




A Lattice Attack on CRYSTALS-Kyber with Correlation Power Analysis

Yen-Ting Kuo   and Atsushi Takayasu 

The University of Tokyo, Tokyo, Japan
{kuruwakuo,takayasu-a}@g.ecc.u-tokyo.ac.jp

Abstract. *CRYSTALS-Kyber* is a key-encapsulation mechanism, whose security is based on the hardness of solving the *learning-with-errors* (LWE) problem over module lattices. As in its specification, Kyber prescribes the usage of the *Number Theoretic Transform* (NTT) for efficient polynomial multiplication. Side-channel assisted attacks against *Post-Quantum Cryptography* (PQC) algorithms like Kyber remain a concern in the ongoing standardization process of quantum-computer-resistant cryptosystems. Among the attacks, *correlation power analysis* (CPA) is emerging as a popular option because it does not require detailed knowledge about the attacked device and can reveal the secret key even if the recorded power traces are extremely noisy. In this paper, we present a two-step attack to achieve a full-key recovery on lattice-based cryptosystems that utilize NTT for efficient polynomial multiplication. First, we use CPA to recover a portion of the secret key from the power consumption of these polynomial multiplications in the decryption process. Then, using the information, we are able to fully recover the secret key by constructing an LWE problem with a smaller lattice rank and solving it with lattice reduction algorithms. Our attack can be expanded to other cryptosystems using NTT-based polynomial multiplication, including Saber. It can be further parallelized and experiments on simulated traces show that the whole process can be done within 20 minutes on a 16-core machine with 200 traces. Compared to other CPA attacks targeting NTT in the cryptosystems, our attack achieves lower runtime in practice. Furthermore, we can theoretically decrease the number of traces needed by using lattice reduction if the same measurement is used. Our lattice attack also outperforms the state-of-the-art result on integrating side-channel hints into lattices, however, the improvement heavily depends on the implementation of the NTT chosen by the users.

Keywords: CRYSTALS-Kyber, lattice, side-channel attack, number theoretic transform

1 Introduction

1.1 Background

With the development of quantum computation, what is usually hard to solve on the traditional computer (factorization, DLP, etc) will become efficiently solvable

by applying Shor’s algorithm [26], which will make the public-key cryptosystems most people use now unreliable. Thus, there is a significant interest in post-quantum cryptography (PQC) algorithms, which are based on mathematical problems presumed to resist quantum attacks. To standardize such algorithms, the National Institute of Standards and Technology (NIST) initiated a process to solicit and evaluate PQC candidates being submitted [22]. After three rounds of the process, they had identified four candidate algorithms for standardization and four more to be evaluated in round 4.

CRYSTALS-KYBER (Kyber) [2] is one out of the four candidates that are confirmed to be standardized in July, 2022, and it is the only public-key encryption and key-establishment algorithm. It belongs to the category of lattice-based cryptography, and in particular a module Learning With Errors (module-LWE) scheme. Kyber prescribes the usage of the Number Theoretic Transform (NTT) for efficient polynomial multiplication. Via point-wise multiplication of transformed polynomials, i.e., $ab = \text{NTT}^{-1}(\text{NTT}(a) \circ \text{NTT}(b))$, multiplication can be performed in time $O(n \log n)$, where n is the degree of polynomial a and b . Kyber has three parameter sets: Kyber512, Kyber768 and Kyber1024 with security level similar to that of AES128, AES192 and AES256.

Power analysis attacks, introduced by Kocher [15,16], exploit the fact that the instantaneous power consumption of a cryptographic device depends on the data it processes and on the operation it performs. There exist simple power analysis attacks on Kyber that can compromise a message or private key using only one or several traces. In particular, Primas *et al.* [24] and Pessl *et al.* [23] recover data passed through an NTT by templating the multiplications or other intermediate values within the NTT. Hamburg *et al.* [13] present a sparse-vector chosen ciphertext attack strategy, which leads to full long-term key recovery. These attacks are still limited in that they either require extensive profiling efforts or they are only applicable in specific scenarios like the encryption of ephemeral keys.

As opposed to above methods, Mujdei *et al.* [21] showed that leakage from the schoolbook polynomial multiplications after the incomplete NTT can be exploited through correlation power analysis (CPA) style attacks. CPA attacks exploit the dependency of power consumption on intermediate values, we provide an introduction of CPA attacks below and refer to work of Mangard *et al.* [18] for further details. The presented attack required 200 power traces to recover all the coefficients, which enables full key recovery. More precisely, they guess two coefficients at once within the range $(-\frac{q}{2}, \frac{q}{2}]$, implying a search over q^2 combinations.

In order to model the effect of these side-channel leakage, Dachman-Soled *et al.* [8] proposed a general lattice framework that quantifies the LWE security loss when revealing a so-called hint $(\mathbf{v}, \mathbf{w}, l) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^m \times \mathbb{Z}$ satisfying

$$\langle (\mathbf{v}, \mathbf{w}), (\mathbf{s}, \mathbf{e}) \rangle = l.$$

The inner product of this equation is usually performed in \mathbb{Z}_q , which is referred to as *modular-hint*. They also dealt with leakage $l \bmod q$ reduction, a

so-called *perfect hint*. Their results was later improved by May and Nowakowski [19], where they only addressed hints for the secret \mathbf{s} only, i.e., hints (\mathbf{v}, l) with $\langle \mathbf{v}, \mathbf{s} \rangle = l$.

1.2 Our Contribution

In this paper, we propose a way that utilizes correlation power analysis to fully recover the secret key of Kyber. Our attack consists of two steps. First, by exploiting the correlation of Hamming weight of some intermediates and the power consumption of the decryption process in Kyber, precisely the part where we multiply the secret polynomial with ciphertext, we can recover some of the coefficients of the secret key in the NTT domain. Secondly, since there will be some ambiguity about whether the recovered coefficients are indeed correct, we sample part of the recovered coefficients and construct a lattice problem by Kannan’s embedding proposed by [14]. Then one can recover the entire secret key by solving the lattice problem by using lattice reduction algorithms such as BKZ [5].

We also examined the attack on simulated traces of ARM cortex-M0 generated by a toolkit named ELMO [10]. Experiments show that we can indeed recover the secret key with 200 traces. With some fine-tuning on the acceptance threshold of power analysis, we can even have guaranteed success in sampling all correct coefficients with 600 traces and still have enough ones to construct a solvable lattice problem.

There are three parameter sets for Kyber, and our attack can be easily adapted to all parameter sets. The time it takes to recover the secret key is linear to the number of coefficients in the secret key. The power analysis part of our attack can be parallelized to further accelerate the process. Although the idea of our attack is similar to that of Mujdei *et al.* [21], we only require $O(q)$ search, which directly reflects on the runtime of the CPA. For reference, our attack is about 16 times faster than Mujdei *et al.* [21] without parallelization. Since our SCA and that of [21] use different methods of measurement, it is hard to compare the result. However, if we use the same measurement, by using the lattice reduction, we can theoretically decrease the number of required power traces. It may get some wrong coefficients by doing so, but we can fix that by sampling portion of recovered coefficients and using lattice reduction to find the rest of them.

For the lattice attack part of our attack, as opposed to the above methods, our approach uses divide-and-conquer methods in a way that we only consider a portion of the secret key at a time. That is, the hints $\langle \mathbf{v}, \mathbf{s} \rangle = l$ gathered from our method are inner products of vectors with smaller dimension. This can be done because in the computation of decryption of Kyber, the secret key is divided into blocks by the intrinsic property of module-LWE. Furthermore, the NTTs of each sub-key are usually incomplete since it can achieve fastest speed in that way [6]. Due to these properties of Kyber, The techniques of Dachman-Soled *et al.* [8] and May *et al.* [19] to solve the LWE instance are not suitable for our cases. Since we only consider a portion of the secret key at a time. The number

of hints we need is extremely lower than their methods. However, we do need to perform multiple times of the lattice reduction to achieve a full key recovery.

Our lattice attack can be applied to other cryptosystems that utilizes NTT-based polynomial multiplications. For example, Saber [9] is a lattice-based KEM based on Module Learning With Rounding problem. Although it is not specifically designed to use NTT by choosing an *NTT-friendly* ring, it is still possible to achieve fast computation by NTT by enlarging the ring as shown in the work by Chung *et al.* [6]. However, the improvement from our attack depends on the implementation of the NTT, namely how many layers of NTT the implementation chooses to apply to it.

We use the official reference implementation of the Kyber key encapsulation mechanism provided by the authors [3] as the target. We also provide an efficient open-source Python implementation of our framework. The source code is available at <https://github.com/kuruwa2/kyber-sca>.

Organization. The rest of this paper is organized as follows. In Section 2, we introduce how Kyber is implemented with Number Theoretic Transform. In Section 3, we illustrate how to apply differential power analysis to the NTT part of Kyber. In Section 4, we construct a simpler lattice problem from the recovered coefficients and conduct an experiment by lattice reduction algorithm to determine the least number of coefficients we need to recover from differential power analysis. In Section 5, we analyze the success rate of our attack and conclude the paper.

2 Preliminaries

In this section, we explain the lattices and module-learning with errors problem, go into some details about Kyber, and review the Number Theoretic Transform.

2.1 Lattices

Let $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_n] \in \mathbb{Z}^{m \times n}$ be an integer matrix. We denote by

$$\Lambda(\mathbf{B}) := \{\alpha_1 \mathbf{b}_1 + \dots + \alpha_n \mathbf{b}_n \mid \alpha_i \in \mathbb{Z}\}$$

the lattice generated by \mathbf{B} . If the rows of \mathbf{B} are linearly independent, \mathbf{B} is a *basis matrix* of $\Lambda(\mathbf{B})$. The number of rows n in any basis matrix of some lattice Λ is called the *rank* of Λ . The *determinant* of a lattice Λ with basis matrix \mathbf{B} is defined as

$$\det(\Lambda) := \sqrt{\det(\mathbf{B}\mathbf{B}^T)}$$

The determinant does not depend on the choice of basis. We also denote by $\lambda_i(\Lambda)$ the *i-th successive minimum* of Λ . A lattice vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\| = \lambda_1(\Lambda)$ is called the *shortest vector* of Λ . $\lambda_1(\Lambda)$ can be estimated by the following heuristic.

Heuristic 1 (Gaussian Heuristic) *Let Λ be an n -dimensional lattice. Gaussian heuristic predicts that the norm of the shortest vector $\lambda_1(\Lambda)$ equals*

$$gh(\Lambda) := \sqrt{\frac{n}{2\pi e}} \det(\Lambda)^{1/n}.$$

2.2 Module-LWE

Learning with errors (LWE) problem [25] and its extension over rings [17] or modules are the basis of multiple NIST PQC candidates.

Let \mathbb{Z}_q be the ring of integers modulo q and for given power-of-2 degree n , define $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ as the polynomial ring of polynomials modulo $x^n + 1$. For any ring \mathcal{R} , $\mathcal{R}^{\ell \times k}$ denotes the ring of $\ell \times k$ -matrices over \mathcal{R} . We also simplify $\mathcal{R}^{\ell \times 1}$ to \mathcal{R}^ℓ if there is no ambiguity. Single polynomials are written without markup, vectors are bold lower case \mathbf{a} and matrices are denoted with bold upper case \mathbf{A} . β_η denotes the centered binomial distribution with parameter η and \mathcal{U} denote the uniform distribution. If χ is a probability distribution over a set S , then $x \leftarrow \chi$ denotes sampling $x \in S$ according to χ . If χ is only defined on \mathbb{Z}_q , $x \leftarrow \chi(\mathcal{R}_q)$ denotes sampling the polynomial $x \in \mathcal{R}_q$, where all coefficients of the coefficients in x are sampled from χ .

The learning with errors (LWE) problem was introduced by Regev [25] and its decision version states that it is hard to distinguish m uniform random samples $(\mathbf{a}_i, b_i) \leftarrow \mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$ from m LWE-samples of the form

$$(\mathbf{a}_i, b_i = \mathbf{a}_i^\top \mathbf{s} + e_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q,$$

where the secret vector $\mathbf{s} \leftarrow \beta_\eta(\mathbb{Z}_q^n)$ is fixed for all samples, $\mathbf{a}_i \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ and $e_i \leftarrow \beta_\eta(\mathbb{Z}_q)$ is a small error. A module version of LWE, called Mod-LWE [4] essentially replaces the ring \mathbb{Z}_q in the above samples by a quotient ring of the form \mathcal{R}_q with corresponding error distribution $\beta_\eta(\mathcal{R}_q)$.

$$(\mathbf{a}_i, b_i = \mathbf{a}_i^\top \mathbf{s} + e_i) \in \mathcal{R}_q^{k \times 1} \times \mathcal{R}_q.$$

The rank of the module is k and the dimension of the ring \mathcal{R}_q is n . The case $k = 1$ corresponds to the ring-LWE problem introduced in [17]. We also commonly integrate m number of samples by the matrix multiplication,

$$(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}) \in \mathcal{R}_q^{m \times k} \times \mathcal{R}_q^m.$$

Let $\lambda_i(A)$ denote the i -th minimum of lattice A . The LWE problem can be considered as an average version of the *Bounded Distance Decoding* (BDD) problem: Given a vector such that its distance from the lattice is at most $\lambda_1(A)/2$, the goal is to find the closest lattice vector to it. A dual problem of BDD is the so-called *unique Shortest Vector Problem* (uSVP): Given $\gamma \geq 1$, and lattice A such that $\lambda_2(A) \geq \gamma \cdot \lambda_1(A)$, the goal is to find a non-zero vector $\mathbf{v} \in A$ of norm $\lambda_1(A)$. The reduction between LWE, BDD, and uSVP will be further discussed in Section 4.2.

2.3 CRYSTALS-Kyber

Kyber [2] is a Key Encapsulation Mechanism (KEM) submitted to the NIST standardization process, and it is among the four confirmed candidates to be standardized [22]. The security of Kyber is based on the module-LWE problem. For the three parameter sets in the proposal, Kyber512, Kyber768, and

Table 1: Parameter sets for Kyber [1].

name	n	k	q	η_1	η_2
Kyber512	256	2	3329	3	2
Kyber768	256	3	3329	2	2
Kyber1024	256	4	3329	2	2

Kyber1024, the parameters are all set to $n = 256$ and $q = 3329$. For most parameters $\eta = 2$ is used, except for Kyber512, where $\eta = 3$. The parameter sets differ in their module dimension $k = 2, 3$, and 4 respectively. The three parameter sets listed in Table 1.

Kyber consists of the CCA2-KEM Key Generation, PKE- and CCA2-KEM-Encryption, and CCA2-KEM-Decryption algorithms, which are summarized in Algorithms 1, 2, 3 and 4, respectively.

Algorithm 1 Kyber-CCA2-KEM Key Generation (simplified)

Output: Public key pk , secret key sk

- 1: Choose uniform seeds ρ, σ, z
 - 2: $\mathcal{R}^{k \times k} \ni \hat{\mathbf{A}} \leftarrow \text{Sample}_{\mathcal{U}}(\rho)$
 - 3: $\mathcal{R}_q^k \ni \mathbf{s}, \mathbf{e} \leftarrow \text{Sample}_{\beta_\eta}(\sigma)$
 - 4: $\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$
 - 5: $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \text{NTT}(\mathbf{e})$
 - 6: **return** $(pk := (\hat{\mathbf{t}}, \rho), sk := (\hat{\mathbf{s}}, pk, \text{Hash}(pk), z))$
-

Algorithm 2 Kyber-PKE Encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{t}}, \rho)$, message m , seed τ

Output: Ciphertext c

- 1: $\mathcal{R}^{k \times k} \ni \hat{\mathbf{A}} \leftarrow \text{Sample}_{\mathcal{U}}(\rho)$
 - 2: $\mathcal{R}_q^k \ni \mathbf{r}, \mathbf{e}_1, \mathcal{R}_q \ni e_2 \leftarrow \text{Sample}_{\beta_\eta}(\tau)$
 - 3: $\mathbf{u} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}}^\top \circ \text{NTT}(\mathbf{r})) + \mathbf{e}_1$
 - 4: $v \leftarrow \text{NTT}^{-1}(\hat{\mathbf{t}}^\top \circ \text{NTT}(\mathbf{r})) + e_2 + \text{Encode}(m)$
 - 5: **return** $c := (\mathbf{u}, v)$
-

In these algorithms, and in the rest of this paper, the notation $a \circ b$ means pairwise multiplication of polynomials, or vectors of polynomials, in the NTT domain. For example, if $a = (a_0, a_1)$ and $b = (b_0, b_1)$, $a \circ b = (a_0 b_0, a_1 b_1)$.

Kyber uses a variant of the Fujisaki-Okamoto transform [11] to build an IND-CCA2 secure KEM scheme. This transform applies an additional re-encryption of the decrypted message, using the same randomness as used for the encryp-

tion of the received ciphertext. The decryption is only valid if the re-computed ciphertext matches the received ciphertext.

Algorithm 3 Kyber-CCA2-KEM Encapsulation (simplified)

Input: Public key $pk = (\hat{\mathbf{t}}, \rho)$
Output: Ciphertext c , shared key K

- 1: Choose uniform m
- 2: $(\bar{K}, \tau) \leftarrow \text{Hash}(m \parallel \text{Hash}(pk))$
- 3: $c \leftarrow \text{PKE.Enc}(pk, m, \tau)$
- 4: $K \leftarrow \text{KDF}(\bar{K} \parallel \text{Hash}(c))$
- 5: **return** (c, K)

Algorithm 4 Kyber-CCA2-KEM Decapsulation (simplified)

Input: Secret key $sk = (\hat{\mathbf{s}}, pk, h, z)$, ciphertext $c = (\mathbf{u}, v)$
Output: Shared key K

- 1: $m \leftarrow \text{Decode}(v - \text{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u})))$
- 2: $(K, \tau) \leftarrow \text{Hash}(m \parallel h)$
- 3: $c' \leftarrow \text{PKE.Enc}(pk, m, \tau)$
- 4: **if** $c = c'$ **then**
- 5: **return** $K := \text{KDF}(K \parallel \text{Hash}(c))$
- 6: **else**
- 7: **return** $K := \text{KDF}(z \parallel \text{Hash}(c))$
- 8: **end if**

2.4 Number Theoretic Transform

For lattice-based schemes using polynomial rings, polynomial multiplications in en-/decryption are the most computationally expensive step. The Number Theoretic Transform (NTT) is a technique that can achieve efficient computation for those multiplications.

The NTT is similar to the Discrete Fourier Transform (DFT), but instead of over the field of complex numbers, it operates over a prime field \mathbb{Z}_q . It can be seen as a mapping between the coefficient representation of a polynomial from \mathcal{R}_q (called the normal domain) to the evaluation of the polynomial at the n -th roots of unity (called the NTT domain). This bijective mapping is typically referred to as forward transformation. The mapping from the NTT domain to the normal domain is referred to as backward transformation or inverse NTT. In the NTT domain, the multiplication of polynomials can be achieved by point-wise multiplication, which is much cheaper than multiplication in the normal domain. Typically, one would perform the forward transformation, multiply the

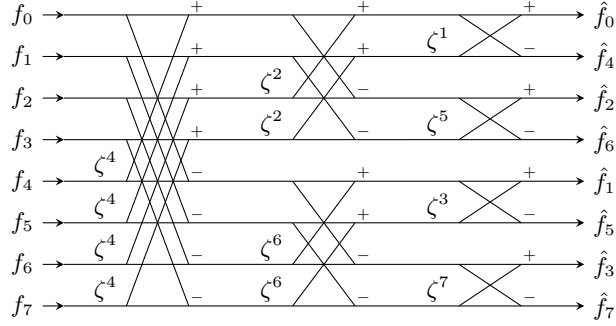


Fig. 1: 8-coefficient Cooley-Tukey decimation in time NTT

polynomials pointwisely in the NTT domain, and go back using the backward transformation. For \mathcal{R}_q with a $2n$ -th primitive root of unity ζ , the NTT transformation of an n -degree polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ is defined as:

$$\hat{f} = \text{NTT}(f) = \sum_{i=0}^{n-1} \hat{f}_i x^i, \quad \text{where} \quad \hat{f}_i = \sum_{j=0}^{n-1} f_j \zeta^{(2i+1) \cdot j}.$$

Similarly,

$$f = \text{NTT}^{-1}(\hat{f}) = \sum_{i=0}^{n-1} f_i x^i, \quad \text{where}$$

$$f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \zeta^{-i \cdot (2j+1)}.$$

The NTT transform and its inverse can be applied efficiently by using a chaining of $\log_2 n$ butterflies. It is a divide and conquer technique that splits the input in half in each step and solves two problems of size $n/2$. The construction for an 8-coefficient NTT using the Cooley-Tukey butterfly [7] with decimation in time is depicted in Figure 1, with the output being in bit-reversed order. Notice that both NTT and inverse NTT are a linear transform, thus they can be expressed by matrix multiplications, e.g. $[f_i]^\top = \mathbf{M}[\hat{f}_i]^\top$ for some $n \times n$ matrix \mathbf{M} .

Kyber uses an NTT-friendly ring. But in Kyber, only n -th primitive roots of unity exist, therefore the modulus polynomial $x^n + 1$ only factors into polynomials of degree 2. Hence, the last layer between nearest neighbors of the NTT is skipped and in NTT domain multiplication is not purely point-wise, but multiplications of polynomials of degree 1. That is, the Kyber ring is effectively $\mathbb{F}_{q^2}[y]/(y^{128} + 1)$, where \mathbb{F}_{q^2} is the field $\mathbb{Z}_q[x]/(x^2 - \zeta)$. Also note that in Kyber, polynomials in the NTT domain are always considered in bit-reversed order (cf. Figure 1). Therefore, in the following bit-reversal is implicitly expected in the NTT domain and indices for NTT-coefficients are noted in regular order.

3 Correlation Power Analysis

In this section, we provide a comprehensive introduction to correlation power analysis (CPA) provided by Mangard *et al.* [18] in Section 3.1, and then we apply the idea to reveal the secret key of Kyber in Section 3.2.

The goal of CPA is to reveal secret keys of cryptographic devices based on a large number of power traces that have been recorded while the devices encrypt or decrypt different plaintexts. The probability of success for CPA depends on the quality and number of traces. Due to the fact that CPA does not require detailed knowledge about the attacked devices, it is the most popular type of power analysis attack. Furthermore, they can reveal the secret key even if the recorded power traces are extremely noisy.

3.1 General Description

We now discuss in detail how such an analysis reveals the secret keys of cryptographic devices in five steps. To reveal one coefficient we need to apply the five steps, however, step 2 can be applied only once and the power consumption can be used multiple time for each coefficient that needs to be recovered.

Step 1: Choosing an Intermediate Result of the Executed Algorithm.

The first step of a CPA is to choose an intermediate result of the cryptographic algorithm that is executed by the device. This intermediate value needs to be a function $f(d, k)$, where d is a known non-constant data value and k is a small part of the key. In most attack scenarios, d is either the plaintext or the ciphertext.

Step2: Measuring the Power Consumption. The second step of a CPA is to measure the power consumption of the device while it encrypts or decrypts D different data blocks. For each of these encryption or decryption runs, the attacker needs to know the corresponding data value d that is involved in the calculation of the intermediate result chosen in Step 1. We denote these known data values by vector $\mathbf{d} = (d_1, \dots, d_D)^\top$, where d_i denotes the data value in the i -th encryption or decryption process.

During each of these runs, the attacker records a power trace. We denote the power trace that corresponds to data block d_i by $\mathbf{t}_i^\top = (t_{i,1}, \dots, t_{i,T})$, where T denotes the length of the trace. The attacker measures a trace for each of the D data blocks, and hence, the traces can be written as matrix \mathbf{T} of size $D \times T$.

It is important that the measured traces are correctly aligned. This means that the power consumption values of each column t_j of the matrix \mathbf{T} need to be caused by the same operation. In practice, attackers typically try to measure only the power consumption that is related to the targeted intermediate result. If the plaintext is known, the attacker sets the trigger of the oscilloscope to the sending of the plaintext from the PC to the cryptographic device and records the power consumption for a short period of time.

Step 3: Calculating Hypothetical Intermediate Values.

The next step of the attack is to calculate a *hypothetical intermediate value* for every possible choice of k . We write these possible choices as vector $\mathbf{k} = (k_1, \dots, k_K)$, where K denotes the total number of possible choices of k . In the context of CPA, we

usually refer to the elements of this vector as key hypotheses. Given the data vector \mathbf{d} and the key hypotheses \mathbf{k} , an attacker can easily calculate hypothetical intermediate values $f(d, k)$ for all D en-/decryption runs and for all K key hypotheses. This calculation results in a matrix \mathbf{V} of size $D \times K$.

$$\mathbf{V} = [f(d_i, k_j)]_{D \times K}$$

A j -th column of \mathbf{V} contains the intermediate results that have been calculated based on the key hypothesis k_j . It is clear that one column of \mathbf{V} contains those intermediate values that have been calculated in the device during the D en-/decryption runs because \mathbf{k} contains all possible choices for k . We refer to the index of this element as ck . Hence, k_{ck} refers to the key of the device. The goal of CPA is to find out which column of \mathbf{V} has been processed during the D en-/decryption runs. We immediately know k_{ck} as soon as we know which column of \mathbf{V} has been processed in the attacked device.

Step 4: Mapping Intermediate Values to Power Consumption Values.

The next step of a CPA is to map the hypothetical intermediate values \mathbf{V} to a matrix \mathbf{H} of *hypothetical power consumption values*. For this purpose, the attacker typically uses models like *Hamming-weight model* or *Hamming-distance model* depending on the scenarios of attack. Using the techniques, the power consumption of the device for each hypothetical intermediate value $v_{i,j}$ is simulated in order to obtain a hypothetical intermediate value $h_{i,j}$.

The quality of the simulation strongly depends on the knowledge of the attacker about the analyzed device. The better the simulation of the attacker matches the actual power consumption characteristics of the device, the more effective the CPA is. The most commonly used power models to map \mathbf{V} to \mathbf{H} are the Hamming-distance and Hamming-weight models.

Step 5: Comparing the Hypothetical Power Consumption Values with the Power Traces.

After having mapped \mathbf{V} to \mathbf{H} , the final step of a CPA can be performed. In this step, each column \mathbf{h}_i of the matrix \mathbf{H} is compared with each column \mathbf{t}_j of the matrix \mathbf{T} . This means that the attacker compares the hypothetical power consumption values of each key hypothesis with the recorded traces at every position. The result of this comparison is a matrix \mathbf{R} of size $K \times T$, where each element $r_{i,j}$ contains the result of the comparison between the columns \mathbf{h}_i and \mathbf{t}_j . The comparison is done based on the Pearson correlation coefficient,

$$r_{i,j} = \frac{\sum_{d=1}^D (h_{d,i} - \bar{h}_i) \cdot (t_{d,j} - \bar{t}_j)}{\sqrt{\sum_{d=1}^D (h_{d,i} - \bar{h}_i)^2 \cdot \sum_{d=1}^D (t_{d,j} - \bar{t}_j)^2}}$$

where \bar{h}_i and \bar{t}_j denote the mean values of the columns \mathbf{h}_i and \mathbf{t}_j . It has the property that the value $r_{i,j}$ is the higher, the better columns \mathbf{h}_i and \mathbf{t}_j match. The key of the attacked device can hence be revealed based on the following observation.

The power traces correspond to the power consumption of the device while it executes a cryptographic algorithm using different data inputs. The intermediate result that has been chosen in step 1 is a part of this algorithm. Hence, the device

needs to calculate the intermediate value \mathbf{v}_{ck} during the different executions of the algorithm. Consequently, also the recorded traces depend on these intermediate values at some position. We refer to this position of the power traces as ct , i.e., the column \mathbf{t}_{ct} contains the power consumption values that depend on the intermediate value \mathbf{v}_{ck} .

The hypothetical power consumption values \mathbf{h}_{ck} have been simulated by the attacker based on the values \mathbf{v}_{ck} . Therefore, the columns \mathbf{h}_{ck} and \mathbf{t}_{ct} are strongly related. In fact, these two columns lead to the highest value in \mathbf{R} , i.e., the highest value of the matrix \mathbf{R} is the value $r_{ck,ct}$. An attacker can hence reveal the index for the correct key ck and the moment of time ct by simply looking for the highest value in the matrix \mathbf{R} . The indices of this value are then the result of the CPA.

Sometimes, CPA produce high correlation coefficients for many key hypotheses at the time when targeted intermediate result is processed. The high correlation peaks for wrong keys are sometimes referred to as ghost peaks. These peaks happen because the hypothetical intermediate values are correlated. The height of these correlations depends on the intermediate result that is attacked.

3.2 Application on CRYSTALS-Kyber

Our attack targets the decryption process of Kyber, i.e. line 1 of Algorithm 4, with the aim of recovering the victim’s secret key $\hat{\mathbf{s}}$. To decrypt a message the recipient calculates $\text{NTT}^{-1}(\hat{\mathbf{s}}^\top \circ \hat{\mathbf{u}})$, where $\hat{\mathbf{u}}$ is the decompressed ciphertext in the NTT domain and \circ denotes the pairwise multiplication. The pairwise multiplication is done in the quotient ring $\mathbb{Z}_q[x]/(x^2 - \zeta_i)$ as we discussed in Section 2.4, where ζ_i are the primitive roots of unity of \mathbb{Z}_q . In such a ring, the product of two polynomials $a = a_0 + a_1x$ and $b = b_0 + b_1x$ can be easily computed as

$$ab = (a_0b_0 + a_1b_1\zeta_i) + (a_0b_1 + a_1b_0)x \pmod q.$$

However, in most of the processors, modular multiplication is still expensive since it needs divisions by q . Fortunately, we can avoid the divisions by the Montgomery reduction algorithm summarized in Algorithm 5. By setting $R = 2^{16}$, division by R can be replaced by a simple bit shifting and $x \pmod R$ can be done by returning the lower 16 bits of x , which results in an integer between $-R/2$ and $R/2 - 1$. The algorithm works because first, t is chosen so that $a - tq$ is divisible by R . Second, t is in the range $[-R/2, R/2 - 1]$, thus $a - tq$ is in the range $[-qR + q, qR - 1]$, which guarantees that b is in the correct range.

Let x_0 and y_0 be two integers in the range $[-q+1, q-1]$, we refer to the result of Montgomery reduction of $x_0 \times y_0$ by Algorithm 5 as $\text{fcmul}(x_0, y_0)$. Then the

Algorithm 5 Montgomery reduction

Input: Integers q, R with $\gcd(q, R) = 1$
Integer $q^{-1} \in [-R/2, R/2 - 1]$ such that $qq^{-1} \equiv 1 \pmod R$
Integer $a \in [-qR/2, qR/2 - 1]$

Output: Integer $b \in [-q + 1, q - 1]$ such that $b \equiv aR^{-1} \pmod q$

- 1: $t \leftarrow ((a \bmod R)q^{-1}) \bmod R$
 - 2: $b \leftarrow (a - tq)/R$
 - 3: **return** b
-

product $r_0 + r_1x = ab2^{-16}$ can be computed as follow:

$$\begin{aligned}
r_0 &\leftarrow \mathbf{fqmul}(a_1, b_1) \\
r_0 &\leftarrow \mathbf{fqmul}(r_0, \zeta_i 2^{16}) \\
r_0 &\leftarrow \mathbf{fqmul}(a_0, b_0) + r_0 \\
r_1 &\leftarrow \mathbf{fqmul}(a_1, b_0) \\
r_1 &\leftarrow \mathbf{fqmul}(a_0, b_1) + r_1.
\end{aligned} \tag{1}$$

The unwanted constant can be dealt within the inverse NTT together when we divide the coefficient by n , thus no extra multiplications is needed.

Now suppose we want to reveal the coefficients $(\hat{s}_{2i}, \hat{s}_{2i+1})$, notice that they are point-wisely multiplied by the ciphertext $(\hat{u}_{2i}, \hat{u}_{2i+1})$, then our first chosen intermediate value is $\mathbf{fqmul}(\hat{s}_{2i+1}, \hat{u}_{2i+1})$, i.e. r_0 in the first line of equation (1). The intermediate value meets the requisite described in Section 3.1, and the total number of possible choices of $\hat{s}_{2i+1} \in [0, q - 1]$ is q . Following the steps in Section 3.1, we can get a list of the most possible candidates of \hat{s}_{2i+1} . There can be some incorrect candidates with high score in this step, for example, $q - \hat{s}_{2i+1}$ can be such a candidate since the Hamming weight of $\mathbf{fqmul}(q - \hat{s}_{2i+1}, \hat{u}_{2i+1})$ is strongly correlated with $\mathbf{fqmul}(\hat{s}_{2i+1}, \hat{u}_{2i+1})$.

Now that we have some highly confident candidates for \hat{s}_{2i+1} , we can then use it and newly guessed \hat{s}_{2i} to calculate the hypothetical value of r_1 . And we can repeat the same process except that the intermediate values are now $\mathbf{fqmul}(\hat{s}_{2i}, \hat{u}_{2i+1}) + \mathbf{fqmul}(\hat{s}_{2i+1}, \hat{u}_{2i})$, i.e. r_1 in the last line of equation 1. Following the same steps, we can find the candidate with the highest correlation coefficient, and if it is higher than some threshold, we accept the guess. If not, we try the next candidate of \hat{s}_{2i+1} . If there is no candidate with high enough correlation coefficient, we just return failure. Then we guess the next one with same process targeting the next intermediate values.

The complexity can be easily calculated, if K is the number of possible keys, T is the scanned window size, D is the number of power traces, then we need TK computations of correlation coefficient of length D vectors to recover one coefficient of the secret key, which is linear to all the parameters. For Kyber512, we need to repeat the process above 256 times to recover the 512 coefficients in the NTT domain. The CPA process is identical across different parameter sets of Kyber, thus it is easy to adapt to Kyber768/1024 without any problem. It can also be parallelized as long as we know the starting point of each \mathbf{fqmul} in

the power trace, since the length of all power traces is the same, we only need to evaluate the starting point once and store the result. For Kyber512 on a 16 core computer, our CPA can scan through all coefficients within 5 minutes.

However, we will run into some problems. If the correct coefficient $(\hat{s}_{2i}, \hat{s}_{2i+1})$ has high score, then it is likely that $(q - \hat{s}_{2i}, q - \hat{s}_{2i+1})$ has high score too, since the Hamming weight of them are highly correlated. So to prevent it from getting accepted, we can increase the threshold for acceptance, however, it may cause the correct ones to get rejected too. Furthermore, in some rare cases, $(q - \hat{s}_{2i}, q - \hat{s}_{2i+1})$ may have a higher score than the correct one and be accepted, we call such cases false positive. The way we deal with it is to sample the accepted guesses and hope the coefficients we sampled are all correct ones. The number of sampled coefficients will be further discussed in Section 4.

4 Lattice Attack

In this section, we describe how to construct a simpler LWE problem from the coefficients that have been recovered in the CPA attack, then we do a hardness analysis that determines the least number of coefficients needed to be recovered in the CPA.

4.1 Lattice Construction

Now we have some of the coefficients being recovered, the next step is to recover the unknown coefficients by the lattice attack. Because of the structure of incomplete NTT in Kyber, we know that coefficients are split into $2k$ groups of 128 ones. We will focus on one group and notice that the rest of the steps need to repeat $2k$ times to derive the full secret key.

Let $\mathbf{M} = [\mathbf{m}_0, \mathbf{m}_2, \dots, \mathbf{m}_{254}]$ be the inverse NTT matrix as we mentioned in Section 2.4. Suppose we have recovered $128 - \ell$ coefficients in $\hat{\mathbf{s}}_i$, one of the groups in $\hat{\mathbf{s}}$, from the polynomial multiplication $\hat{\mathbf{s}} \circ \hat{\mathbf{u}}$, i.e., we need to recover the remaining ℓ coefficients. Let $A = \{a_0, a_1, \dots, a_{127-\ell}\}$ be the indices that are successfully recovered in the CPA step, and $B = \{b_0, b_1, \dots, b_{\ell-1}\}$ be the indices that are still unknown, then the inverse NTT $\text{NTT}^{-1}(\hat{\mathbf{s}}_i) = \mathbf{M}\hat{\mathbf{s}}_i = \mathbf{s}_i \pmod q$ can be split into two halves as followed:

$$\mathbf{M}_A \hat{\mathbf{s}}_{i,A} + \mathbf{M}_B \hat{\mathbf{s}}_{i,B} = \mathbf{s}_i \pmod q,$$

where $\mathbf{M}_A := [\mathbf{m}_{a_0}, \dots, \mathbf{m}_{a_{127-\ell}}]$ is a matrix whose columns are those of \mathbf{M} whose indices are in A , $\hat{\mathbf{s}}_{i,A} = [\hat{s}_{a_0}, \dots, \hat{s}_{a_{127-\ell}}]^\top$, and the similar definition for \mathbf{M}_B and $\hat{\mathbf{s}}_{i,B}$.

Notice that \mathbf{s}_i is an extremely short vector since it is the secret key sampled from β_η . By calling the known vector $\mathbf{t} = \mathbf{M}_A \hat{\mathbf{s}}_{i,A}$, the known basis $\mathbf{A} = -\mathbf{M}_B$, and an unknown vector $\mathbf{s}'_i = \hat{\mathbf{s}}_{i,B}$, we now have $\mathbf{t} = \mathbf{A}\mathbf{s}'_i + \mathbf{s}_i \pmod q$, which is exactly the definition of an LWE problem. Compared to the original module-LWE problem in Kyber, this problem becomes simpler since the rank of \mathbf{A} is less than the original one.

4.2 Hardness Analysis

We use the standard technique of Kannan’s embedding to solve the LWE problem. First we treat the LWE problem as a BDD/uSVP problem and then apply a lattice reduction algorithm. For example, given the instance above ($\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s}'_i + \mathbf{s} \pmod{q}$), consider the lattice $\Lambda(\mathbf{B}_{BDD})$ generated by

$$\mathbf{B}_{BDD} = \begin{bmatrix} \mathbf{I}_\ell & \mathbf{A}' \\ \mathbf{0} & q\mathbf{I}_{n-\ell} \end{bmatrix},$$

where $[\mathbf{I}_\ell \mid \mathbf{A}']$ denotes the reduced row echelon matrix of \mathbf{A}^\top , which can be easily calculated by Gaussian elimination. We can then solve the BDD of $\Lambda(\mathbf{B}_{BDD})$ with respect to the target point \mathbf{t} which reveals \mathbf{s}' and \mathbf{s} .

Alternatively, we can reduce this BDD to uSVP by a technique called Kannan’s embedding [14]. Given the BDD instance above, we consider the following basis matrix

$$\mathbf{B}_{Kan} = \left[\begin{array}{cc|c} \mathbf{I}_\ell & \mathbf{A}' & \mathbf{0} \\ \mathbf{0} & q\mathbf{I}_{n-\ell} & \mathbf{t}^\top \\ \hline & & 1 \end{array} \right].$$

Recall that the lattice $\Lambda(\mathbf{B}_{Kan})$ contains all linear combinations of the vectors in \mathbf{B}_{Kan} . The equation $\mathbf{t} = \mathbf{A}\mathbf{s}'_i + \mathbf{s}_i \pmod{q}$ can be written as $\mathbf{t} = \mathbf{A}\mathbf{s}'_i + \mathbf{s}_i + q\mathbf{k}$, where $\mathbf{k} \in \mathbb{Z}_q^n$, so there exists a row vector $[-\mathbf{s}''^\top \mid -\mathbf{k}'^\top \mid 1] \in \mathbb{Z}_q^{n+1}$ such that the shortest vector in $\Lambda(\mathbf{B}_{Kan})$ is $[-\mathbf{s}''^\top \mid -\mathbf{k}'^\top \mid 1] \cdot \mathbf{B}_{Kan} = [\mathbf{s}_i^\top \mid 1] \in \mathbb{Z}_q^{n+1}$.

The norm of vector $[\mathbf{s}_i^\top \mid 1]$ is $\sqrt{\|\mathbf{s}_i\|^2 + 1} \approx \sqrt{n}\sigma_s$. If this norm is smaller than the norm of the shortest vector estimated by the Gaussian Heuristic, this uSVP instance can be solved, and the more gap between the first and second successive minima, i.e., the bigger $\lambda_2(\Lambda(\mathbf{B}_{Kan}))/\lambda_1(\Lambda(\mathbf{B}_{Kan}))$ is, the easier the uSVP will be. Since the volume of the lattice $\Lambda(\mathbf{B}_{Kan})$ is $q^{n-\ell}$, $\lambda_2(\Lambda(\mathbf{B}_{Kan}))$ can be estimated by

$$\lambda_2(\Lambda(\mathbf{B}_{Kan})) \approx \sqrt{\frac{n+1}{2\pi e}} q^{(n-\ell)/(n+1)}.$$

To determine the least number of coefficients we must recover in the CPA step, we do an experiment on solving the SVP randomly generated by script. The result is shown in Fig. 2, where the blue line is the success rate of finding $[\mathbf{s}_i^\top \mid 1]$ by the BKZ algorithm¹ of block size 50 for 20 randomly generated \mathbf{s} , and the red line is the running time of the algorithm. From the result, the critical point of guaranteed success is on $\ell = 89, \ell = 90$ for Kyber512, Kyber768/1024, respectively. This means that in the CPA step, we need at least $128 - 89 = 39$ (or 38 for Kyber768/1024) recovered coefficients so that we can have a fully recovered secret key when using the BKZ algorithm of block size 50 to solve the reduced SVP problem. Notice that in order to do a full key recovery, the number of recovered coefficients need to be multiplied by $2k$, where k is the module dimension for each version of Kyber. The reason that Kyber768/1024 is easier to solve is because η of Kyber768/1024 is smaller than that of Kyber512.

¹ We ran the experiment using the BKZ implementation from `fpyl11` in Sage9.2. See <https://github.com/fplll/fpylll>

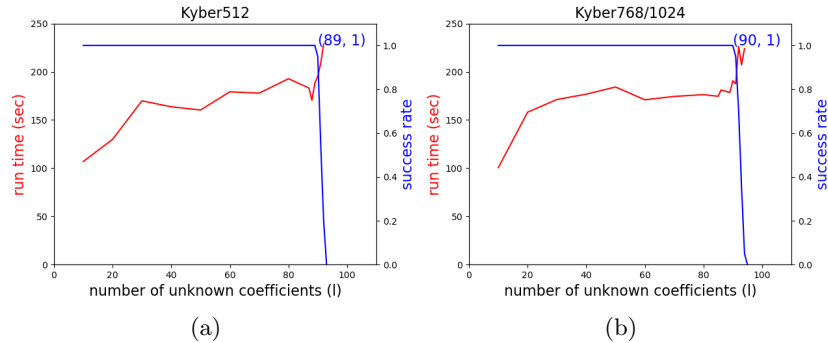


Fig. 2: Success rate and running time on randomly generated uSVP in the lattice \mathbf{B}_{Kan} for (a) Kyber512 and (b) Kyber768/1024

5 Experiments

We experimented our attacks on simulated power traces of the ARM cortex-M0 processor, then estimate how many traces we need to conduct our attack.

5.1 ELMO

Our simulated traces were generated using the ELMO [10], which emulates the power consumption of an ARM Cortex M0 processor and produces noise-free traces. The tool reproduces the 3-stage pipeline of an M0 processor, which means that the algorithmic noise is taken into account. ELMOs quality has been established by comparing leakage detection results between simulated and real traces from a STM32F0 Discovery Board [20]. For reference, to conduct a successful key recovery power analysis on the lattice-based signature scheme FALCON, the required numbers of simulated power traces and real acquisitions are 2000 and 5000 [12].

5.2 Results

Table 2 gives the results of our experiment done on the simulated traces. The *threshold* is the minimum correlation coefficient of acceptance that we set as a parameter in Section 3.2. *Recovered rate* is the average number of successfully recovered coefficients, and *false positive* is the average number of coefficients that are accepted but turn out to be wrong. The *success rate* is the possibility of all 39/38 coefficients we randomly sample being the correct ones when we choose from all coefficients that are accepted by the CPA step, which can be directly calculated by $\binom{a-39}{b} / \binom{a}{b}$ if a is the *recovered rate* and b is the *false positive*. Therefore, it does not mean the overall success rate of our attack, the overall success rate will be arbitrarily closed to 1 if we keep sampling the coefficients as long as we have at least 39/38 correct ones.

Table 2: Experimental results on different acceptance threshold and trace number. Left hand side of success rate is for Kyber512 and right is for Kyber768/1024.

Threshold	Trace number	Recovered rate	False positive	Success rate
0.63	200	110.5/128	6/128	0.07(0.07)
	400	118.75/128	4.25/128	0.18(0.19)
	600	124.75/128	3/128	0.32(0.33)
	800	124.75/128	1.75/128	0.52(0.54)
0.65	200	98.75/128	4.5/128	0.10(0.11)
	400	109/128	4.25/128	0.15(0.16)
	600	112/128	2.25/128	0.39(0.40)
	800	116.25/128	1.5/128	0.55(0.56)
0.67	200	79.25/128	2.5/128	0.19(0.20)
	400	86/128	0.5/128	0.77(0.78)
	600	83.75/128	0.25/128	0.88(0.89)
	800	86.5/128	0/128	1(1)
0.69	200	58/128	1/128	0.33(0.34)
	400	53.25/128	0.25/128	0.82(0.82)
	600	49.75/128	0/128	1(1)
	800	49.5/128	0/128	1(1)

It can be seen that although adding trace numbers does not help much to increase the recovered coefficients, it does help to lower the false positive, which directly affects the success rate. Increasing the threshold of acceptance will also lower the false positive and recovered rate, but notice that if the recovered rate drops below 39, our attack may fail. Since the running time of the overall attack is dominated by CPA, we would argue that the fewer the number of power traces the better it is, as long as the success rate is higher than 0.05.

5.3 Application to Saber

Saber [9] is a lattice-based key encapsulation mechanism based on the Module Learning With Rounding problem. Saber is one of the round 3 candidates of the NIST post-quantum cryptography standardization competition. The polynomial ring used within Saber is $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ with $q = 2^{13}$ and $n = 256$ across all parameter sets. Saber also offers three security levels: Lightsaber with security level similar to AES-128, Saber with one similar to AES-192 and Firesaber with one similar to AES-256.

Because Saber was not specifically designed to benefit from NTT-based multiplication by using an *NTT-friendly* ring, it uses a combination of Toom-4 and Karatsuba to implement efficient polynomial arithmetic. However, as shown in the work by [6], NTTs can be used to obtain efficient polynomial arithmetic in finite fields modulo a power-of-two. They did this by choosing a prime $p > nq^2/2$ such that $n|(p-1)$, computing the multiplication by the NTT over

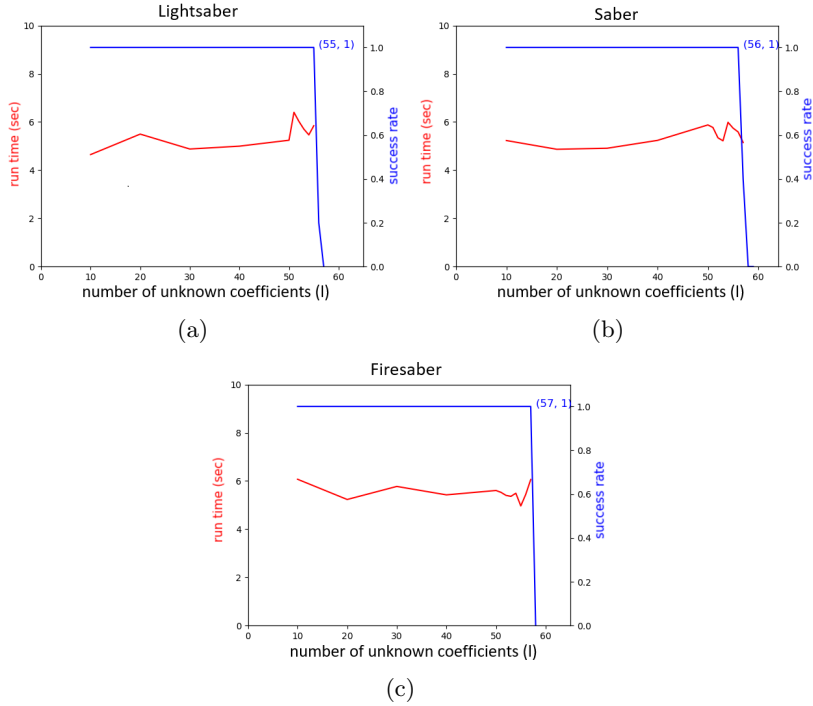


Fig. 3: Success rate and running time on randomly generated USVP for (a) Lightsaber, (b) Saber and (c) Firesaber

$\mathbb{Z}_p[x]$, and then reducing the result back to $\mathbb{Z}_q[x]$. Since the modulus is much bigger in the NTT for Saber, the SCA for pointwise multiplication on Saber needs to target a smaller portion of the intermediate value, which results in smaller signal-to-noise ratio. In [21], a minimum of 10000 traces was required to mount a successful attack.

Figure 3 shows our lattice attack when applying to the SCA proposed by [21]. Since the implementation uses 6 layers of NTTs, we divide the coefficients into $512/2^6 = 8$ groups and find the minimum number of coefficients we needed to recover other one. We can see that it needs 9/8/7 coefficients out of 64 to guarantee a successful attack for each parameter sets of Saber, which means a total of 72/64/56 coefficients are needed. This saves about 86% ~ 89% of the running time for the SCA. Another way to see the improvement is the possibility to reduce the traces of SCA. Although by doing so, there may be incorrectly recovered coefficients, by our sampling approach as shown before, we only need portion of the coefficients correct to recover the whole secret key. We do want to point out that the improvement heavily depends on the implementation of the incomplete NTT of choice. That is, the less layers of incomplete NTTs an implementation chooses, the less coefficients we need to perform the lattice attack.

6 Conclusion

In this paper, we propose a combined CPA and lattice attack on Kyber. With 200 traces, our attack terminated within 20 minutes on a 16-core computer. Compared to other SCA targeting NTT in the cryptosystems, our attack achieves lower runtime in practice. Furthermore, there is potential for decreasing the number of traces by using lattice reduction if the same measurement is used.

Our future works are to migrate the attacks to real devices and other cryptosystems using the NTT transform multiplication like Saber or NTRU. We can also investigate the effect of popular countermeasures of CPA like masking and hiding on our attack.

Acknowledgement. This work was partially supported by JSPS KAKENHI Grant Number 19K20267, Japan, and JST CREST Grant Number JPMJCR2113, Japan.

References

1. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber (version 3.02) submission to round 3 of the NIST post-quantum project. Specification document (2021)
2. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367 (2018). <https://doi.org/10.1109/EuroSP.2018.00032>
3. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., Stehlé, D.: Kyber. <https://github.com/pq-crystals/kyber> (2023)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ITCS '12, Association for Computing Machinery (2012). <https://doi.org/10.1145/2090236.2090262>, <https://doi.org/10.1145/2090236.2090262>
5. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011. pp. 1–20 (2011)
6. Chung, C.M.M., Hwang, V., Kannwischer, M.J., Seiler, G., Shih, C.J., Yang, B.Y.: NTT multiplication for NTT-unfriendly rings. Cryptology ePrint Archive, Paper 2020/1397 (2020), <https://eprint.iacr.org/2020/1397>, <https://eprint.iacr.org/2020/1397>
7. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation* **19**, 297–301 (1965)
8. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Advances in Cryptology CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Proceedings, Part II. pp. 329–358 (2020). https://doi.org/10.1007/978-3-030-56880-1_12, https://doi.org/10.1007/978-3-030-56880-1_12
9. D’Anvers, J.P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology – AFRICACRYPT 2018. pp. 282–305 (2018)

10. ELMO: Evaluating leaks for the arm cortex-m0. <https://github.com/sca-research/ELMO>, accessed: 2022-10-17
11. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: *Advances in Cryptology - CRYPTO '99*, 19th Annual International Cryptology Conference, Proceedings. pp. 537–554 (1999). https://doi.org/10.1007/3-540-48405-1_34
12. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M.: The hidden parallelepiped is back again: Power analysis attacks on Falcon. *Cryptology ePrint Archive*, Paper 2022/057 (2022), <https://eprint.iacr.org/2022/057>, <https://eprint.iacr.org/2022/057>
13. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2021**, Issue 4, 88–113 (2021). <https://doi.org/10.46586/tches.v2021.i4.88-113>, <https://tches.iacr.org/index.php/TCHES/article/view/9061>
14. Kannan, R.: Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research* **12**(3), 415–440 (1987), <http://www.jstor.org/stable/3689974>
15. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) *Advances in Cryptology — CRYPTO' 99*. pp. 388–397 (1999)
16. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: *Advances in Cryptology - CRYPTO '96*, 16th Annual International Cryptology Conference, Proceedings. pp. 104–113 (1996). https://doi.org/10.1007/3-540-68697-5_9
17. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. *J. ACM* **60**(6) (2013). <https://doi.org/10.1145/2535925>, <https://doi.org/10.1145/2535925>
18. Mangard, S., Oswald, E., Popp, T.: *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer Publishing Company, Incorporated, 1st edn. (2010)
19. May, A., Nowakowski, J.: Too many hints - when LLL breaks LWE. *Cryptology ePrint Archive*, Paper 2023/777 (2023), <https://eprint.iacr.org/2023/777>, <https://eprint.iacr.org/2023/777>
20. McCann, D., Oswald, E., Whitnall, C.: Towards practical tools for side channel aware software engineering: Grey box' modelling for instruction leakages. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. pp. 199–216 (2017)
21. Mujdei, C., Wouters, L., Karmakar, A., Beckers, A., Mera, J.M.B., Verbauwhede, I.: Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Trans. Embed. Comput. Syst.* (2022). <https://doi.org/10.1145/3569420>, <https://doi.org/10.1145/3569420>
22. National Institute of Standards and Technology. Post-quantum cryptography standardization. <https://csrc.nist.gov/projects/post-quantum-cryptography>, accessed: 2022-10-12
23. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. *Cryptology ePrint Archive*, Paper 2019/795 (2019), <https://eprint.iacr.org/2019/795>, <https://eprint.iacr.org/2019/795>
24. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. *Cryptology ePrint Archive*, Paper 2017/594 (2017), <https://eprint.iacr.org/2017/594>, <https://eprint.iacr.org/2017/594>

25. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing. vol. 56, pp. 84–93 (2005). <https://doi.org/10.1145/1568318.1568324>
26. Shor, P.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). <https://doi.org/10.1109/SFCS.1994.365700>