf{s} = \beta_\mu(\mathbf{R}_q^{\ell\times 1}; r)$;
$\mathbf{b} = (\mathbf{A^T s} + \mathbf{h}) \mod q >> (\epsilon_q - \epsilon_p) \in \mathbf{R}^{l\times 1}$;
return $\big(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := (\mathbf{s})\big)$;

In Algorithm 2 the matrix $\mathbf{A} \in \mathbf{R}_q^{\ell\times\ell}$ is sampled by a pseudo-random generator $gen()$.

This generator is initialized with $seed_{\mathbf{A}}$. The secret $\mathbf{s}$ is sampled using central binomial distribution β_μ, whose coefficients in $\mathbf{R}_q$. Here $\mathbf{h} \in \mathbf{R}_q^{\ell\times 1}$ is a constant vector of polynomials where all coefficients of each polynomial is set to $2^{\epsilon_q-\epsilon_p-1}$. Finally it returns pk as public key and sk as secret key.

Saber.PKE.Enc

Algorithm 3: Saber.PKE.Enc()

Input: $pk = (seed_{\mathbf{A}}, \mathbf{b}), m \in \mathbf{R}_2; r$
Output: A ciphertext c

$\mathbf{A} = gen(seed_{\mathbf{A}}) \in \mathbb{R}_q^{\ell\times\ell}$;
if r *is not specified* **then**
  $r \leftarrow \mathcal{U}\big(\{0,1\}^{256}\big)$;
$\mathbf{s}' = \beta_\mu(\mathbf{R}^{\ell\times 1}; r)$;
$\mathbf{b}' = \big((\mathbf{A}\mathbf{s}' + \mathbf{h}) \mod q\big) >> (\epsilon_q - \epsilon_p) \in \mathbf{R}_p^{\ell\times 1}$;
$v' = \mathbf{b}^{\mathbf{T}}(\mathbf{s}' \mod p) \in \mathbf{R}_p$;
$c_m = (v' + h_1 - 2^{\epsilon_p-1}m \mod p) >> (\epsilon_p - \epsilon_T) \in \mathbf{R}_T$;
return $c := \big(\mathbf{b}', c_m\big)$;

In Algorithm 3, a message $m \in \{0,1\}^n$ is represented as an element of $\mathbf{R}_2$. At the time of encryption $\mathbf{s}'$ is sampled from central binomial distribution β_μ with seed r. If r is not specified then it is sampled uniformly. Computation of $\mathbf{b}'$ and c_m are shown in the algorithm. Here h_1 is a polynomial whose all coefficients are set as $2^{\epsilon_q-\epsilon_p-1}$. The algorithm returns $(c_m, \mathbf{b}')$ asthe ciphertext of the message m.

Saber.PKE.Dec

Algorithm 4: Saber.PKE.Dec()

Input: $sk = \mathbf{s}, c = (\mathbf{b}', c_m)$
Output: Decryption m'

$v = \mathbf{b}'^{\mathbf{T}}(s \mod p) \in \mathbf{R}_p$;
$m' = (v - 2^{\epsilon_p-\epsilon_T}c_m + h_2 \; mod \; p) >> (\epsilon_p - 1) \in \mathbf{R}_2$;
return m';

The decryption algorithm or Saber.PKE.Dec is very straightforward as it is shown in Algorithm 4. Here h_2 is a constant polynomial. All coefficients of the polynomial h_2 are set to $2^{\epsilon_p-2} - 2^{\epsilon_p-\epsilon_T-1} + 2^{\epsilon_q-\epsilon_p-1}$. The Saber.PKE.Dec decrypts the ciphertext c.

3.1.2 Parameter set for Saber.PKE

The parameters for Saber are, n where $n-1$ is the degree of the polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$. l is the rank of the module. q, p, T are the The moduli involved in the scheme are chosen to be powers of 2. $q = 2^{\epsilon_q}, p = 2^{\epsilon_p}$ and $T = 2^{\epsilon_T}$ where $\epsilon_q > \epsilon_p > \epsilon_T$. The coefficients of the secret vectors s and s' are sampled according to a centered binomial distribution $\beta_\mu(\mathbf{R}_q^{l\times 1})$ with parameter μ.[14] In Table 3.1 the parameters for Saber.PKE are given.

Name	Security category	ℓ	n	q	p	T	μ
LightSaber-PKE	1	2	256	2^{13}	2^{10}	2^{3}	10
Saber-PKE	3	3	256	2^{13}	2^{10}	2^{4}	8
FireSaber-PKE	5	4	256	2^{13}	2^{10}	2^{6}	6

Table 3.1: Security of Saber.PKE

In Table 3.2 the security[12] of the Saber.PKE is given corresponding to the above parameters.

Security category	Failure probability	Classical Core-SVP	Quantum core-SVP
1	2^{-120}	2^{118}	2^{107}
3	2^{-136}	2^{189}	2^{172}
5	2^{-165}	2^{260}	2^{236}

Table 3.2: Security of Saber.PKE

3.2 Saber.KEM

Saber.PKE is an IND-CPA secure scheme and can be transformed to be secure against chosen-ciphertext attacks (IND-CCA secure)[14] by applying well-known CCA conversions such as the *Fujisaki- Okamoto* [19] transform.

3.2.1 Construction

Saber.KEM=($KeyGen, Encaps, Decaps$) consists of three algorithms which are described below. In the description $\mathcal{F}, \mathcal{G}, \mathcal{H}$ are the hash functions which are implemented using SHA2-256, while $\mathcal{G}$ is implemented using SHA2-512.

Saber.KEM.KeyGen

As we can see in Algorithm 5, first Saber.PKE.keyGen algorithm (Algorithm 2) is used to generate a *public key, secret key* pair (pk, sk). Now the pk is hashed using $\mathcal{F}$ and taken in pkh and a random number z is sampled uniformly from $\{0,1\}^{256}$.

The pair $\big(pk, sk = (z, pkh, pk, \mathbf{s})\big)$ is returned as *publickey, secret key* pair.

Algorithm 5: Saber.KEM.KeyGen()

Output: A *publickey, secret key* pair (pk, sk)

$(seed_{\mathbf{A}}, \mathbf{b}, \mathbf{s}) =$ Saber.PKE.KeyGen();
$pk = (seed_{\mathbf{A}}, \mathbf{b})$;
$pkh = \mathcal{F}(pk)$;
$z \leftarrow \mathcal{U}(\{0,1\}^{256})$;
return $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := \big(z, pkh, pk, \mathbf{s})\big)$;

Saber.KEM.Encaps

As we can see in Algorithm 6, First a random message m is sampled from $\{0,1\}^{256}$. Now this m together with $\mathcal{F}(pk)$ is hashed using the hash function $\mathcal{G}$, this hash value is split into two parts $\bar{K}$ and r. The message m is encrypted using Saber.PKE.Enc (Algorithm 3) with *public key pk* and feeding r as a random seed. The generated ciphertext is now hashed together with $\bar{K}$ using the hash $\mathcal{H}$ i.e., $K = \mathcal{H}(\bar{K}, c)$. The pair (c, K) is returned.

Algorithm 6: Saber.KEM.Encaps()

Input: $pk := (seed_{\mathbf{A}}, \mathbf{b})$
Output: A ciphertext and a hash pair

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$;
$(\bar{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$;
$c =$ Saber.PKE.Enc$(pk, m; r)$;
$K = \mathcal{H}(\bar{K}, c)$;
return (c, K);

Saber.KEM.Decaps

As we see in Algorithm 7, first the ciphertext c is decrypted using the Saber.PKE.Dec (Algorithm 4) with secret key sk. This decrypted message along with pkh which is $\mathcal{F}(pk)$ (see Algorithm 5), is hashed using the hash function $\mathcal{G}$. Similar to the Saber.KEM.Encaps we again split this hash value into two parts i.e., $\bar{K}'$ and r'. Now m' is again encrypted using Saber.PKE.Enc with public key pk and random seed r'. Now this encryption should be similar to the encryption sent before. Therefore, we check if c and this new ciphertext c' are equal or not. If they are equal then we hash $(\hat{K}', c)$ and return it otherwise we return the hash of (z, c).

This returned hash value should be equal to K if everything goes as expected otherwise it will return something else.

Algorithm 7: Saber.KEM.Decaps()

Input: $sk := (z, pkh, pk, s), c$
Output: A hash value

$m' = \text{Saber.PKE.Dec}(sk, c)$;
$(\bar{K}', r') = \mathcal{G}(pkh, m')$;
$c' = \text{Saber.PKE.Enc}(pk, m'; r')$;
if $c = c'$ **then**
  return $K = \mathcal{H}(\bar{K}', c)$;
else
  return $K = \mathcal{H}(z, c)$;

3.2.2 Parameter set for Saber.KEM

Similar to the Saber.PKE, the parameters for Saber.KEM [14] are, n where $n - 1$ is the degree of the polynomial ring $\mathbb{Z}_q[X]/(X^n + 1)$. ℓ is the rank of the module. q, p, T are the The moduli involved in the scheme are chosen to be powers of 2. $q = 2^{\epsilon_q}, p = 2^{\epsilon_p}$ and $T = 2^{\epsilon_T}$ where $\epsilon_q > \epsilon_p > \epsilon_T$. The coefficients of the secret vectors s and s' are sampled according to a centered binomial distribution $\beta_\mu(\mathbf{R}_q^{\ell \times 1})$ with parameter μ. In Table 3.3 the parameters for Saber.KEM are given.

Name	Security category	ℓ	n	q	p	T	μ
LightSaber-PKE	1	2	256	2^{13}	2^{10}	2^3	10
Saber-PKE	3	3	256	2^{13}	2^{10}	2^4	8
FireSaber-PKE	5	4	256	2^{13}	2^{10}	2^6	6

Table 3.3: Security of Saber.KEM

In Table 3.4 the security of the Saber.PKE is given corresponding to the above parameters.

Security category	Failure probability	Classical Core-SVP	Quantum core-SVP
1	2^{-120}	2^{118}	2^{107}
3	2^{-136}	2^{189}	2^{172}
5	2^{-165}	2^{260}	2^{236}

Table 3.4: Security of Saber.KEM

3.3 Conclusion

In this chapter, we only describe the lattice-based pqc Saber.PKE and Saber.KEM mechanism and the corresponding parameter sets and the security table for PKE and for KEM. The correctness proof and security of Saber.PKE is described in the paper[15] and the whole implementation of the scheme is described in [3]. From the security table, we see that the LightSaber-PKE, Saber-PKE, FireSaber-PKE achieve 107, 172, 236 bit security respectively in a quantum computer. So till now, it is almost impossible to break the scheme in a quantum environment. In the next chapter we have seen that despite achieving this security, we can break the security of saber by fault attack. In the next chapter, we will give the overview of only the decapsulation mechanism and describe the attack idea which has been done on the lattice-based pqc Kyber and new hope[30] previously. After this, we describe our attack idea by assuming a fault model.

Chapter 4

Attack Models and Description

In this chapter, we describe some fault attacks on Lattice-based KEMs which done earlier. Among them, we describe the attack models, which can be work for our CCA-secure scheme SABER. Also, we will describe the models that can not work for our scheme SABER and we will give the reason why the model does not work for SABER. After that, we make some assumptions on fault in the target device. Depending on these assumptions, we construct attack models which will work for our scheme SABER.

4.1 Previous Fault Attacks on Lattice-based KEMs

4.1.1 Attack 1

In the paper [33] Ravi et. al showed that by injecting a fault on key Generation and encapsulation, they find the secret key. They use the fact that long secrets are generated by expanding a short seed which is used multiple times but in different domain separater. They inject a fault in the key Generation mechanism such that it generates equal secret $\mathbf{s}$ and error $\mathbf{e}$ by using the same domain separater. Then the public key $\mathbf{b}$ will be

$$\mathbf{b} = \mathbf{a}.\mathbf{s} + \mathbf{e} = \mathbf{a}.\mathbf{s} + \mathbf{s}$$

One can solve the previous equation by Gaussian elimination method and can find the secret $\mathbf{s}$ easily. This practical fault attack works for lattice-based schemes New Hope, Kyber, Frodo, Dilithium. Because these schemes are based on the hardness of the Learning with Errors (LWE) problem.

Now the security of our scheme Saber is based on the hardness of the Module Learning with Rounding (MLWR) problem. In our problem the error $\mathbf{e}$ depends on the secret $\mathbf{s}$. After generating the secret secret $\mathbf{s}$, when the key Generation generates the public key $\mathbf{b}'$, then automatically error $\mathbf{e}$ is made. So in our scheme, probability of getting identical secret $\mathbf{s}$ and error $\mathbf{e}$ is negligible. For this reason, the previous model of fault attack does not work for our scheme Saber.

4.1.2 Attack 2

Valencia et al. [39] have showed that by injecting a fault on decryption of CPA-secure scheme LPR-encryption, it is possible to recover the secret. They have injected fault to the decryption method multiple times with the same secret, which may not occur almost every time in the CPA-secure scheme. However, this model does not work for our CCA-scheme SABER, because in this case, the Fujisaki-Okamoto transform (FO) will detect the fault. Pessl and Prokop [29] have proposed a practical Fault Attack on CCA-secure Lattice KEMs Kyber and New Hope and their masking algorithm. Skipping a single instruction in the decoding process (which is a part of decapsulation), they observe the output shared key. If this fault acually computes the incorrect decrypted message of a valid ciphertext, then the decapsulation algorithm returns a random shared key and the ciphertext (they call it an effective fault). If this fault does not change the decrypted message of a valid ciphertext, then the decapsulation algorithm returns a shared key which depends on the message and the ciphertext (they call it an ineffective fault). If an attacker constructs valid ciphertext by using the encapsulation process, then she(/he) must know the valid shared key (which depends on the message and ciphertext). So by observing the output of decapsulation, the attacker can distinguish the cases, whether the fault is effective or not effective.

Assumptions

i. The attacker has access to encapsulation so, he (/she) can construct lots of (ciphertext, shared key) pairs by the encapsulation mechanism.

ii. The attacker can skip an instruction in the decoding process, which is running in decapsulation.

Structure of Attack [30]

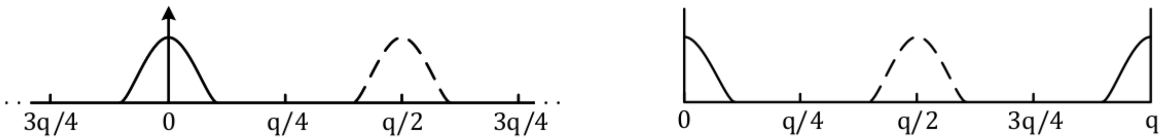
They model their attack idea for LPR encryption and then apply their attack on the LPR-based KEM Kyber and New Hope. The secret is involved in input of decoding process of m' (Algorithm 14).

$$\begin{aligned} m' &= v - us \\ &= br + e_2 + m\lfloor \frac{q}{2} \rfloor - (ar + e_1)s \\ &= asr + er + e_2 + m\lfloor \frac{q}{2} \rfloor - ars - e_1 s \\ &= m\lfloor \frac{q}{2} \rfloor + er + e_2 - e_1 s \end{aligned}$$

Now m' contains some terms with the message bit m, they denote the term $er + e_2 - e_1 s$ by d and call it as encryption noise. i.e., for each coefficient $i = 0, \ldots, 255$

$$d[i] = (e.r)[i] + e_2[i] - (e_1.s)[i]$$

. Each $d[i]$ belongs to the interval $[-\frac{q}{4}, \frac{q}{4}]$, because otherwise decording process will not be able to remove the encryption noise. [9]



(a) Range: $[-q/4, 3q/4)$ (b) Range: $[0, q)$

Figure 4.1: Typical probability distribution of the coefficients of the noise plaintext m 0 . The solid line marks the distribution for a 0 bit, the dashed line for a 1 bit. [29]

As we can see in Figure 4.2, the decoding device of Kyber first multiplies m' by 2, which scales the x-axis in the figure by a factor of 2 and, after that, add $\frac{q}{2}$ with it. After the integer is divided by q, we get a value between 0 and 2, picking the LSB then gives the correct decoded bit.

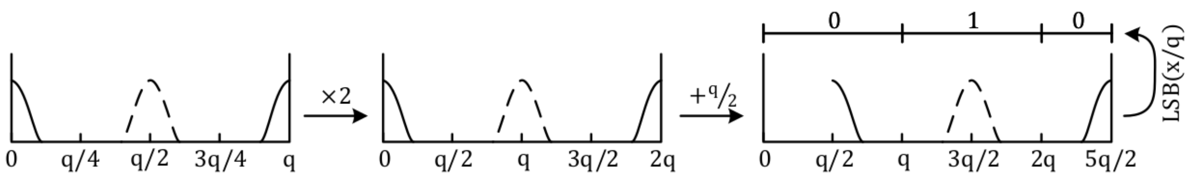


Figure 4.2: Visualization of the decoding routine used in Kyber's reference implementation [29]

For the fault injection, they have skipped the addition by $\frac{q}{2}$ in decoding for one coefficient of m'. The fault injection is showed in Figure 4.3.

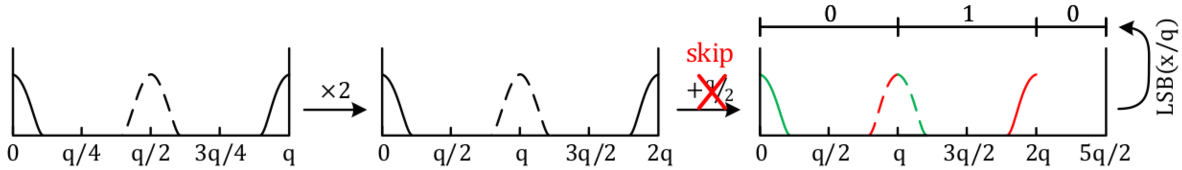


Figure 4.3: Visualization of Kyber's decoding routine if we skip the addition of q/2. Parts of the distribution shown in green are still correctly decoded, despite the fault injection (ineffective fault). Red parts are incorrectly decoded (effective fault)[29]

From Figure 4.3, we say that if the encryption noise $d[i] \geq 0$, then the faulted coefficient remains unchanged. so, in that case, the whole message remains unchanged (since we don't inject fault on decoding process of other coefficients). In this case, the fault will be ineffective. Otherwise, the faulted coefficient will be changed. In this case, the message will change and, so the fault will be effective. So depending upon the fault is effective or not, the attacker gets an inequality $d[i] \geq 0$ or < 0. Since $d[i]$ contain the secret value (since $d = er + e_2 - e_1 s$), so actually the attacker get a linear inequality involving the secrets $(e.r)[i] + e_2[i] - (e_1.s)[i] \geq 0$or < 0 . By querying a large number of times, the attacker gets a system of inequalities. They construct an algorithm to solve this system of inequalities.

We construct a fault attack model like this (Model 3) for our scheme Saber and able to get an inequality involving the secret by distinguishing the effective and ineffective fault. We describe it later.

4.1.3 Attack 3

In the paper by Ravi et al. [34], they have presented a generic and practical EM side-channel assisted chosen-ciphertext attacks applicable to six IND-CCA secure LWE/LWR based PKE/KEMs. These schemes are also round 2 candidates in the ongoing NIST standardization process. They demonstrate very efficient strategies to instantiate the EM side-channel as two particular plaintexts checking oracle, which facilitates their attacks over such unprotected schemes. They demonstrate their attack on the latticed-based post-quantum scheme KYBER and FRODO [6].

Model Of Attack

They construct the dummy ciphertexts c (may not be valid) such that all the coefficient of the decrypted message is zero except the 0^{th} coefficient and the 0^{th} coefficient of the decrypted message depends only on one secret coefficient. To find the secret coefficient $s_0[0]$, they select construct the dummy ciphertext $c = (\mathbf{u}', v)$ where $u'_0[0] = k_u$ is non zero and others coefficients of $\mathbf{u}'$ are zero and $v = k_v$. Then the decrypted message will be

$$m'[j] = \begin{cases} Poly\_to\_Msg(k_v - k_u s_0[0]), \texttt{if} \quad j = 0 \\ Poly\_to\_Msg(-k_u s_0[j]), \ \texttt{if} \ 1 \leq j \leq n-1, \end{cases}$$

where $Poly\_to\_Msg()$ function returns the decrypted message bits of the ciphertext c, after the calculation $v - \mathbf{u}'.\mathbf{s}$, where $\mathbf{s}$ is the secret. They choose the value (k_u, k_v) such that

$$m'[j] = \begin{cases} \mathcal{D}(s_0[0]), \texttt{if} \quad j = 0 \\ 0, \ \texttt{if} \ 1 \leq j \leq n-1 \end{cases}$$

where $\mathcal{D}$ is a function depending on the secret $s_0[0]$. So the decrypted message the value of the decrypted message $m' = 0$ or $m' = 1$ solely depends on $s_0[0]$. They observe the decrypted message $m' = 0$ or $m' = 1$ by EM side-channel analysis. They are able to collect enough ciphertexts such that they can uniquely evaluate the value $s_0[0]$, by observing the decrypted message.

4.2 Preliminaries before our attack

SABER.KEM is a CCA-secure scheme. So the secret $\mathbf{s}$ are non-ephemeral. That means, if we can recover the secret $\mathbf{s}$, we can execute it multiple times. For this reason, our focus is to recover the secret. If we target the Key-Generation and encapsulation mechanisms, then we can't take advantage because these processes are one-time operations. So by injecting fault in these algorithms, we can't recover the secret with very high probability. For this reason, we target the decapsulation method to injecting the fault. The structure of decapsulation is given by in Figure 4.4.

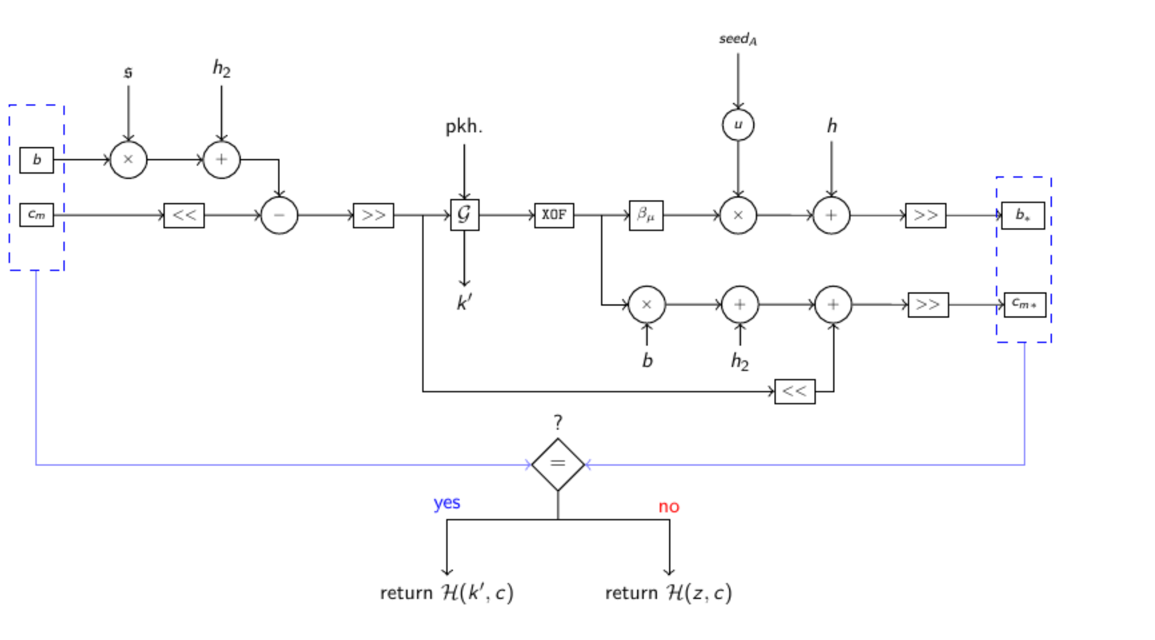


Figure 4.4: Decapsulation of SABER

The input of the decapsulation mechanism is a ciphertext say c. The decapsulation oracle first decrypt the ciphertext $c = \big(b, c_m\big)$, then re-encrypt the decrypted message and compare the ciphertext with the given ciphertext c. If they are equal, then return the hash value which is depending on the message and the given ciphertext c. Otherwise, it will return the hash value which is depending on the given ciphertext c and a random value z.

Throughout this chapter we will describe some attacks on SABER for the parameter set $(l = 3, n = 256, q = 2^{13}, p = 2^{10}, T = 2^4, \mu = 8)$. So for this parameter, the secret $\mathbf{s}$ is a 3×1 vector of polynomials of degree 256, where each coefficient of the polynomials are in $\mathbb{Z}_{2^{13}}$ sampled from the central binomial distribution with parameter $\mu = 8$.

Since the secret coefficients are in $\mathbb{Z}_{2^{13}}$ and sampled from central binomial distribution with parameter $\mu = 8$, so each coefficient belongs to the set $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. The ciphertext c contains two parts. One part is $\mathbf{b}'$ which is a 3×1 vector of polynomials of degree 256, where each coefficient of the polynomials are in $\mathbb{Z}_{2^{10}}$, so the coefficient lie in $\{0, 1, \ldots, 1023\}$. The another part of secret is c_m which is a polynomial of degree 256 and each coefficient of this polynomial are in $\mathbb{Z}_{2^4}$. So each coefficient of c_m are in $\{0, 1, \ldots, 15\}$.

Let us take

$$\mathbf{s} = \begin{bmatrix} s_0[0] + s_0[1] \cdot x + \cdots + s_0[255] \cdot x^{255} \\ s_1[0] + s_1[1] \cdot x + \cdots + s_1[255] \cdot x^{255} \\ s_2[0] + s_2[1] \cdot x + \cdots + s_2[255] \cdot x^{255} \end{bmatrix}$$

be the secret. Where $s_i[j] \in S = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ $\forall i \in \{0, 1, 2\}$ and $\forall j \in \{0, 1, .., 255\}$.

$c = (\mathbf{b}', c_m)$ is ciphertext, where

$$\mathbf{b}' = \begin{bmatrix} b_0[0] + b_0[1] \cdot x + \cdots + b_0[255] \cdot x^{255} \\ b_1[0] + b_1[1] \cdot x + \cdots + b_1[255] \cdot x^{255} \\ b_2[0] + b_2[1] \cdot x + \cdots + b_2[255] \cdot x^{255} \end{bmatrix}$$

where $b_i[j] \in \{0, 1, \ldots, 1023\}$ $\forall\ i \in \{0, 1, 2\}$ and $\forall\ j \in \{0, 1, \ldots, 255\}$ and $\mathbf{c_m} = c_m[0] + c_m[1] \cdot x + \cdots + c_m[255] \cdot x^{255}$, where $c_m[i] \in \{0, 1, \ldots, 15\}$ $\forall\ i \in \{0, 1, \ldots 255\}$.

The inner product of the vectors of polynomials $\mathbf{b}'$ and $\mathbf{s}$ is a polynomial in $\mathbf{R}_p$ and is denoted by $\langle \mathbf{b}', \mathbf{s} \rangle$ and i^{th} coefficient of the polynomial is given by

$$\langle \mathbf{b}', \mathbf{s} \rangle[i] = \sum_{k=0}^{2} \sum_{j=0}^{i} b_k[i - j \cdot] s_k[j] - \sum_{k=0}^{2} \sum_{j=i+1}^{255} b_k[256 + i - j] \cdot s_k[j] \mod p,$$

where $i \in \{0, 1, \ldots, 255\}$

Throughout this chapter when we say the secret $\mathbf{s}$, the ciphertext $\mathbf{b}'$ and c_m we mean that they look like as above. We denote the i^{th} coefficient of message by $m'[i]$. By denoting $m' = 0$, we mean that $m'[i] = 0$ $\forall\ i \in \{0, 1, \ldots, 255\}$. Also by denoting $m' = 1$, we mean that $m'[i] = \begin{cases} 0 & \forall i \neq 0 \\ 1 & \text{for } i = 0 \end{cases}$

4.3 Our Proposed Attack Model 1

In the paper [34] Ravi et al. describe their attack for the latticed-based KEM KYBER and Frodo. To find a secret coefficient say $s_i[j]$, they construct a set of dummy ciphertext such that the decrypted message will be $m' = 0$ (all the bits are zero) and $m' = 1$ (The only first bit is one) depending only secret coefficient $s_i[j]$. And they show that they can distinguish the two messages $m' = 0$ (all the bits are zero) and $m' = 1$ (only the first bit is one) by EM-side channel analysis. We have described this attack previously.

In our attack model, we skip an instruction in the decryption process which, runs in the decapsulation method. Then we construct some dummy ciphertexts such that all the bits of the decrypted message will be zero except the 1st bit and, the 1st bit will be either 0 or 1 depending on only one secret coefficient. i.e., the faulted decrypted message m' will be either 0(all the bits are zero) or 1(Only the first bit is one) for our constructed ciphertext. We assume that we can distinguish two messages $m' = 0$ and $m' = 1$ by EM-Power analysis. We will describe our attack briefly.

4.3.1 Idea of the attack:

We construct a dummy ciphertext pair $c = (\mathbf{b}', c_m)$ (may be not valid ciphertext [1]) such that

1. the decrypted message of c will be: $m'[i] = \text{Saber.PKE.Dec}(c) = 0\ \forall i \in \{0, 1, \ldots, 255\}$. i.e., for this ciphertext the decrypted message does not depends on the secret value.

2. Now we inject a fault in the decryption method, which is a part of decapsulation. After injecting the fault, the decrypted message will be:

$$m'[i] = \begin{cases} 0 & \forall i \neq 0 \\ \mathcal{D}_c(s_k[i]) & \text{for } i = 0 \end{cases}$$

   where $\mathcal{D}_c : S \to \{0, 1\}$ is a function, which computes the 0^{th} element of decrypted message and this function depends on only one secret variable $s_k[i]$.

After injecting the fault, if the decapsulation oracle decrypt the ciphertext c having above two properties, then the decrypted message m' will be either 0 (all coefficient is zero) or 1 (least significant bit is 1 and others are 0).

From the paper of Ravi et al. [34] we see that we can identifying the two cases $m' = 0$ and $m' = 1$ by EM side-channel information. After identifing the decrypted message m', we guess the secret $\mathbf{s}$.

First we will demonstrate our attack simulation to retrieve one coefficient $s_0[0]$. If we choose the ciphertext $c = (\mathbf{b}', c_m)$ (may be invalid) with $b_0[0]$ as non zero and other coefficients of $\mathbf{b}'$ are set to zero and $c_m = a + a \cdot x + a \cdot x^2 + \cdots + a \cdot x^{255}$, where $a \in \mathbb{Z}_{2^4}$. Then from the Algorithm 4, we get the decrypted message as

$$m'[i] = (s_0[i]b_0[0] - 2^{\epsilon_p - \epsilon_T}a + h_2 \mod p) >> 9 \text{ , for all } i \in \{0, 1, \ldots, 255\}$$

So each $m'[i]$ depends on $b_0[0], a$ and $s_0[i]$, where $b_0[0] \in \mathbb{Z}_{2^{10}}$, $s_0[i] \in S = \{-4, -3, \ldots, 3, 4\}$ and $a \in \mathbb{Z}_{2^4}$. We compute $(s.x' - 2^{\epsilon_p - \epsilon_T}y' + h_2 \mod p) >> 9$, for all $x' \in \mathbb{Z}_{2^{10}}$, $s \in S = \{-4, -3, \ldots, 3, 4\}$ and $y' \in \mathbb{Z}_{2^4}$. And by observing the computation value, we find the pairs (x, y) such that $(s.x - 2^{\epsilon_p - \epsilon_T}y + h_2 \mod p) >> 9 = 0\ \forall s \in S$. So if $(b_0[0], a) = (x, y)$, then $(s_0[i].b_0[0] - 2^{\epsilon_p - \epsilon_T}a + h_2 \mod p) >> 9 = 0\ \forall i \in 0, 1, \ldots, 255$ and $s_0[i] \in S$.
Therefore if we choose $(b_0[0], a) = (x, y)$, then $m' = 0$. We run the Algorithm 8 and get a list of (x, y) pairs which we have maintained above.

[1] Running with the ciphertext c the decapsulation oracle may return the random key

Algorithm 8: Algorithm to generate suitable ciphertexts for the attack

```
Result: a list of pairs (x, y) such that (s.x − 2^(ε_p−ε_T) y + h_2 mod p)>>9 = 0 ∀s ∈ S
for x = 0; x < 1023; x++ do
    for y = 0; y < 15; y++ do
        count = 0;
        for s runs on the set S do
            m = s.x − 2^6 y + h_2 mod p>>9 ;
            if m = 0 then
                count = count + 1;
        if count = |S| then
            print (x, y);
```

From the above list of pairs we take the pairs $(b_0[0], a)$ as `(0x1,0x0),(0x11,0xf),(0x10,0x1),(0x16,0xf),(0x16,0x1),(0x21,0x1),(0x21,0xf),(0x3c7,0x0)` to solve our problem. So the ciphertext $c = (\mathbf{b}', c_m)$ with above $(b_0[0], a)$'s satisfies the first criteria of chosen-ciphertext described in 1.

4.3.2 Fault assumption

We inject a fault in such a way that when the decapsulation oracle decrypts the ciphertext c, then it skips the instruction "adding with h_2" for 0^{th} coefficient. For 0^{th} coefficient of the decrypted message, we skip the step in decapsulation as shown in the figure Figure 4.5

After injecting the fault, the decrypted message for the above structured ciphertext $(\mathbf{b}', c_m)$ will be

$$m''[i] = \begin{cases} (s_0[0]b_0[0] - 2^{\epsilon_p - \epsilon_T} a \mod p) >> 9 & \text{, for } i = 0 \\ (s_0[i]b_0[0] - 2^{\epsilon_p - \epsilon_T} a + h_2 \mod p) >> 9 & \text{, for all } i \in \{1, \ldots, 255\} \end{cases}$$

So for a fixed pair $(b_0[0], a)$ which we select from the algorithm8, $m'[i] = 0 \forall i \neq 0$, and $m'[0]$ is depends on the secret coefficient $s_0[0]$. So the constructed ciphertext satisfy the 2nd condition1. For a fixed pair $(b_0[0], a)$ we call the secret coefficient $s_0[0]$ is in class `X`, if $m''[0] = 1$ otherwise we call $s_0[0]$ is in class `O`.

Now we compute the value $(s.x' - 2^{\epsilon_p - \epsilon_T} y' \mod p) >> 9$ $\forall x' \in \mathbb{Z}_{2^{10}}$, $\forall s \in S$ and $\forall y' \in \mathbb{Z}_{2^4}$. By observing the values we find a set
$X = \{(x, y) :$ (x, y) `is one of the member of the list, getting from Algorithm` 8 `and` $(s.x - 2^{\epsilon_p - \epsilon_T} y \mod p) >> 9 = 0$ `for some` $s \in S$ `and` $(s.x - 2^{\epsilon_p - \epsilon_T} y \mod p) >> 9 = 1$ `for some` $s \in S\}$.

i.e., if we choose $(b_0[0], a) = (x, y)$, where $(x, y) \in X$, then $(s_0[i].b_0[0] - 2^{\epsilon_p - \epsilon_T} a + h_2 \mod p) >> 9 = 0$, $\forall s_0[i] \in S, \forall i \in \{1, 2, \ldots, 255\}$ but the value of $(s_0[0].b_0[0] - 2^{\epsilon_p - \epsilon_T} a$

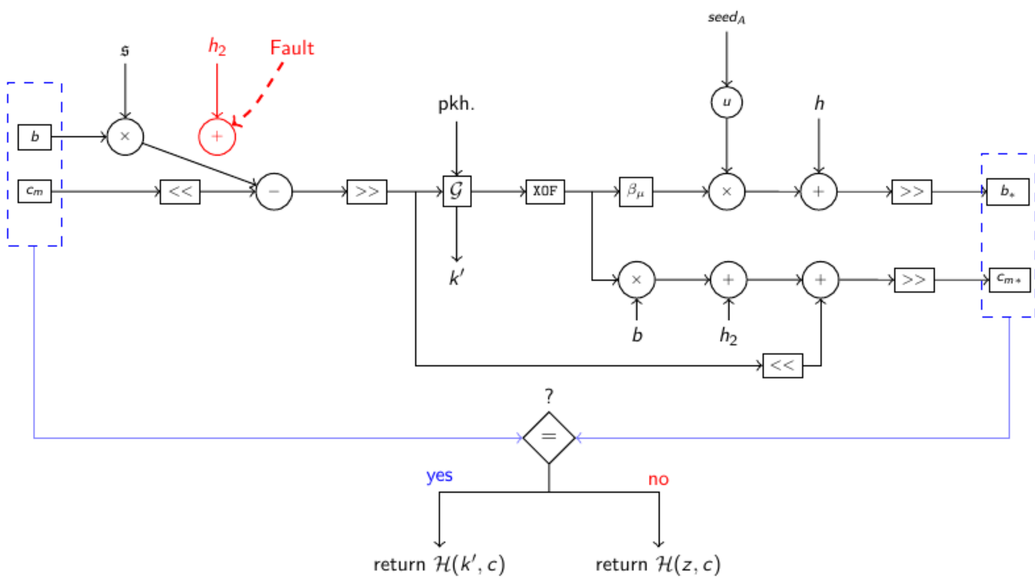


Figure 4.5: Attack model 1

mod p)>>9 $= 0$ depends on the secret coefficient $s_0[0]$.
i.e., if we choose $(b_0[0], a) = (x, y)$, where $(x, y) \in X$, then

$$m''[i] = \begin{cases} \texttt{Depends on } s_0[0] & \text{, for } i = 0 \\ 0 \text{ , for all } i \in \{1, \ldots, 255\} \end{cases}$$

Now running the Algorithm 9 we get the list of touple (x, y, s, m''), such that $(x, y) \in X$ and if we choose $(b_0[0], a) = (x, y)$, then $m''[0] = (s_0[0].b_0[0] - 2^{\epsilon_p - \epsilon_T} a \mod p) >> 9$ for $s_0[0] = s$.

Algorithm 9:

```
Result: a list of pairs (x, y, s, m'') such that
(s.x − 2^{ε_p−ε_T} y + h_2  mod p)>>9 = 0   ∀s ∈ S but
(s.x − 2^{ε_p−ε_T} y  mod p)>>9 = 0  or  1  depending on  s ∈ S

for x = 0; x < 1023; x++ do
    for y = 0; y < 15; y++ do
        count = 0;
        count' = 0 for s runs on the set S do
            m' = s.x − 2^6 y + h_2 mod p>>9        [ε_p − εT = 6];
            m'' = s.x − 2^6 y mod p>>9        //fault step;
            if m' = 0 then
                count = count + 1;
            if m'' = 0 then
                count' = count' + 1;
        if count = |S| then
            for s runs on the set S do
                if count' ≠ 0 and count' ≠ |S| then
                    m'' = s.x − 2^6 y mod p>>9        ;
                    print (x, y, s, m'');
```

Let (x, y, s, m'') be an output of the Algorithm 9. Now we choose $(b_0[0], a) = (x, y)$ and $s_0[0] = s$. Then we say

$$s_0[0] \in \begin{cases} \texttt{class 0} & \text{, if } m'' = 0 \\ \texttt{class X} & \text{, if } m'' = 1 \end{cases}$$

From the outputs of Algorithm 9, we take some outputs to construct the Table 4.1.

(x, y)	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
s	(0x1,0)	(0x11,0xf)	(0x10,0x1)	(0x16,0xf)	(0x16,0x1)	(0x21,0xf)	(0x21,0x1)	(0x3c7,0)
−4	X	X	X	X	X	X	X	0
−3	X	0	X	X	X	X	X	0
−2	X	0	X	0	X	X	X	0
−1	X	0	X	0	X	0	X	0
0	0	0	X	0	X	0	X	0
1	0	0	X	0	X	0	X	0
2	0	0	X	0	X	0	0	X
3	0	0	X	0	0	0	0	X
4	0	0	0	0	0	0	0	X

Table 4.1

The i, j^{th} element of the table is defined by

$$T_{i,j} = \begin{cases} \texttt{O} & \text{, if } (b_0[0], a) = c_j \text{ and } s_0[0](= \text{ the value of } s \text{ in ith row}) \in \texttt{class O} \\ \texttt{X} & \text{, if } (b_0[0], a) = c_j \text{ and } s_0[0](= \text{ the value of } s \text{ in ith row}) \in \texttt{class X} \end{cases}$$

Now if we choose $(b_0[0], a) = c_i$, from the above table, then before injecting fault, m' was zero[2] . After injecting fault,

$$m' = \begin{cases} 0, & \text{if } s_0[0] \in \texttt{class O} \\ 1, & \text{if } s_0[0] \in \texttt{class X} \end{cases}$$

For example, if we choose $(b_0[0], a) = (\texttt{0x1}, \texttt{0x0})$, then after decryption with fault, if we see that $m' = 0$, then $s_0[0] \in \{0, 1, 2, 3, 4\}$. if we see that $m' = 1$, then $s_0[0] \in \{-4, -3, -2, -1\}$.

4.3.3 Method of attack

We inject the fault in the decapsulation process and then query to the decapsulation oracle with ciphertexts (b', c_m).

We choose $c = (b', c_m)$, of the form $\mathbf{b'} = \begin{bmatrix} b_0[0] \\ 0 \\ 0 \end{bmatrix}$, and $\mathbf{c_m} = a + a \cdot x + \cdots + a \cdot x^{255}$

Now we will send the ciphertext $c = c_1, c_2, \ldots, c_8$ one by one (maintaining the order). By observing the decrypted message m' we write the corresponding class (i.e., `class O` or `class X`). This way we will get a ordered sequence of `class O` and `class X` of length eight. This ordered sequence will uniquely represent one row of the table 4.1 because we have constructed the table in that way. The secret coefficient $s_0[0]$ will be the value of s corresponding to that row.

Example: With this method, if we get the ordered sequence $(\texttt{X}, \texttt{X}, \texttt{X}, \texttt{X}, \texttt{X}, \texttt{X}, \texttt{X}, \texttt{O})$. This sequence is the first row of the Table 4.1, so the secret coefficient $s_0[0]$ will be the value of s corresponding to the first row, which is -4. If we get the ordered sequence $(\texttt{O}, \texttt{O}, \texttt{X}, \texttt{O}, \texttt{O}, \texttt{O}, \texttt{O}, \texttt{X})$. This sequence is the eighth row of the Table 4.1, so the secret coefficient $s_0[0]$ will be the value 3.

To find $s_0[0]$, we have to query for eight ciphertexts by using this process. We now describe another technique to decrease the number of queries.

[2] **We can check that all decryption message whether zero or not by using power/EM side channel information.**

Reducing the number of queries

We divide the set of secrets S into two disjoing proper subsets say S_1 and S_2, by following the rule:

1 We query to decapsulation oracle with some ciphertext c with $(b_0[0], a) = c_i$. Then we observe the class of the secret belonging.

If the secret s is in `class O`, then $s \in S_1$

If the secret s is in `class X`, then $s \in S_2$

2 Then we again divide the subsets S_1 and S_2, by applying the above rule 1.

By dividing the set S into smaller subsets with the above rule, we have the following binary tree in Figure 4.6.

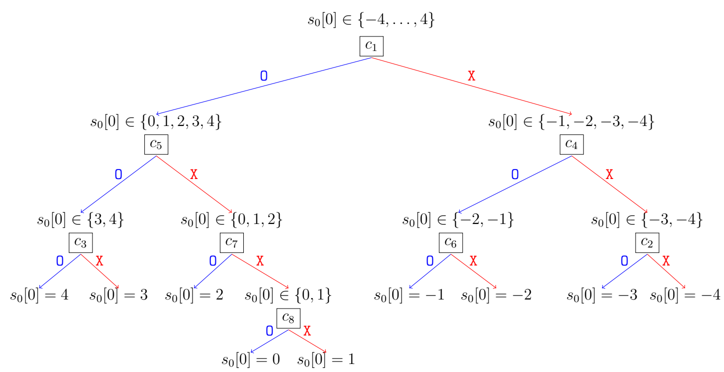


Figure 4.6: Binary tree with each leaf node as the secret for attack model 1

After injecting the fault in device, we query to decapsulation oracle with a constructed ciphertext c_1. Then we can move to left or right down the tree depending on the secret coefficient is in class(O) or class (X). Now we can arrive at any leaf node of the tree in Figure 4.6 starting from the root by exactly one path. The height of the tree is 4. So to find the secret coefficient $s_0[0]$, we have to query at most 4 times.

Example:

Suppose the secret coefficient $s_0[0] = 0$. We will use our technique to find this secret.

1. First we query the ciphertext $(b_0[0], a) = c_1$, from the Table 4.1 we observe that $s_0[0]$ is in `class O`. So according to the Figure 4.6, $s_0[0] \in \{0, 1, 2, 3, 4\}$.

2. Now we query with the ciphertext $(b_0[0], a) = c_5$ and from the Table 4.1 we can observe that the secret $s_0[0]$ is in `class X` . Again from the Figure 4.6, $s_0[0] \in \{0, 1, 2\}$

3. We query with the ciphertext $(b_0[0], a) = c_7$ and again we observe that the secret $s_0[0]$ is in `class X` and that gives us $s_0[0] \in \{0, 1\}$.

4. Finally we query with the ciphertext $(b_0[0], a) = c_8$ and we will observe that the secret $s_0[0]$ is in `class 0` and finally we are arrived at the leaf node and found the secret $s_0[0] = 0$.

4.3.4 To retrieve the full secret s

To find the secret $s_i[j]$, First we have to construct the ciphertext $c = (\mathbf{b}', c_m)$ such that 0^{th} bit of the decrypted message depends on the secret $s_i[j]$, where $i \in \{0, 1, 2\}$ and $j \in \{0, 1, \ldots, 255\}$.

Now if we choose $c = (\mathbf{b}', c_m)$, such that only $b_i[k]$ is non zero, where $i \in \{0, 1, 2\}$ and $k \in \{0, 1, \ldots, 255\}$, others coefficients of $\mathbf{b}'$ are zero and $c_m = a + a \cdot x + \cdots + a \cdot x^{255}$, where $a \in \mathbb{Z}_{2^4}$. Then the decrypted message will be
$\forall t \in \{0, 1, \ldots, 255\}$

$$m'[t] = \begin{cases} (s_i[j] \cdot b_i[k] - 2^{\epsilon_p - \epsilon_T} a + h_2 \mod p) >> 9 \ \texttt{if} \\ \qquad\qquad k = t - j \\ (-s_i[j] \cdot b_i[k] - 2^{\epsilon_p - \epsilon_T} a + h_2 \mod p) >> 9 \ \texttt{if} \\ \qquad\qquad k = 256 + t - j \end{cases}$$

So each $m'[t]$ depends on $b_i[k], a$ and $s_i[j]$., where $k = t - j$ or $k = 256 + t - j$

We run the Algorithm 8 and get a list of (x, y) pairs such that the decrypted message m' will be 0 (all coefficient is zero) for all $s_i[j] \in S$.

Now, after injecting the fault, we will get all the coefficients of the decrypted message m'' to remain unchanged except the 0^{th} coefficient. The 0^{th} coefficient will be changed or remain unchanged depending on the secret coefficient $s_i[j]$. Now after fault injection to the 0^{th} coefficient of at the time of decryption will be:

$$m''[0] = \begin{cases} (s_i[j] \cdot b_i[k] - 2^{\epsilon_p - \epsilon_T} a \mod p) >> 9 \ \texttt{if} \ \ j + k = 0 \\ (-s_i[j] \cdot b_i[k] - 2^{\epsilon_p - \epsilon_T} a \mod p) >> 9 \ \texttt{if} \ \ j + k = 256 \end{cases}$$

So for a fixed pair $(b_i[k], a)$ which we select using the Algorithm 8, we have the decryption $m''[i] = 0 \ \ \forall i \neq 0$, and $m''[0]$ is depends on the secret coefficient $s_i[j]$. So the constructed ciphertext satisfy the second condition 1.

Let

$$s'_i[j, k] = \begin{cases} s_i[j] \ \texttt{if} \ \ j + k = 0 \\ -s_i[i] \ \ \texttt{if} \ \ j + k = 256 \end{cases}$$

Then we can write the 0^{th} coefficient of decrypted message of the constructed ciphertext c , after fault injection as follows:

$$m''[0] = (s_i'[j,k] \cdot b_i[k] - 2^{\epsilon_p - \epsilon_T} a \mod p) >> 9$$

For a fixed pair $(b_i[k], a)$ we call the coefficient $s_i'[j,k]$, is in class `X`, if

$$m''[0] = 1$$

otherwise we call $s_i'[j,k]$ is in class `O`. We choose pair $(b_i[k], a) = (b_0[0], a)$ and processing similar steps of finding the secret $s_0[0]$, we can find $s_i'[j,k]$ $\forall i, j$. Then from there we can derive $s_i[j]$ depending on the value of j and k.

4.3.5 Total number of queries

For each secret $s_i[j]$, we have to query almost 4 times to the decapsulation oracle. Now there are 768 many secret coefficients. So the total number of the query is almost $768 \times 4 = 3072$. So we need to conduct at most 3072 many faults.

In this section, we describe an attack model to recover the secret $\mathbf{s}$. Here we construct the model by assuming the assumptions stated above. From the paper of Pessl and Prokop [29] we know that injecting this fault is practically possible. Also from the paper of Ravi et al. [34] we got that information that by EM-power analysis we can distinguish two messages $m = 0$ and $m = 1$. We are expecting that we can do this attack practically. Now we write a program of this model. In the end, the program returns a vector $\mathbf{s}'$. We check that $\mathbf{s}'$ satisfies the relation $\lfloor \mathbf{A}.\mathbf{s} \rceil_{\mathbf{q} \to \mathbf{p}} = \mathbf{b}$. Algorithm 15 is the pseudo-code to simulate the attack. If we make this fault physically, then our attack model will work.

4.4 Our Proposed Attack Model 2

In this attack model, we are not injecting any fault. In this model, we only construct the ciphertext with a special pattern. Here we do not choose a ciphertext such that the decryption of the ciphertext is zero or one. The decrypted message could be anything. Here we assume that we can see only one particular bit of the decrypted message.

4.4.1 Idea of the attack:

We construct ciphertexts $c = (\mathbf{b}', c_m)$ (maybe not valid ciphertext [3]) such that One bit of decrypted message $m'[i]$ only depends on one coefficient of the secret $\mathbf{s}$. If an attacker is able to see only one decrypted message bit, then by querying these kinds of ciphertext to the decapsulation oracle, the attacker can recover the full secret key.

[3] If we query to the decapsulation oracle with c, then it may return the random shared key

4.4.2 Assumption

We can see only 0^{th} coefficient of the decrypted message m' in decapsulation. We will recover the coefficients of $\mathbf{s}$ one by one.

First we will demonstrate our attack simulation to recover one coefficient $s_0[0]$. If we construct the ciphertext $c = (\mathbf{b}', c_m)$ (may be invalid) with $b_0[0]$ as non zero and other coefficients of $\mathbf{b}'$ are set to zero and $c_m = 0$ (all coefficient of c_m is zero). In this attack model, when we say the ciphertext $c = (\mathbf{b}', c_m)$ with some value of $b_0[0]$, we mean that except $b_0[0]$ the other coefficient of $\mathbf{b}'$ are zero and c_m until we mention other ciphertext construction.
Then the decrypted message of c will be:

$$m'[i] = (s_0[i]b_0[0] + h_2 \mod p)>>9 \text{ , for all } i \in \{0, 1, \ldots, 255\}$$

So each $m'[0]$ depends on $b_0[0]$ and $s_0[0]$. Hence for fixed $b_0[0]$ the 0^{th} bit of the decrypted message of c only depends on the secret $s_0[0]$. We run the following algorithm and get a list of x value such that for fixed $b_0[0] = x$, the decrypted message bit $m'[0]$ will vary when we select different $s_0[0]$ from the set S.

Algorithm 10: Algorithm to generate suitable ciphertext

Result: a list of pairs x such that 0^{th} bit of decrypted message of the construced ciphertext depends on the value of $s_0[0]$

```
for x = 0; x < 1023; x++ do
    count = 0;
    for s runs on the set S do
        m = s.x + h_2 mod p>>9    ;
        if m = 0 then
            count = count + 1;
    if count ≠ |S| and count ≠ 0 then
        print (x);
```

From the list of x's, we take the values `0x8e`, `0x11c`, `0x10a`, `0x5f`, `0xc7`, `0x1a2`, `0x73`, `0x1ba` for our attack simulation. For a fixed constructed ciphertext c with non zero coefficient $b_0[0]$, we call the secret coefficient $s_0[0]$ is $\texttt{classX}_1$ if $m[0] = 1$, otherwise we call the secret coefficient $s_0[0]$ is in $\texttt{classX}_0$.

Now for the constructed ciphertext c with $b_0[0] = x$, where x belongs to the above selected list, we observe the following result, where each row represent the class of secrets for fixed coefficient $b_0[0] = c_i$.
Now if we choose $b_0[0]$ from the above table then,

$$m'[0] = \begin{cases} 0 & \text{if } s_0[0] \text{ is in class } X_0 \\ 1 & \text{if } s_0[0] \text{ is in class } X_1 \end{cases}$$

c_i	x	$s=-4$	$s=-3$	$s=-2$	$s=-1$	$s=0$	$s=1$	$s=2$	$s=3$	$s=4$
c_1	`0x8e`	X_1	X_1	X_1	X_0	X_0	X_0	X_1	X_1	X_1
c_2	`0x11c`	X_0	X_0	X_1	X_1	X_0	X_1	X_1	X_0	X_0
c_3	`0x10a`	X_0	X_0	X_1	X_1	X_0	X_0	X_1	X_0	X_0
c_4	`0x5f`	X_1	X_1	X_0	X_0	X_0	X_0	X_0	X_1	X_1
c_5	`0xc7`	X_0	X_1	X_1	X_0	X_0	X_0	X_1	X_1	X_0
c_6	`0x1a2c`	X_1	X_1	X_0	X_1	X_0	X_1	X_0	X_0	X_1
c_7	`0x73`	X_1	X_1	X_1	X_0	X_0	X_0	X_0	X_1	X_1
c_8	`0x1ba`	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_1

Table 4.2

4.4.3 Method of attack

To find $s_0[0]$ we query to the decapsulation oracle with ciphertexts c. where $c = (b', c_m)$, of the form $\mathbf{b'} = \begin{bmatrix} b_0[0] \\ 0 \\ 0 \end{bmatrix}$, and $\mathbf{c_m} = 0$.

Now we will send the ciphertext c with $b_0[0] = c_1, c_2, \ldots, c_8$ one by one (maintaining the order). Observing the 0^{th} coefficient $m^{'}[0]$ of decryted message m' we write the corresponding class (i.e., class $\mathtt{X}_0$ or class$\mathtt{X}_1$). Doing this way we will get a ordered sequence of class $\mathtt{X}_0$ and class $\mathtt{X}_1$ of length 8. This ordered sequence will be only one column of the table 4.2 because we have constructed the table in this way. The secret coefficient $s_0[0]$ will be the value of s lies on that column.

Example: Suppose we get the ordered sequence $(\mathtt{X}_1, \mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_1, \mathtt{X}_1, \mathtt{X}_1, \mathtt{X}_1, \mathtt{X}_1)$ by doing this method. This sequence is the 2nd secret column of the table4.2, so the secret coefficient $s_0[0]$ will be the value of s of the 2nd secret column, which is -3. Similarly If we get the ordered sequence $(\mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_0, \mathtt{X}_0)$, then the secret coefficient $s_0[0]$ will be the value of s of the 5th column, which is 0.

In this process to find $s_0[0]$, we have to query 8 ciphertexts. We now describe another technique to decrease the number of queries.

Reducing the number of queries

We divide the set of secrets S into two disjoing proper subsets say S_1 and S_2, by following the rule:

We query to decapsulation oracle with some ciphertext c with $b_0[0] = c_i$. Then we observe the class of the secret belonging.

1. If the secret s is in class $\mathtt{X}_0$, then $s \in S_1$
2. If the secret s is in class $\mathtt{X}_1$, then $s \in S_2$

Then we again divide the subsets S_1 and S_2, by applying the above rule.

By in method, we construct the tree:

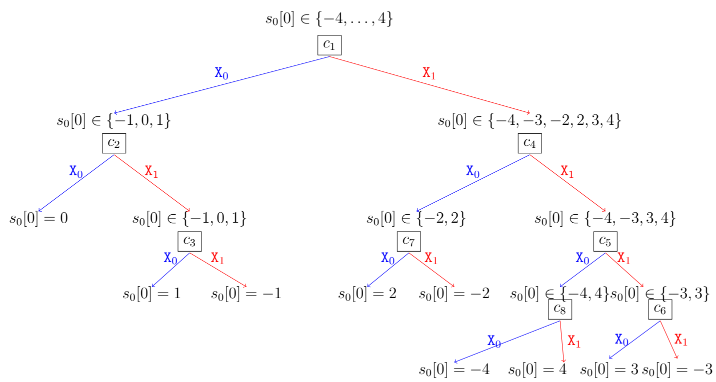


Figure 4.7: Binary tree with each leaf node as the secret for attack model 2

Now we can arrive at any leaf node of the tree4.7 from the root by exactly one path. The height of the tree is 4. So to find the secret coefficient $s_0[0]$ we have to query atmost 4 times.

Example: Suppose the secret coefficient $s_0[0] = 0$.
First we query the ciphertext c with $b_0[0] = c_1$. Then from the table4.2, we observe that $s_0[0]$ is in class$\mathtt{X}_0$. After that we query with the ciphertext c with $b_0[0] = c_2$ and from the table4.2 we will observe that the secret $s_0[0]$ is in class $\mathtt{X}_0$ and finally we arrive at the leaf node and we get the secret $s_0[0] = 0$.

$s_0[0]$ is any one of the leaf nodes of the above tree. From the beginning, we reach a leaf node of the above tree, by using atmost 4 many queries to the decapsulation oracle.
So, to find $s_0[0]$, We have to query atmost 4 times to the decapsulation oracle.

4.4.4 To retrieve the full secret s

To find the secret $s_i[j]$, first we have to construct the ciphertext $c = (\mathbf{b}', c_m)$ such that 0^{th} bit of the decrepted message is depends on the secret $s_i[j]$4.4.1.

Now if we choose $c = (\mathbf{b}', c_m)$, such that only $b_i[k]$ is non zero, where k is a number such that $j + k = 0$ others coefficients of $\mathbf{b}'$ are zero. and $c_m = 0$. Then the 0^{th} coefficient of the decrypted message will be:

$$m'[0] = \begin{cases} (s_i[j]b_i[k] + h_2 \mod p) \texttt{>>} 9 \;\texttt{if}\; j+k=0 \\ (-s_i[j]b_i[k] + h_2 \mod p) \texttt{>>} 9 \;\texttt{if}\; j+k=256 \end{cases}$$

i.e.,

$$m'[0] = \begin{cases} (s_i[j]b_i[k] + h_2 \mod p) \texttt{>>} 9 \;\texttt{if}\; j=k=0 \\ (-s_i[j]b_i[k] + h_2 \mod p) \texttt{>>} 9 \;\texttt{if}\; k=256-j \end{cases}$$

So each $m'[0]$ depends on $b_i[k], a$ and $s_i[j]$ (or $-s_i[(j+k) \mod 256]$ depending on the condition $(j+k)=0$ or $(j+k)=256$.

So for a constructed fixed ciphertext c with non zero value $b_i[k]$, (where $j+k=0$ or $j+k=256$) which we select from the 4.2, $m'[0]=0$ or not depends on the secret coefficient $s_i[j]$ if $j=k=0$ or on $-s_i[j]$, if $k=256-j$.

Let

$$s'_i[j,k] = \begin{cases} s_i[j] \;\texttt{if}\; j=k=0 \\ -s_i[i] \;\texttt{if}\; k=256-j \end{cases}$$

Then we can write the 0^{th} coefficient of decrypted message of the constructed ciphertext c as follows:

$$m'[0] = (s'_i[j,k]b_i[k] + h_2 \mod p) \texttt{>>} 9$$

For fixed $(b_i[k], a)$ we call the coefficient $s'_i[j,k]$ is in class $\texttt{X}_1$, if

$$m'[0] = 1$$

otherwise we call $s'_i[j]$ is in class $\texttt{X}_0$. We choose $b_i[(256-j) \mod 256] = b_0[0]$ and processing similar steps of finding the secret $s_0[0]$, we can find $s1_i[j]$ $\forall i,j$. Then from this we get $s_i[j]$.

4.4.5 Total number of queries

For each secret $s_i[j]$, we have to query atmost 4 times to the decapsulation oracle. Now there are total 768 many secret coefficients. So the total number of the query is $768 \times 4 = 3072$.

In this section, we describe an attack model to recover the secret $\mathbf{s}$. Here we construct the model by assuming the assumptions stated above. We have not demonstrated the attack practically. We only write a program which is following this model and at the end this program outputs a vector $\mathbf{s}'$. We check that $\mathbf{s}'$ satisfies the relation $\lfloor \mathbf{A}.\mathbf{s} \rceil_{\mathbf{q}\to\mathbf{p}} = \mathbf{b}$. So in the future, if we are able to inject this fault practically in the device where the decapsulation mechanism runs, then our model will work. The pseudo-code to simulate attack is described in Algorithm16.

4.5 Generalize version of model 2

There is no speciality of the 0^{th} coefficient. If we assume that we can see only the k^{th} bit of decryption message m' but we don't know the position k. Now we construct ciphertext

$c = (\mathbf{b}', c_m)$, where $b_i[j]$ as non zero and $c_m = 0$ while all the other coefficients of $\mathbf{b}'$ are set to zero. Then the k^{th} coefficient of the message is:

$$m'[k] = \begin{cases} (s_i[p]b_i[j] + h_2 \quad mod\ p)>>9 & \text{if } p + j = k \\ (-s_i[p]b_i[j] + h_2 \quad mod\ p)>>9 & \text{if } p + j = k + 256 \end{cases}$$

The set of (p, j) pairs such that $p + j = k$ is $=\{(0, k), (1, k-1), \ldots\ldots (k, 0)\}$. So if we query the constructed ciphertext with non zero $b_i[k]$, then by previous method, we recover $s_i[0]$, similarly with non zero $(b_i[k-1])$, we recover $s_i[1]$ so on.

If we query the constructed ciphertext with non zero $b_i[255]$, then by previous method, we recover $-s_i[k+1]$, similarly with non zero $b_i[254]$, we recover $-s_i[k+2]$ so on. Proceeding similar way we can find $s_i[j]$, where $i \in \{0, 1, 2\}$ and $0 \leq j < 256$. Here k will be any position of $\{0, 1, \ldots, 255\}$. For each $k \in \{0, 1, \ldots, 255\}$, we get a vector of polynomials say $\mathbf{s}^{(k)}$. Then starting from $k = 0$ we check $\lfloor \mathbf{A} \cdot \mathbf{s}^{(k)} \rceil = \mathbf{b}'$ or not. If they are equal, then the secret $\mathbf{s} = \mathbf{s^{(k)}}$ and we stop the process, otherwise we increase k by 1 and continue this process.

After getting the values $s_i[0], s_i[1], \ldots, s_i[k], -s_i[k+1], -s_i[k+2], \ldots, -s_i[255]$, where $i \in \{0, 1, 2\}$ and k is fixed number, we use the Algorithm 11 to get the actual secret $\mathbf{s}$.

Algorithm 11:

Data: The values $Rotr(s_i, j)[k]$, where $i \in \{0, 1, 2\}$, $j \in \{0, 1, \ldots, 255\}$, public key $pk = (\mathbf{A}, \mathbf{b}')$

Result: The actual secret $\mathbf{s}$

```
for k = 0; k < 255; k+ do
    for i = 0; i < 3; i++ do
        for j = 0; j ≤ k; j++ do
            s^(k)[i][j] = s^(k)[i][j];
        for j = k+1; j ≤ 255; j++ do
            s^(k)[i][j] = −s^(k)[i][j];
        b^(k) = ⌊A.s^(k)⌉;
        if b^(k) = b′ then
            s = s^(k);
            break;
```

4.5.1 Total number of queries

First we get the sequence $s_i[0], s_i[1], \ldots, s_i[k], -s_i[k+1], -s_i[k+2], \ldots, -s_i[255]$, where $i \in \{0, 1, 2\}$ by processing similar way. Now we know k is a fixed value but we don't know the value k. So the number of position of k is 256. For each case we find the corresponding secret and check that this secret is actual or not.

4.6 Our Proposed Attack Model 3

In this model first, we generate valid ciphertext and corresponding shared key with help of the encapsulation process. Then inject a fault by skipping one instruction for one coefficient of decryption posses which is running in the decapsulation mechanism. After that, we query this valid ciphertext to the faulted decapsulation oracle. Then depending upon the fault is effective [4] or ineffective [5] we construct a system of linear inequality with secret variables. In the paper [29] Pessl present a fault attack on Kyber and New Hope. By skipping one instruction in the decapsulation method, they generate a system of linear inequality with unknown secrets. Then solving the inequality, they are able to find the secret. In this section, we describe the model up to generating the linear inequality, by skipping a fault in the decapsulation process.

4.6.1 Observation

Now in the decryption (Algorithm 4) we calculate the two steps Input:$(\mathbf{b}', c_m)$
$v = \mathbf{b}'^{\mathbf{T}}.\mathbf{s}$
$m' = (v - 2^{\epsilon_p - \epsilon_T} \cdot c_m + h_2) >> 9$

Let $M' = v - 2^{\epsilon_p - \epsilon T} \cdot c_m + h_2$ then,

$$\begin{aligned}
M' &= v - 2^{\epsilon_p - \epsilon T} \cdot c_m + h_2 \\
&= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - \frac{p}{T} \cdot c_m + h_2 \\
&= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - (\frac{p}{T} \cdot c_m + E_2) + h_2 \\
&= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - (\mathbf{b}^{\mathbf{T}}.\mathbf{s}' + h_1 - \frac{p}{2} \cdot m) - E_2 + h_2 \\
&= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - (\mathbf{b}^{\mathbf{T}}.\mathbf{s}' + h_1) + \frac{p}{2} \cdot m - E_2 + h_2
\end{aligned}$$

where $v = \mathbf{b}'^{\mathbf{T}}.\mathbf{s} \in R_p$ and $2^{\epsilon_p - \epsilon_T} = \frac{p}{T}$, E_2 is a rounding error and the value c_m taken from Algorithm 3. Let d be the decryption noise i.e.,

$$\begin{aligned}
d &= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - (\mathbf{b}^{\mathbf{T}}.\mathbf{s}' + h_1) - E_2 + h_2 \mod p \\
&= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - (\mathbf{b}^{\mathbf{T}}.\mathbf{s}' + h_1 - \frac{p}{2} \cdot m) - \frac{p}{2} \cdot m - E_2 + h_2 \mod p \\
&= \mathbf{b}'^{\mathbf{T}}.\mathbf{s} - \frac{p}{T} \cdot c_m - \frac{p}{2} \cdot m + h_2 \mod p
\end{aligned}$$

[4]If the decrypted message changed after injecting fault, then the decapsulation oracle return a random shared key, then we call it as effective fault

[5]If the decrypted message remains the same after injecting fault, then the decapsulation oracle return the shared key, which we get from encapsulation process, then we call it as ineffective fault

where $c_m = \mathbf{b^T} \cdot \mathbf{s}' + h_1 - \frac{p}{2} \cdot m$. Now $0 \leq d < \frac{p}{2}$, because if $d \geq \frac{p}{2}$, then we can't recover the actual message. So

$$\frac{p}{2} \cdot m \leq d + \frac{p}{2} \cdot m < \frac{p}{2} + \frac{p}{2} \cdot m$$
$$i.e., \quad \frac{p}{2} \cdot m \leq M' < \frac{p}{2} + \frac{p}{2} \cdot m$$

So if $m = 1$, then $\frac{p}{2} \leq M' < p$. Also if $m = 0$, then $0 \leq M' < \frac{p}{2}$.

Let $d' = d - h_2 \mod p$ and $M'' = d' + \frac{p}{2} \mod p$. Then

$$-h_2 \mod p \leq d - h_2 \mod p < \frac{p}{2} - h_2 \mod p$$
$$i.e., \quad -h_2\ mod\ p \leq d' < \frac{p}{2} - h_2 \mod p$$

If $0 \leq d < h_2$, then $-h_2 \mod p \leq d' < 0$. In this case

$$\frac{p}{2} \cdot m - h_2 \mod p \leq d' + \frac{p}{2} \cdot m < \frac{p}{2} \cdot m$$
$$i.e., \quad \frac{p}{2} \cdot m - h_2\ mod\ p \leq M'' < \frac{p}{2} \cdot m$$

If $0 \leq d < h_2$ and we skip the addition with h_2 in the last step of decryption, then the decrypted message will be M''>>9. In this case if actual message bit $m = 1$, then $\frac{p}{2} - h_2 \mod p \leq M'' < \frac{p}{2}$. So in this case the faulted decrypted message bit m' will be 0. By similar calculating, we see that if the actual message bit is $m = 0$, the decrypted message bit m' will be 1.

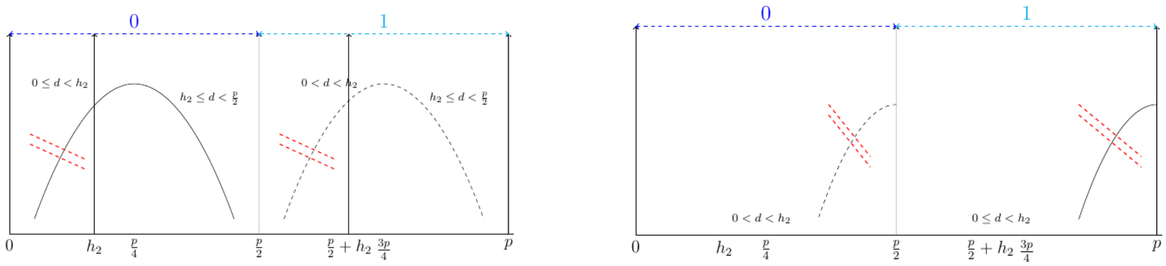
So from the above observation, we can say that if we don't add h_2 in the last step of decryption and $0 \leq d < h_2$, then the decrypted message bit is not equal to the actual message bit. [6] Also by similar calculation we can see that if we don't add h_2 in the last step of decryption and $d \geq h_2$, then the decrypted message bit is equal to the actual message bit.

Now we query with a valid ciphertext c to decapsulation oracle and h_2 is not added for one coefficient in the last step of decryption, which is a part of the decapsulation process. If the oracle gives a valid shared key for that ciphertext, then we will arrive at a conclusion that $d \geq h_2$ otherwise $d \leq h_2$.

4.6.2 Fault Assumption

We inject a fault in decapsulation in such a way that, when it decrypt the 0^{th} coefficient, then it skips the instruction "adding with h_2".

[6] if the encryption noise $0 \leq d < h_2$, then the faulted decoding returns an incorrect result.i.e, the fault is effective. and if the encryption noise $h_2 \geq d$, then the faulted decoding returns an correct result. i.e., the fault is ineffective.



(a) Without injecting the fault

(b) After injecting the fault

Figure 4.8

4.6.3 Structure of Attack Simulation Model

1. We construct a valid ciphertext $c = (\mathbf{b}', c_m)$, for a message m of 256 bits by KEM.Encaps algorithm.

2. We query with the ciphertext c to the decapsulation oracle in which we inject a fault (stated in Fault Assumption).

3. If the decapsulation output is $\mathcal{H}(\bar{K}', c)$, then we consider it as "ineffective fault" and $d[0]$ and otherwise we call it as "effective fault" for the ciphertext c.

4. If we see that the fault is effective for the ciphertext c, then the 0^{th} coefficient of the decryption noise $d[0]$ for the ciphertext c will be $< h_2$. i.e.,

$$d[0] = ((b'^T.s)[0] - \frac{p}{T} \cdot c_m[0] - \frac{p}{2} \cdot m[0] + h_2) \mod p < h_2$$

.

$$i.e., \sum_{k=0}^{2}(b_k[0 \cdot s_k[0] - \sum_{j=1}^{255} b_k[256 - j \cdot s_k[j]) \mod p < \frac{p}{T} \cdot c_m[0] + \frac{p}{2} \cdot m[0] \mod p$$

5. If we see that the fault is ineffective for the ciphertext c, then the 0^{th} coefficient of the decryption noise $d[0]$ for the ciphertext c will be $\geq h_2$. i.e.,

$$d[0] = ((b'^T.s)[0] - \frac{p}{T} \cdot c_m[0] - \frac{p}{2} \cdot m[0] + h_2) \mod p \geq h_2$$

$$i.e., \sum_{k=0}^{2}(b_k[0 \cdot s_k[0] - \sum_{j=1}^{255} b_k[256 - j \cdot s_k[j]) \; mod \; p \geq \frac{p}{T} \cdot c_m[0] + \frac{p}{2} \cdot m[0] \mod p$$

6. From the above step we get a linear inequality which contains $3 \times 256 = 768$ many unknowns $s_i[j]$.

7. Repeating the whole process many times and store the inequality in a file.

8. In this method we get a system of linear inequalities, with 768 many unknowns.

9. Now our problem is reduce to the problem find a solution of a system of linear inequalities where the unknowns $s_i[j] \in \{-4, -3, \ldots, 0, 1, \ldots, 4\}$.

10. After solving this system of inequality we can extract the secret s.

In [29], they have done similar work to attack Kyber. We have done the above process up to storing a system of a linear-inequality.

4.7 Conclusion:

In this chapter we describe three attack models, we are able to find secrets by attack simulation for two models (Model1 and Model 2). In practical purpose Model 1, is totally depends on the practicality of skipping an instruction in the decryption process and capability of distinguishing message 0 and 1 by Em-power analysis. In the paper [34], they show that by EM-power analysis, we are able to distinguish two cases. This attack seems to be practically possible.

Chapter 5

Conclusion and future work

5.1 Conclusion

In this thesis, we have seen that if we assume the assumption of model 1 and model 2, then we can recover the secret. Here we present our attack for the parameter set ($n = 256, l = 3, q = 2^{13}, p = 2^{10}, \mu = 8$). The parameter set has no extra significance in these attacks. So can do the same attacks for the others parameters of SABER. In this thesis, we describe an attack model 3 and we can construct a system of linear inequality such that the secret satisfies the linear inequality. Our next target is to solve this system of inequality.

5.2 Future work

During my master's thesis we get some models Model 1 and Model 2 and then doing the attack simulation, we have seen that we recover the secret. Now we are planning to do the following after this internship:

- From the paper of Ravi et al. [34], we have seen that we can distinguish two cases: 1. when the decrypted message $m^{'} = 0$ (all bits are zero) and 2. when the decrypted message $m^{'} = 1$ (all bits are zero except the LSB) and from the paper of Pessl and Prokop [29], we have seen that we can skip one instruction for one step. Based on these assumption, we have constructed our model 1 thus have done the attack simulation. But we want to do this attack physically. Also, we want to check if there is another way to distinguish between these two cases.

- Assuming that we can see one decrypted message bit, we complete our attack 2. But we don't know how much feasible this assumption is for practical purposes. So my next target is to check the practicality of model 2.

- My 3rd and the most important target is to solve the system of inequalities, which recovers the actual secret, described in mode 3. It is important because the assumption of model 3 is weak (i.e., this assumption is practically possible)[29].

- Now we compute the inner product of the vectors $\mathbf{s}$ and $\mathbf{b}^{'}$ in decryption, which is a part of the decapsulation algorithm. For computing the inner product, we use the karatsuba, Toom-Cook 3 and school-book multiplications. we have observed that the multiplication steps depend on the secret coefficients. we want to check whether the secret information is being licked or not after injecting fault in some step of multiplication.

- After the fault attack, the important task is to strong the scheme SABER with halp of the countermeasure against these fault attacks.

Bibliography

[1] Post-quantum cryptography pqc. URL https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[2] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the bkw algorithm on lwe. Cryptology ePrint Archive, Report 2012/636. https://eprint.iacr.org/2012/636.

[3] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede. Time-memory trade-off in toom-cook multiplication: an application to module-lattice based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):222–244. doi: 10.13154/tches.v2020.i2.222-244. URL https://tches.iacr.org/index.php/TCHES/article/view/8550.

[4] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. Cryptology ePrint Archive, Report 2015/769. https://eprint.iacr.org/2015/769.

[5] D. Boneh, R. DeMillo, and R. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*.

[6] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. Cryptology ePrint Archive, Report 2016/659, . https://eprint.iacr.org/2016/659.

[7] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 353–367, . doi: 10.1109/EuroSP.2018.00032.

[8] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Report 2017/634, . https://eprint.iacr.org/2017/634.

[9] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Report 2017/634, 2017. https://eprint.iacr.org/2017/634.

[10] Murray R. Bremner. *Lattice Basis Reduction: An Introduction to the LLL Algorithm and Its Applications*. CRC Press, Inc., USA, 1st edition. ISBN 1439807027.

[11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security*, volume 7073 of *LNCS - Lecture Notes in Computer Science*, pages 1–20, Seoul, South Korea. Springer. doi: 10.1007/978-3-642-25385-0_1. URL https://hal.inria.fr/hal-01109961.

[12] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. Lwe with side information: Attacks and concrete security estimation. Cryptology ePrint Archive, Report 2020/292. https://eprint.iacr.org/2020/292.

[13] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi: 10.1109/TIT.1976.1055638.

[14] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. Cryptology ePrint Archive, Report 2018/230, . https://eprint.iacr.org/2018/230.

[15] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. Cryptology ePrint Archive, Report 2018/230, . https://eprint.iacr.org/2018/230.

[16] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6/7):467–488.

[17] Borko Furht, editor. *The RSA Public-Key Encryption Algorithm*, pages 757–757. Springer US, Boston, MA. ISBN 978-0-387-30038-2. doi: 10.1007/0-387-30038-4_206. URL https://doi.org/10.1007/0-387-30038-4_206.

[18] O. Goldreich, D. Micciancio, S. Safra, and J. P. Seifert. Approximating shortest lattice vectors is not harder than approximating closet lattice vectors. *Inf. Process. Lett.*, 71 (2):55–61. ISSN 0020-0190. doi: 10.1016/S0020-0190(99)00083-6. URL https://doi.org/10.1016/S0020-0190(99)00083-6.

[19] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604. https://eprint.iacr.org/2017/604.

[20] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, 1996.

[21] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International*

Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi: 10.1007/3-540-68697-5_9.

[22] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 388–397, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3540663479.

[23] F. Koeune and François-Xavier Standaert. A tutorial on physical security and side-channel attacks. In *FOSAD*, .

[24] Franço is Koeune and François-Xavier Standaert. *A Tutorial on Physical Security and Side-Channel Attacks*, page 78–108. Springer-Verlag, Berlin, Heidelberg, . ISBN 3540289550.

[25] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, .

[26] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6), November . ISSN 0004-5411. doi: 10.1145/2535925. URL https://doi.org/10.1145/2535925.

[27] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, . https://eprint.iacr.org/2012/230.

[28] Isaiah Lankham Bruno Nachtergaele and Anne Schilling. The Gram-Schmidt Orthogonalization procedure. URL https://math.libretexts.org/@go/page/260. [Online; accessed 2021-07-01].

[29] Peter Pessl and Lukas Prokop. Fault attacks on cca-secure lattice kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):37–60, . doi: 10.46586/tches.v2021.i2.37-60. URL https://tches.iacr.org/index.php/TCHES/article/view/8787.

[30] Peter Pessl and Lukas Prokop. Fault attacks on cca-secure lattice kems. Cryptology ePrint Archive, Report 2021/064, . https://eprint.iacr.org/2021/064.

[31] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Info. Comput.*, 3(4):317–344, July 2003. ISSN 1533-7146.

[32] Jean-Jacques Quisquater. Electromagnetic analysis.

[33] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "not used" once - practical fault attack on pqm4 implementations of nist candidates. Cryptology ePrint Archive, Report 2018/211, . https://eprint.iacr.org/2018/211.

[34] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, . doi: 10.13154/tches.v2020.i3.307-335. URL https://tches.iacr.org/index.php/TCHES/article/view/8592.

[35] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, . Association for Computing Machinery. ISBN 1581139608. doi: 10.1145/1060590.1060603. URL https://doi.org/10.1145/1060590.1060603.

[36] Oded Regev. Lecture notes: Lattices in computer science. . URL https://cims.nyu.edu/~regev/teaching/lattices_fall_2009.

[37] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL https://doi.org/10.1145/359340.359342.

[38] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509. ISSN 0097-5397. doi: 10.1137/S0097539795293172. URL https://doi.org/10.1137/S0097539795293172.

[39] Felipe Valencia, Tobias Oder, Tim Güneysu, and Francesco Regazzoni. Exploring the vulnerability of r-lwe encryption to fault attacks. In *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems*, CS2 '18, page 7–12, New York, NY, USA. Association for Computing Machinery. ISBN 9781450363747. doi: 10.1145/3178291.3178294. URL https://doi.org/10.1145/3178291.3178294.

Appendix A

LPR scheme

Algorithm 12: LPR Key Generation

Input:

Output: Keypair (pk, sk)

$s, e \in \mathcal{R}_q \leftarrow \mathcal{X}^n$;

$a \leftarrow \mathcal{U}_q^n$;

$b = as + e$;

$return(pk = (a, b), sk = (a, s))$;

Algorithm 13: LPR Encryption

Input: Public key $pk = (a, b)$, $n - bit$ message m

Output: Ciphertext (u, v)

1: $r, e_1, e_2 \in \mathcal{R}_q \rightarrow \mathcal{X}^n$;

2: $u = ar + e_1$;

3: $v = br + e_2 + m\lfloor \frac{q}{2} \rfloor$;

4: return (u, v);

Algorithm 14: LPR Decryption

Input: Secret key $sk = (a, s)$, ciphertext (u, v)

Output: Message m

1: $m^{'} = v - us$;

2: return Decode$(m^{'})$;

Appendix B

Psudo-code of attack model 1

15

Algorithm 15: Attack Simulation

```
Output: Secret s
for i = 0; i < 3; i++ do
    for j = 0; j < 256; j++ do
        c1 = (0x1, 0x0);
        create_ciphertext(i, (256 − j) mod 256, c1, ciphertext) ;
        decaps_fault(ciphertext, t);
        if t = 0 then
            c5 = (0x16, 0x1);
            create_ciphertext(i, (256 − j) mod 256, c5, ciphertext) ;
            decaps_fault(ciphertext, t);
            if t = 0 then
                c3 = (0x10, 0x1);
                create_ciphertext(i, (256 − j) mod 256, c3, ciphertext) ;
                decaps_fault(ciphertext, t);
                if t = 0 then
                    s[i][j] = −4;
                else
                    s[i][j] = −3;
            else
                c7 = (0x21, 0x1);
                create_ciphertext(i, (256 − j) mod 256, c7, ciphertext) ;
                decaps_fault(ciphertext, t);
                if t = 0 then
                    s[i][j] = −2;
                else
                    c8 = (0x3c7, 0x0);
                    create_ciphertext(i, (256 − j) mod 256, c8, ciphertext) ;
                    decaps_fault(ciphertext, t);
                    if t = 0 then
                        s[i][j] = 0;
                    else
                        s[i][j] = −1;
        else
            c4 = (0x16, 0xf);
            create_ciphertext(i, (256 − j) mod 256, c4, ciphertext) ;
            decaps_fault(ciphertext, t);
            if t = 0 then
                c6 = (0x21, 0xf);
                create_ciphertext(i, (256 − j) mod 256, c6, ciphertext) ;
                decaps_fault(ciphertext, t);
                if t = 0 then
                    s[i][j] = 1;
                else
                    s[i][j] = 2;
            else
                c2 = (0x11, 0xf);
                create_ciphertext(i, (256 − j) mod 256, c2, ciphertext) ;
                decaps_fault(ciphertext, t);
                if t = 0 then
                    s[i][j] = 3;
                else
                    s[i][j] = 4;
for i = 0; i < 3; i++ do
    s[i][0] = −s[i][0];
```

In the `attack_simulation` algorithm (Algorithm 15) we use two functions `create_ciphertext`()

and `decaps_fault`(). Here using the function `create_ciphertext`, we create a $ciphertext = (b', c_m)$ (may not be valid) such that the $b_i[(256 - j)] = c[0]$ and others coefficients of b' are zero and each coefficient of c_m is $c[1]$.

Using the function `decaps_fault`, we send the structured *ciphertext* to faulted decapsulation oracle and t is the distingusher of two cases $m' = 0$ and $m' = 1$.

After running the Attack simulation algorithm we get a secret $\mathbf{s}$, then we check each coefficient $s_i[j]$ with the actual secret. We see that both are the same. So our attack simulation is successful.

Appendix C

Simulation Program of Model 2

Algorithm 16: Attack Simulation2

```
Input:
Output: Secret s
for i = 0; i < 3; i++ do
    for j = 0; j < 256; j++ do
        c_1 = 0x8e;
        create_ciphertext(i, (256 − j) mod 256, c_1, ciphertext) ;
        decaps_fault2(ciphertext, t);
        if t = 0 then
            c_2 = 0x11c;
            create_ciphertext(i, (256 − j) mod 256, c_2, ciphertext) ;
            decaps_fault2(ciphertext, t);
            if t = 0 then
                s_[i][j] = 0
            else
                c_3 = 0x10a;
                create_ciphertext(i, (256 − j) mod 256, c_3, ciphertext) ;
                decaps_fault2(ciphertext, t);
                if t = 0 then
                    s[i][j] = 1;
                else
                    s[i][j] = −1;
        else
            c_4 = 0x5f;
            create_ciphertext(i, (256 − j) mod 256, c_4, ciphertext) ;
            decaps_fault2(ciphertext, t);
            if t = 0 then
                c_7 = 0x73;
                create_ciphertext(i, (256 − j) mod 256, c_7, ciphertext) ;
                decaps_fault2(ciphertext, t);
                if t = 0 then
                    s[i][j] = 2;
                else
                    s[i][j] = −2;
            else
                c_5 = 0xc7;
                create_ciphertext(i, (256 − j) mod 256, c_7, ciphertext) ;
                decaps_fault2(ciphertext, t);
                if t = 0 then
                    c_8 = 0x1ba;
                    create_ciphertext(i, (256 − j) mod 256, c_8, ciphertext) ;
                    decaps_fault2(ciphertext, t);
                    if t = 0 then
                        s[i][j] = −4;
                    else
                        s[i][j] = 4;
                else
                    c_6 = 0x1a2;
                    create_ciphertext(i, (256 − j) mod 256, c_6, ciphertext) ;
                    decaps_fault2(ciphertext, t);
                    if t = 0 then
                        s[i][j] = 3;
                    else
                        s[i][j] = −3;
for i = 0; i < 3; i++ do
    s[i][0] = −s[i][0];
```

In the `attack_simulation2` algorithm (Algorithm 16) we use two functions `create_ciphertext()` and `decaps_fault2()`. Here using the function `create_ciphertext`, we create a *ciphertext* $= (b', c_m)$ (may not be valid) such that the $b_i[k] = c1$, where k depend on j and others coefficientsof $\mathbf{b}'$ are zero and each coefficient of c_m is 0. Using the function `decaps_fault2`, we send the structured *ciphertext* to faulted decapsulation oracle and t is the first coefficient of decrypted message.

After running the Attack simulation algorithm2 we get a $\mathbf{s}$, then we check each coefficient $s_i[j]$ with actual secret up to 13 bits. We see that both are the same. So our attack simulation is successful.