STAMP-Single Trace Attack on M-LWE Pointwise Multiplication in Kyber

Bolin Yang[1], Prasanna Ravi[2,3], Fan Zhang[1], Ao Shen[1] and Shivam Bhasin[2,3]

[1] Zhejiang University, Hangzhou, China
[2] Temasek Laboratories, Nanyang Technological University, Singapore
[3] School of Computer Science and Engineering, Nanyang Technological University, Singapore

yangbolin@zju.edu.cn prasanna.ravi@ntu.edu.sg fanzhang@zju.edu.cn
aoshen@zju.edu.cn sbhasin@ntu.edu.sg

Abstract.
In this work, we propose a novel single-trace key recovery attack targeting side-channel leakage from the key-generation procedure of Kyber KEM. Our attack exploits the inherent nature of the Module-Learning With Errors (Module-LWE) problem used in Kyber KEM. We demonstrate that the inherent reliance of Kyber KEM on the Module-LWE problem results in higher number of repeated computations with the secret key, compared to the Ring-LWE problem of similar security level. We exploit leakage from the pointwise multiplication operation in the key-generation procedure, and take advantage of the properties of the Module-LWE instance to enable a potential single trace key recovery attack. We validated the efficacy of our attack on both simulated and real traces, and we performed experiments on both the reference and assembly optimized implementation of Kyber KEM, taken from the *pqm4* library, a well-known benchmarking and testing framework for PQC schemes on the ARM Cortex-M4 microcontroller. We also analyze the applicability of our attack on the countermeasures against traditional SCA such as masking and shuffling. We believe our work motivates more research towards SCA resistant implementation of key-generation procedure for Kyber KEM.

Keywords: Kyber · Side-Channel Attack · Single-Trace Attack · Post Quantum Cryptography

1 Introduction

The NIST Post-Quantum Cryptography (PQC) standardization process for *post-quantum cryptography* completed its third round in July 2022, when it announced the first list of algorithms for Public Key Encryptoin (PKE), Key Encapsulation Mechanisms (KEMs) and Digital Signatures schemes (DSS) that are intended to be standardized [NIS16]. Among the several categories of PQC schemes that were considered in the NIST PQC standardization, schemes from lattice-based cryptography dominated the field owing to its fine balance of security and efficiency guarantees. In particular, NIST selected two schemes - Kyber KEM [SAB+22] and Dilithium DSS [LDK+22] which are based on the well-known Module Learning With Error (MLWE) problem.

The standardization of Kyber and Dilithium in particular, will ensure that they will be implemented on a wide-range of computational platforms, starting from the low-end microcontrollers, FPGAs all the way until general purpose PCs and workstations. More importantly, we will also witness their rapid adoption in embedded devices which naturally makes them susceptible to Side-Channel Attacks (SCA). Resistance of PQC

implementations against SCA was also an important point of consideration during the NIST PQC process [NIS16], and this resulted in a large body of work that studied the susceptibility of PQC implementations on embedded devices, to SCA.

In particular, Kyber KEM has been subjected to a wide range of SCA, with most attacks targeting leakage from the decapsulation procedure, to recover its long term secret key. Such key recovery attacks can be split into two categories - (1) Single trace attacks [PPM17] [PP19] [HHP+21] [LZH+22] and (2) Multi trace attacks [MWK+22] [YWY+23]. Multi trace attacks targeting the decapsulation procedure are applicable when Kyber is used in a static-key setting. Multi-trace attacks are easier to mount than single-trace attacks since they work by recovering incremental information about the secret key over multiple executions. On the other hand, single trace attacks attempt to recover the secret key of the size of a few thousand bits, in a single trace, by combining as much information as possible from a single trace. Moreover, single trace attacks are much more susceptible to noise and jitter in the side-channel measurements compared to multi-trace attacks.

Thus, it is natural for a designer to consider using Kyber in an ephemeral key setting, where the secret key is refreshed for every key exchange. This requires to execute both the key-generation procedure and decapsulation procedure on the target device, for every key exchange. Thus, using Kyber KEM in an ephemeral setting can serve as an attractive alternative for a designer, when SCA is considered to be a realistic threat, given that generic side-channel protection, such as masking, incurs an almost 2-3x overhead in runtime for Kyber KEM [BGR+21] [HKL+22].

Using Kyber KEM in an ephemeral setting only makes it susceptible to single-trace attacks, since the secret key is only manipulated for a single execution. However, the designer also needs to consider protecting the key-generation procedure against single-trace attacks, as it needs to be executed for every key-exchange on the target device. In this respect, the Number Theoretic Transform (NTT) used for polynomial multiplication within Kyber KEM has been targeted by several single trace attacks like [PP19]. These attacks are also applicable to the key-generation procedure, as the NTT is also applied over the secret key polynomial to generate the public key. However, to the best of our knowledge, we are not aware of any other attacks that are applicable specifically to the other parts in decapsulation procedure. Thus, it is possible for a designer to contemplate protecting only the NTT operation using dedicated shuffling and masking countermeasures against single trace attacks [RPBC20]. This begets the question if "solely protecting the NTT operation in the key-generation procedure suffices to provide concrete protection against single-trace attacks. Are there any other vulnerabilities specifically in the key-generation procedure that make them susceptible to single-trace attacks?"

In this work, we answer this question positively by assessing the possibility of single trace attacks, targeting the pointwise multiplication operation in the key-generation procedure of Kyber KEM. We demonstrate that the inherent reliance of Kyber KEM on the Module-LWE problem results in a higher number of repeated computations with the secret key, compared to the Ring-LWE problem of the similar security level. We exploit leakage from the pointwise multiplication operation in the key-generation procedure, and take advantage of the properties of the Module-LWE instance to enable potential single trace key recovery attacks. Thus, our work demonstrates an additional side-channel vulnerability in the key-generation procedure of Kyber KEM which should also be protected against single-trace attacks, along with the NTT operation. Therefore, we believe our work motivates more research towards SCA resistant implementation of key-generation procedure for Kyber KEM. The contributions of our work are manifold:

Contribution

1. We propose a novel single-trace attack targeting the pointwise multiplication operation within the key-generation procedure of Kyber KEM. The use of the Module-

LWE problem for Kyber KEM ensures that the pointwise multiplication operation is computed between a public matrix of dimension $(k \times k)$, and the secret vector of polynomials with dimension k. While the pointwise multiplication operation between vectors of size k in the decapsulation procedure has been targeted by SCA in [MWK+22], their attack required 200 traces for key recovery. However, our attack exploits the polynomial multiplication between a public matrix and private vector dimension k $(k > 1)$, which results in multiple repeated computations manipulating the secret key, whose side-channel leakage provides significantly more information for potential single trace key recovery attacks.

2. We validated the efficacy of our attack on both simulated and real traces, and we performed experiments on both the reference and assembly optimized implementation of Kyber KEM, taken from the *pqm4* library, a well-known benchmarking and testing framework for PQC schemes on the ARM Cortex-M4 microcontroller. The offline searching space of our single trace attack will be around 2^{21} under specific situation. We also demonstrate that key recovery can be significantly accelerated by implementing the post-processing of the side-channel information on a GPU based machine, that exploits the parallelization property of the polynomial multiplication operation in Kyber KEM. The running time of the analysis phase is improved from several minutes to only a few seconds.

3. We also analyze the applicability of our attack on the countermeasures against traditional SCA. While we show that generalized masking can not be used to defend against our attack, but shuffling the pointwise multiplication operation serves as a concrete countermeasure against our attack.

4. Our work demonstrates an additional side-channel vulnerability in the key-generation procedure of Kyber KEM, which should also be protected against single-trace attacks, along with the NTT operation. Therefore, we believe our work motivates more research towards SCA resistant implementation of key-generation procedure for Kyber KEM.

Organization of the Paper

In Section. 2, the notations used in this paper, the introduction of LWE, Kyber and prior works are provided. In Section. 3, we briefly introduce the template attack, and evaluate the target of our template attack on Kyber. In Section. 4, we describe our key enumeration method to locate the correct candidate after template attack. In Section. 5, we propose an accelerated implementation of our attack using CUDA toolkit and GPU. In Section. 6, we discuss the applicability of our attack to implementations protected with countermeasures such as masking and shuffling. Conclusion and future works are listed in Section. 7.

2 Preliminaries

2.1 Notation

In this paper, the polynomial ring $\mathbb{Z}_q[x]/\phi(x)$ is denoted as R_q where $\phi(x) = x^n + 1$. The variables and functions are written in italic style. One polynomial vector is denoted with lowercase bold letters as $\boldsymbol{a}$. The polynomial matrix is denoted with italic uppercase bold letters as $\boldsymbol{A}$, with one polynomial in i-th rows and j-th columns in matrix denoted as $A_{i,j}$. The polynomial in R_q is denoted with regular letters as a, and one coefficient of this polynomial can be denoted as $a[i]$. $\boldsymbol{A} \circ \boldsymbol{s}$ denotes that one polynomial matrix multiplies one polynomial vector.

2.2 M-LWE Problem

The lattice-based problems include The Closest Vector Problem (**CVP**), the Shortest Vector Problem (**SVP**), Bounded-Distance Decoding Problem (**BDD**) and Learning with Error Problem (**LWE**), etc. Among those problems, the LWE problem is frequently used for constructing cryptographic schemes. In a basic LWE problem, a matrix $\boldsymbol{A}$ and vector $\boldsymbol{s}$, $\boldsymbol{e}$, with coefficients sampled uniformly at random in $\mathbb{Z}_q^{m\times n}$ and $\mathbb{Z}_q^n$ respectively, are multiplied as $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}\ (mod\, q)$, where $\boldsymbol{s}$, $\boldsymbol{e}$ are small and unknown vectors. The security of LWE problem is based on the difficulty of recovering the secret vector $\boldsymbol{s}$ from $\boldsymbol{A}$ and $\boldsymbol{b}$ in large dimension. While the operations of matrices require large memory resources with the large dimension for security necessity.

Therefore, the structured LWE problems (e.g. Ring-LWE, Module-LWE) were proposed for better performance within the cryptographic algorithms. In structured LWE problems, the basic element is the polynomial ring as $R_q = \mathbb{Z}_q[x]/\phi(x)$. In Ring-LWE, one polynomial is rotated and extended as a matrix $\boldsymbol{A}$, i.e. every column of the matrix is rotated from one polynomial. The computation form of R-LWE is the same as LWE that $\boldsymbol{b} = \boldsymbol{A} \circ \boldsymbol{s} + \boldsymbol{e}$. Ring-LWE offers key size reduction compared to standard LWE with just two polynomials as the public key, rather than one matrix and a vector. However, the dimension of the polynomial will increase rapidly when higher security level is required, which means the implementation of polynomial multiplication need to be re-organized every time for different n. And the additional structure of the elements may also leak more information of the lattice. So the structured LWE problem needs to be further optimized.

Then the Module-LWE (M-LWE) problem was introduced. In M-LWE problem, $\boldsymbol{s}$, $\boldsymbol{e}$ are small polynomial vectors sampled from R_q^k with small coefficients, where k indicates k polynomials in one vector. The polynomial matrix $\boldsymbol{A}$ is uniformly sampled from $R_q^{k\times k}$. The dimension of one polynomial in M-LWE is often fixed as n. So the multiplication between two polynomials can be accelerated by a specific structure, for example, NTT. The security level of the M-LWE problem can be improved simply by increasing value of k, rather than larger n, while in this paper, we can prove that larger k may lead to more side channel leakages.

2.3 Kyber

Kyber, an IND-CCA2-secure key encapsulation mechanism (KEM) based on the M-LWE problem, is the only candidate in the finalist of the Public-key Encryption and Key-establishment Algorithms, proposed by NIST after three rounds competition.

The basic version of Kyber is an IND-CPA-secure public key encryption (PKE) scheme, which can encrypt messages with a fixed length. Using Fujisaki-Okamoto (FO) transform, the Kyber.PKE scheme can be transformed into Kyber.KEM. The KEM can be used to establish a session key between two parties.

As mentioned before, the basic elements in Kyber are sampled from R_q^k, where $R_q = \mathbb{Z}_q[x]/(x^n+1)$, $q = 3329$, $n = 256$. Kyber has three security levels, Kyber-512 (NIST Level 1), Kyber-768 (NIST Level 3) and Kyber-1024 (NIST Level 5) with $k = 2, 3, 4$, respectively.

The simplified version of Kyber.PKE is described in Algorithm. 1. Since the FO transform is not our target in this paper, we do not list the description of KEM here.

In Algorithm. 1, the $Parse, XOF, CBD$, and PRF functions sample polynomials from hashed random data. The $Encode, Decode, Compress$, and $Decompress$ functions transform the data between polynomials and binary value. The details of those functions and NTT please refer to [SAB+22]. In this paper, we focus on the multiplication of the two polynomials $\hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}}$ in NTT domain in KeyGen function.

Algorithm 1 Simplified Kyber.CPA.PKE

procedure KYBER.PKE KEYGEN
Input: None
Output: Public Key pk, Secret Key sk
  $(\rho, \sigma) = G(d)$, ▷ randomly sampled d
  $\hat{\boldsymbol{A}} = Parse(XOF(\rho))$
  $\boldsymbol{s}, \boldsymbol{e} = CBD(PRF(\sigma)), \hat{\boldsymbol{s}} = NTT(\boldsymbol{s}), \hat{\boldsymbol{e}} = NTT(\boldsymbol{e})$
  $\hat{\boldsymbol{t}} = \hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}$
  $pk, sk = Encode(\hat{\boldsymbol{t}}||\rho), Encode(\hat{\boldsymbol{s}})$
end procedure
procedure KYBER.PKE ENCRYPTION
Input: pk, m(message), r(random seed)
Output: Ciphertext c
  $\hat{\boldsymbol{A}} \leftarrow \rho$ ▷ $\rho \leftarrow pk$
  $\boldsymbol{r}, \boldsymbol{e_1}, e_2 = CBD(PRF(r)), \hat{\boldsymbol{r}} = NTT(\boldsymbol{r})$
  $\boldsymbol{u} = NTT^{-1}(\hat{\boldsymbol{A}} \circ \hat{\boldsymbol{r}}) + \boldsymbol{e_1}$
  $v = NTT^{-1}(\hat{\boldsymbol{t}} \circ \hat{\boldsymbol{r}}) + e_2 + Decompress(Decode(m))$ ▷ $\hat{\boldsymbol{t}} \leftarrow pk$
  $c = (c_1||c_2) = Encode(Compress(\boldsymbol{u}, v))$
end procedure
procedure KYBER.PKE DECRYPTION
Input: sk, c
Output: Recovered Message m
  $\boldsymbol{u}, v \leftarrow Decode(c)$
  $m = Encode(Compress(v - NTT^{-1}(\hat{\boldsymbol{s}} \circ NTT(\boldsymbol{u}))))$
end procedure

2.4 Prior Work

Lattice-based schemes have been subjected to a wide-variety of side-channel attacks, intended to perform message recovery and key recovery attacks, targeting all the three procedures (i.e.) key-generation, encapsulation and decapsulation procedures of Kyber KEM. Several of these attacks have specifically targeted the polynomial multiplication operation, and these attacks can be broadly split into two categories - (1) Multi trace attacks and (2) Single trace attacks. Since our attack mainly deals with operations within polynomial multiplication, we look into existing attacks targeting polynomial multiplication in literature.

2.4.1 Single Trace Attacks on Polynomial Multiplication

In traditional SCA methods, the success rate can be improved by increasing the number of side channel traces with statistical approach. As for lattice-based cryptography, single trace attack has received more attention. One reason is that many variables are generated randomly and used only once, which indicates that the attacker cannot obtain enough traces. Especially for the Key Generation function, the secret key should change every time when the attacker restarts this function.

In 2017, Primas *et al.* [PPM17] first introduced the SASCA method to solve the single trace attack on lattice. In this work, the whole NTT function in the Decryption procedure is abstracted into a factor graph with additional side channel information. Then Belief Propagation(BP) is used for finding the most probable correct candidate, by updating the information in the factor graph. In [PPM17], the side channel leakage comes from the inconstant process time of multiplication. Later in 2019, Pessl *et al.* [PP19] improved the attack, by changing the structure of one butterfly function in the factor graph for faster

BP. They also replaced the source of side channel information, targeting the constant time implementation, from timing leakage to the power leakage of loading/storing instruction. In 2022, Li *et al.* [LZH+22] extended SASCA to Toom-Cook multiplication in Saber. They utilized deep neural networks to enhance the template phase and optimized the factor graph by merging tracks based on Bayes' algorithm. Alternatively, In 2021, Aydin *et al.* [AAT+21] first utilized the horizontal leakage from the school-book multiplication operation in FrodoKEM. Even though the multiplication in FrodoKEM does not involve polynomials, this kind of horizontal leakage gives us some inspiration for this work.

2.4.2 Multi-Trace Attacks on Polynomial Multiplication

Polynomial multiplication has also been targeted by multi-trace attacks. In 2021, Hamburg *et al.* [HHP+21] exploited leakage from the NTT operation in a chosen-ciphertext setting, and showed that an attacker can craft invalid ciphertexts, which amplifies leakage information about the secret key from the INTT operation in the decryption procedure. Apart from the NTT operation, in 2022, Mujdei *et al.* [MWK+22] proposed a Correlation Power Analysis (CPA) targeting different kinds of polynomial multiplication strategies, including the multiplication after NTT. In their attack on Kyber, a search over $q^2 = 11082241$ guessing combinations needs to be applied and takes around 5 minutes on average. And because of the features of CPA, they need 200 traces to recover all n coefficients. In 2023, Yang *et al.* [YWY+23] also targeted the polynomial multiplication in Kyber using CPA. The difference is that they use the Chosen Ciphertext Attack to reduce the required traces, but nevertheless require a few hundred traces to perform full key recovery.

Those prior works can be categorized and compared with our attack in Table. 1. In target function column, the Dec, Enc, KeyGen in the brackets means decryption, encryption and key generation, respectively.

Table 1: Summary of Prior Works According to Attack Target and Number of Required Traces

	Target Function	No.Traces
[PPM17]	NTT (Dec)	1
[PP19]	NTT (Enc)	1
[HHP+21]	NTT (Dec)	k
[LZH+22]	Toom-Cook Based (Dec)	1
[AAT+21]	Matrix Multiplication (KeyExchange)	1
[MWK+22] [YWY+23]	Pointwise Multiplication after NTT (Dec)	20-200
Our work	Pointwise Multiplication after NTT (KeyGen)	1

2.5 Motivation and Overview of Our Attack

According to Table. 1, there are several works targeting the NTT in the Decryption or Encryption procedure with single trace, while other works concentrated on the pointwise multiplication under NTT domain in Decryption procedure with multiple traces. A natural question arises whether there are any additional vulnerabilities in the key generation procedure. Note that the TLS 1.3 protocol utilizes the ephemeral key setting. Under ephemeral settings, only single trace attacks are possible to recover the ephemeral key since the key generation is a probabilistic procedure, where one computation will not repeat again. So in this paper, we investigate the susceptibility of key generation procedure to single trace attacks.

In this work, we however show that single trace attacks are indeed possible, and the attack is enabled due to the structure of the M-LWE problem. Since the module dimension

is more than 1 in the M-LWE problem, this allows an attacker to observe much higher side channel leakage, as compared to the R-LWE problem.

So in the first phase of our attack, we leverage this new position of side channel leakage by Template Attack, and demonstrate that the success rate of Template Attack is larger than that in previous works.

In the second phase, we use our new key enumeration method to eliminate the error candidates after Template Attack. Although we still use Factor Graph to explain our method, we do not choose Belief Propagation in this phase, because of the structure of our attack target. Note that the accuracy of this phase can also be influenced by the security level of M-LWE problem.

The complete structure of our attack can be abstracted into the Pseudo Algorithm. 2.

Algorithm 2 Simplified Description of Our Attack (STAMP)

Input: Side Channel Leakage l, Public Key Matrix $\boldsymbol{A}$
Output: Secret Key s
1: F_l = Rank of candidate for $s \leftarrow$ Template attack(l) ▷ Phase 1
2: $\hookrightarrow$ Build Templates on Template Device
3: $\hookrightarrow$ Template Matching on Collected Traces from Targeted Device
4: Specific value of $s \leftarrow$ Key Enumeration(F_l,$\boldsymbol{A}$) ▷ Phase 2
5: $\hookrightarrow$ Use Information from Code and $\boldsymbol{A}$
6: $\hookrightarrow$ Search the Correct Value of s

3 Attack Phase 1: Template Attack

Template Attack [CRR02] was first proposed in 2003. Template attack requires the attacker to have access to a template device with the same character as the target device. First, the attacker could acquire side channel information corresponding to all the candidate values from the template device as templates. The attacker then can match the leakage collected from target device with templates to recover the data processed by the target device. We simply introduce the process of a traditional Template Attack below.

We denote the m collected template traces as $t_j (j \in [0, m-1])$ corresponding to a candidate value s_i and the means of those traces as $\bar{t}$. The noise vector denotes the difference between each template trace and $\bar{t}$ as $N_j = t_j - \bar{t}$. All the noise vectors can be processed into a noise covariance matrix CM_i corresponding to s_i. The elements of the matrix can be defined as

$$CM_i(u, v) = cov(N_u, N_v)$$

Then the attacker collects the trace, denoted as T, from the real target device. And the noise vector here denotes the difference between the real trace and each templates as $N_i = T - \bar{t}$. The attacker could match this real trace with the i-th template by computing the probability p:

$$p(i) = \frac{1}{\sqrt{(2\pi)^n \, |CM_i|}} \cdot e^{-\frac{1}{2} N_i^T CM_i^{-1} N_i}$$

The closest template to the real trace with the highest probability p indicate the most probable data.

In this work, we also use traditional template attack as our first phase of the attack, but we want to leverage more information in a single trace to recover the secret key more accurately.

3.1 Vulnerability Analysis

In this section, we mainly take Kyber-768 with $k = 3$ as an explanation and we denote the multiplication result as one polynomial vector $\boldsymbol{p}$. $A_{i,j}$ denotes the polynomial in row i and column j of matrix $\boldsymbol{A}$. As a feature of M-LWE, the multiplication of the matrix $\boldsymbol{A}$ and secret key $\boldsymbol{s}$ named as *polyvec_basemul_acc_montgomery* is implemented as:

$$A_{0,0} \circ s_0 + A_{0,1} \circ s_1 + A_{0,2} \circ s_2 = p_0 \tag{1}$$
$$A_{1,0} \circ s_0 + A_{1,1} \circ s_1 + A_{1,2} \circ s_2 = p_1 \tag{2}$$
$$A_{2,0} \circ s_0 + A_{2,1} \circ s_1 + A_{2,2} \circ s_2 = p_2 \tag{3}$$

Each polynomial of $\boldsymbol{s}$ is involved in the computation three times, referring to the equations above. As a result, the loading operation of all the coefficients within one polynomial (e.g. s_0) will repeat at least three times in computation of $A_{0,0} \circ s_0$, $A_{1,0} \circ s_0$, and $A_{2,0} \circ s_0$ seperately. The side channel leakages of those three loading operations reveal the same data, although they are mutually independent. We can reduce the effect of noise and enhance the success rate of template matching by utilizing multiple leakages.

Template attacks in recent works typically built the templates on the Hamming Weight (**HW**) of the data rather than the specific value of data. In this work, we also build the template on different **HW**. Since we simply target the loading/storing operation, rather than the whole computation process. Even though the template is built on a specific value, the different data with the same **HW** still leak the same side channel leakage according to Hamming Weight model. However, our target is not NTT or Toom-Cook functions as in previous works. Instead, we focus on the *basemul* function, which is used to multiply two polynomials in the NTT domain in Kyber. The computation procedure for *basemul* is shown in Alg. 3.

Algorithm 3 Pseudo Code of Basemul and Fqmul Function

```
1: procedure BASEMUL(&r[2], &a[2], &b[2], zeta)
2:     r_0 = fqmul(a[1], b[1])        ▷ &a[2] represents address of two coefficients from a
3:     r_0 = fqmul(r[0], zeta)
4:     r_0+ = fqmul(a[0], b[0])
5:     r_1 = fqmul(a[0], b[1])
6:     r_1+ = fqmul(a[1], b[0])
7:     Return r[0], r[1]
8: end procedure
9: procedure FQMUL(a, b)
10:    m = montgomery_reduce(a * b)
11:    Return m
12: end procedure
```

In Kyber, the polynomial multiplication is implemented on every two coefficients in pair with *basemul* function. For instance, the $a[2]$ indicates two coefficients of $A_{0,0}$ and the $b[2]$ indicates two coefficients of s_0 with the same index as $a[2]$. The *fqmul* multiplies two coefficients directly and then applies the *montgomery reduce* to the result.

Notice that in *basemul*, one coefficient will be loaded twice in different *fqmul*. In the whole process of *polyvec_basemul_acc_montgomery*, one polynomial participates in k polynomial multiplications. Then one coefficient within one polynomial will be loaded $3 * 2 = 6$ times when $k = 3$. Based on those conditions, we can build templates on the six loading operations for a more accurate template matching result.

3.2 Template Matching on Simulation Trace

First, we use ELMO [MOW17] as an auto-simulation tool to generate the power traces of the *basemul* function for proving the existence of the side channel leakage. The ELMO will generate noise-free power traces according to the compiled binary file with its own settings. In this work, the **POWERTRACES** and **CYCLEACCURATE** options in ELMO are defined to ensure that the value and cycles of simulated traces are close to real traces.

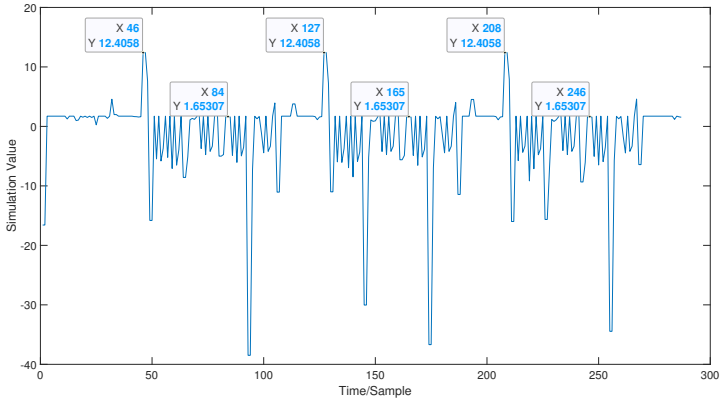


Figure 1: The Simulated Trace for Three Basemul Function

For the demonstration, we only choose three *basemul* functions with same input $b[2]$ (the first two coefficients of secret polynomial s_0), but different $a[2]$ to generate the traces, which indicates the *basemul* function on the first two coefficients of $A_{0,0} \circ s_0$,$A_{1,0} \circ s_0$,and $A_{2,0} \circ s_0$ respectively. And the source codes that we used for the simulation come from the reference implementation of Kyber. One simulated trace of the three *basemul* is shown in Figure 1. The ELMO will also generate a text file called *asm output*, each line of the assembly instruction in this text corresponding to one simulated point in the trace. Referring to Figure 1 and *asm output*, three pairs of the points corresponding to double loading operations with the same value can be found. The points with higher value (12.4058) represent the loading operation at the beginning of the *basemul* computing r_0. And the lower one corresponds to the loading for computing r_1. The difference between those two values is caused by various load instructions. The higher value corresponds to the *ldr*, while the other is *ldrsh*. According to the ARM architecture reference manual, the *ldr* loads a word from memory, while the *ldrsh* loads a signed half word.

Those simulation results preliminarily demonstrate the existence of the leakage target. We then build templates on those simulated traces and verify the success rate of template matching. The template attack contains two phases, which are template building and matching. The template matching will definitely succeed if no noise is factitiously added to those traces during the template building phase.

Then we add the Gaussian noise with incremental standard deviation σ to imitate the real traces. In the matching phase, we choose traces corresponding to the input value in the range of $[1, 3329]$, and examine whether those traces can be classified into correct hamming weight. The results are listed in Table. 2. The template matching success rate gets slightly influenced when the σ is in the $[0.1, 0.4]$ range. Moreover, when the σ gets larger than 0.5, the success rate will significantly decrease. We also list the experiment results of building templates for different number of *basemul* functions. It is obvious that more targeted locations lead to a higher success rate of template matching.

This can be explained from a theoretical perspective. Recall that the added artificial noise follows the Gaussian distribution. If we apply the template matching on only one

Table 2: Template Matching Results for Simulation

σ of Noise	SR of 3 *basemul*	SR of 2 *basemul*	SR of 1 *basemul*
0.1	100.00%	96.13%	88.85%
0.2	98.97%	85.33%	76.33%
0.3	93.94%	75.23%	68.18%
0.4	88.42%	67.21%	59.37%
0.5	79.67%	58.90%	46.78%

point, the added noise still has a slight probability of making the value of this point closer to a wrong candidate. However, if multiple points are considered, the covariance matrix of those points will decrease the error probability as much as possible.

In the prior work [KPP20], the authors investigated the practical leakage model for different platforms. The σ is around 0.5 for an 8-bit XMEGA, while in the range of $[0.4, 3.0]$ for the 32-bit STM32F4 board. According to our simulation results, the attack will have enough success rate only when the σ is smaller than 0.4. Since our target Kyber is commonly implemented on the 32-bit platform, it is necessary to validate the leakage on the 32-bit device and test if we can still get the correct template matching result.

3.3 Template Matching on Real Trace

According to the simulation experiment, we could easily identify the target instruction and associated locations in the traces. However, the real-world captured traces will not be as ideal as simulated traces, so we need to locate the target points in real traces.

3.3.1 Experiment Setup

We choose the ChipWhisperer CW308-STM32F3 as the target board to collect the power traces. This target board is also a candidate board in *pqm4* [KPR+] open-source library. An +20dB amplifier is used to guarantee the signal being clear. Then we use the KEYSIGHT DSOX3034T as the external oscilloscope to capture the power traces. The whole capture experiment device is shown in Figure 2.

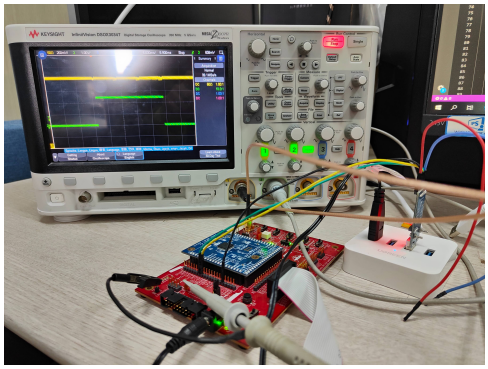


Figure 2: Capture Experiment Device

3.3.2 Leakage Analysis of Reference Implementation

In this part, we target the reference implementation of Kyber with the same codes used in the simulation section. One sample trace is shown in Figure 3. This captured trace is obviously not as clear as the simulated trace. However, the trace can still be separated

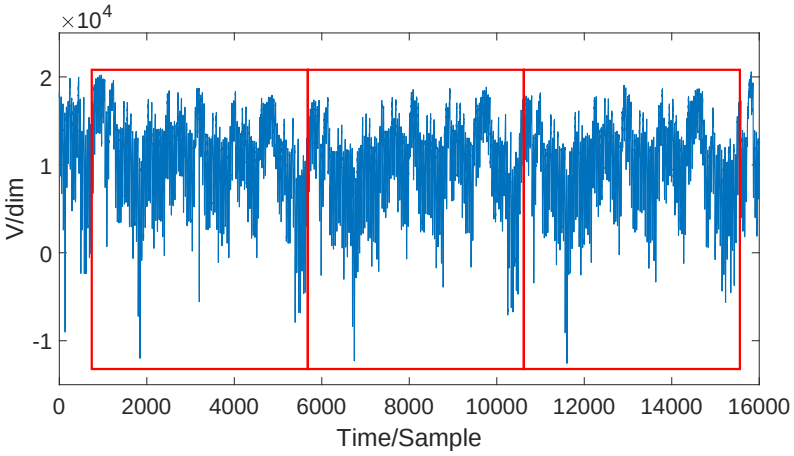


Figure 3: One Sample Trace for Ref Implementation

into three parts using SPA(Simple Power Analysis), marked in red. And each piece of the trace in one red rectangle indicates one *basemul* function. To locate the specific target points, we use Pearson Correlation of the secret key coefficients and identify the highest correlation location. The correlation result is shown in Figure 4. The blue trace indicates the correlation value of the secret key. There are many locations with the correlation value higher than 0.6. With the help of Figure 3, the correlation trace can also be separated into three parts with the same pattern. We mark the highest correlation point value in every part, indicating our attack targets. However, there are also several peaks at the beginning of the trace that do not follow the pattern (at positions around 1000 and 1500 on the x-axis). These isolated peaks correspond to other secret key-related instructions. According to these patterns, three features of the correlation values can be summarized:

1. There are two obvious peaks in the part of the trace corresponding to one *basemul* function. Each peak indicates one loading operation in the *fqmul*.

2. Other peaks, where the correlation values are higher than 0.5 but showing a downward trend, correspond to the computation process in one *basemul*.

3. The value of the correlation coefficient decreases as the number of *basemul* increases.

Feature 1 and 2 can be utilized to locate the target points for template attack. Feature 3 shows that the same operation in real captured traces will have slightly different leakages at different timing. And in [PP19] (Section 6.2), this feature was also discovered and discussed by the authors. This feature can also provide evidence of our idea that we should build the template for all three different loading locations, rather than just a single one. Since those leakages from different locations will provide different additional information, we should utilize them comprehensively rather than only considering one of them.

3.3.3 Template Attack on Real Traces

Based on the leakage analysis before and the discovered features, our template analysis target is loading operations in multiple *basemul* functions with the same input. We recall that those leakages of loading operations are mutually independent and slightly different. Hence two methods for building the templates are considered:

A. Simply combining those peaks together and building the templates on those points simultaneously.

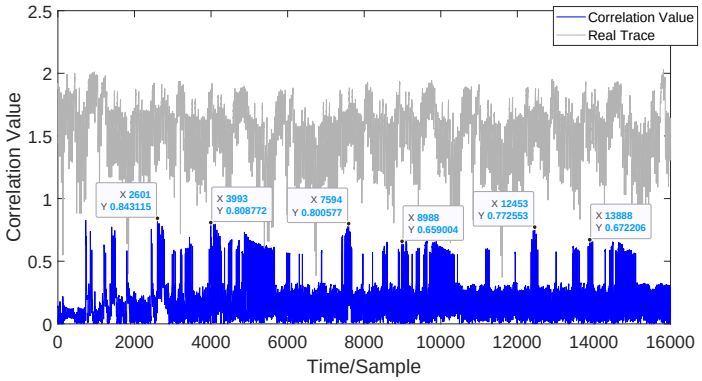


Figure 4: Correlation Location of Loading Secret Key VS. Real Trace for Ref Implementation

B. By averaging the values of peaks in three different parts and building templates on the mean of points.

We examine those two methods by comparing the template matching success rate on the same dataset and list the result in Table. 3. The dataset contains 900 template traces for each **HW** candidate, and we choose 5000 traces to test the template matching success rate. In this table, **Matching Success Rate** indicates that the most possible candidate in the template matching rank is the correct data. And the **Total Success Rate** means that the first 5 possible candidates contain the correct one. It is obvious that combining the peaks together can get better template matching results. So in later experiments we will use method A to implement the template attack.

Table 3: Success Rate for Two Template Building Methods

Candidate	Matching Success Rate	Total Success Rate
A	91.69%	99.99%
B	57.99%	89.37%

3.3.4 Testing the Template Matching on Different Security Level

In the experiments before, our target is Kyber-768 (NIST Security Level 3), using $k = 3$. Larger k implies a higher security level of Kyber. Theoretically, larger k also indicates more multiplication operations for each secret coefficient and more side channel leakages, which has been verified by simulation in Table. 2. Even though the number of the secret coefficients gets larger, we can improve the total success rate by recovering single coefficient more accurately. In this part, we validate this theory on the real device and check whether different k leads to different template matching success rate.

Since we have obtained the real traces with $k = 3$, we use this dataset to test the template matching success rate with $k = 1$ to 3. The results are shown in Table. 4. It is shown that larger k still increases the success rate of template matching even on the real collected traces. That means a higher security level of M-LWE problem will indeed produce more side channel leakage.

Table 4: Template Matching Results for Real Traces

k	1	2	3
SR	94.79%	98.89%	99.95%

3.3.5 Leakage Analysis of Optimized Implementation

We also investigated the same leakage from the optimized implementation of Kyber. The target implementation comes from *pqm4* [KPR+][1] public library. In the recent version of *pqm4*, the *basemul* function is also optimized and re-written in ARM assembly code. We located the loading operation of the secret key and found that to take full advantage of the 32-bit-register, the secret key is loaded pair-wisely in one loading instruction. Half of the register has enough length to save one input coefficient. Meanwhile, one *basemul* function will be accomplished in a single assembly function, rather than several sub functions(e.g. *basemul* in C reference implementation with multiple *fqmul*). That means the input coefficients only need to be loaded once in one *basemul* function. So the number of our target operations decreased to 3, rather than 6.

We list the **Total Success Rate** of our template attack on optimized implementation in Table. 5. In this experiment, we use 400 traces for each **HW** and test the success rate on 4000 traces.

Table 5: Total Success Rate of Template Matching for pqm4 Traces

k	1	2	3
SR	97.48%	98.83%	99.44%

In fact, the template matching results are not affected significantly with all those changes in pqm4. One possible reason is that the assembly implementation has fewer redundant operations, which enhances the resolution of the collected trace. The correlation value corresponding to input coefficient VS. one real trace for assembly implementation are depicted in Figure. 5. It can be clearly seen that both the traces of correlation and power show a more pronounced pattern than those in reference implementation.

There is one more difference between the analysis on pqm4 and reference implementation. We need to build template for **HW** ranging from 0 to 24, rather than 12, because of pairwise loading. So after template matching, we can only get the sum of the **HW** of two input coefficients.

4 Attack Phase 2: Key Enumeration

Template matching is the first stage of our attack. The template matching can not guarantee that the first candidate in the matching rank is the correct one. Therefore, we combine the arithmetical information in the algorithm with the side channel leakages in one trace together for higher attack accuracy.

It is widely studied that the Belief Propagation (BP) can be used in SASCA for the second stage of template attack to identify the correct one in the rank. However, using BP will meet several practical problems, e.g., the convergence of BP and the large memory usage. It is known that imperfect factor graphs will make BP harder to converge. We attempt to provide a more simple and quick analysis phase rather than BP.

[1] Our analysis and experiments were carried out on the implementation of Kyber-768 corresponding to the commit hash 1eeb74e4106a80e26a9452e4793acd6f191fe413.

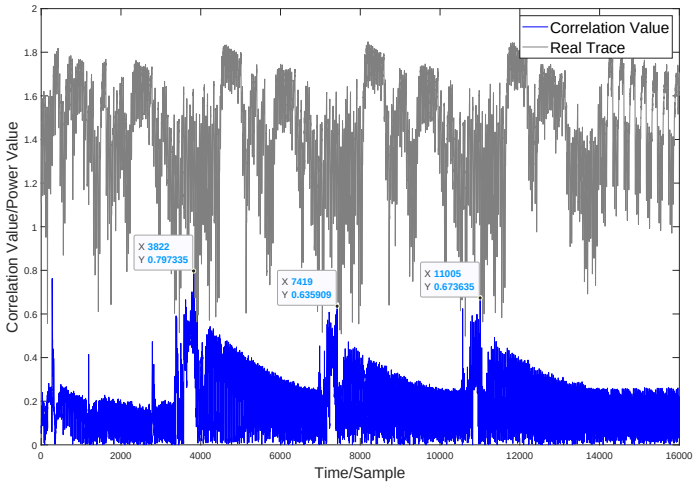


Figure 5: Correlation Location of Loading Secret Key VS. Real Trace for Assembly Implementation

4.1 Factor Graph

The factor Graph is the fundamental part of Belief Propagation. The detailed description of BP can be found in [Mac03]. In the Factor Graph, the variables are represented by circles, and we call those circles as variable nodes. The relationships among those variables are represented by squares, named as factor nodes.

Our template target is the input and output of triple *Basemul* functions. In **KeyGen** function, the polynomial multiplication is implemented between vector $\hat{\mathbf{s}} = NTT(\mathbf{s})$ and matrix $\mathbf{A}$ as shown in Equation. 1. The matrix $\mathbf{A}$ can be computed from the public key, so the attackers will easily get the specific value of $\mathbf{A}$. The Factor graph of our target can be represented as Figure 6. There are two kinds of factor nodes in this figure, F_l indicates the side channel leakage of the connected variable. As mentioned before, our templates are built on the **HW** of data, so the F_l indicates the rank of possible **HW** of the connected variable nodes. Every specific candidate of this coefficient with the same **HW** will have the same initial probability. The equation of F_B is explained in Equation. 4.

$$F_B = \begin{cases} 1, & r_0, r_1 = basemul(s_0, s_1, A_0, A_1) \\ 0, & else \end{cases} \tag{4}$$

In Figure 6, the variable nodes are marked in two colors. The input coefficients s_0, s_1 are marked in red, which means those variables are loaded multiple times and can strengthen the template matching. The parameter $\mathbf{A}$ is written in factor nodes of F_B, which means that those data will be used in Equation. 4 and are public to the attacker. And the output of *basemul* is marked in blue. For those output results, we also attempt to recover their **HW** with template attack on the loading/storing operation, but those data were only stored once. So the template matching results of those blue data may not as accurate as red data.

4.2 Searching the Correct Candidate

Belief Propagation can quickly eliminate incorrect candidates through its message updating scheme, when dealing with large-scale factor graphs, such as those for a complete NTT function. Notice that the number of iterations required for the convergence of BP also depends on the structure of the factor graph.

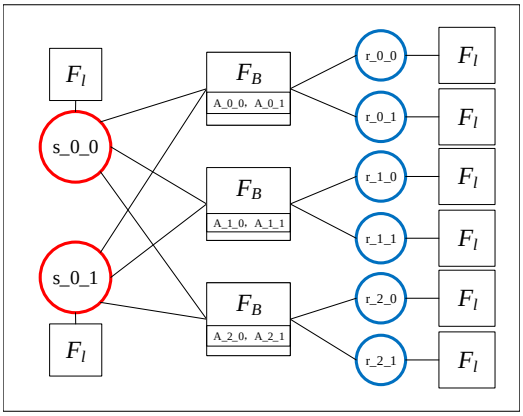


Figure 6: Factor Graph for Triple Basemul Function

In contrast to the complexity of NTT, which contains $n \cdot logn$ interconnected butterfly units, our target graph is relatively simple, consisting of only 3 isolated F_B coefficients. As such, it is not necessary to utilize message scheduling in Belief Propagation, and the probability distribution of candidates can be directly deduced using Equation. 4.

The detailed steps of our key enumeration method are listed below:

1. Reserve the top five most possible **HW** in F_l for later enumeration, denoted as $\mathbf{HW_5}$.
2. Examine every candidates in $\mathbf{HW_5}$, then assign the probability of candidates with the results of F_B.
3. Identify the correct candidate whose probability does not decrease.

Referring to Table. 3, our template matching results demonstrate a potential Total Success Rate up to 99.99%. That means the correct **HW** for the secret coefficient stands a great chance to be captured in the retained $\mathbf{HW_5}$. So the least likely values of **HW** can be directly discarded, roughly reducing the searching space by half.

4.3 Analysis of Second Phase Accuracy

Referring to Figure 6, the factor node F_B is variable due to different values of $\boldsymbol{A}$. For specific values of $\boldsymbol{A}$ in the three F_B nodes, several pairs of input may have the same **HW** value, and their corresponding output from *3basemul* may also have the same **HW**. This means that several candidates may still have the same probability after the key enumeration process. We refer to these pairs of input and output with the same **HW** as **CPIO**, which means Confusing Pairs of Input and Output.

Since the values of $\boldsymbol{A}$ are randomly sampled, there are $3329 * 3329$ possible combinations of two coefficients in $\boldsymbol{A}$. It is not feasible to test all possible scenarios, so we randomly generate some values of $\boldsymbol{A}$ to examine the likelihood of the above situation occurring. We can also take Figure 6 as an example. We randomly generate values for $\boldsymbol{A}$ in the three F_B nodes and assign specific values within [0,3329] to $s[0]$ and $s[1]$. We then compute the **HW** of the input and output and examine how many pairs have the same **HW**. We repeat this test 100 times with different values of $\boldsymbol{A}$ for the three F_B nodes. The results show that approximately 5% to 8% of pairs are **CPIO** for a specific value of $\boldsymbol{A}$.

Note that **CPIO** does not necessarily mean that only two pairs of input and output have the same **HW**. For a specific **HW** and value of $\boldsymbol{A}$, there may be 2 to 4 inputs with the same **HW** and their corresponding outputs also have the same **HW**. This means

that for our attack, approximately 92% to 95% of the recovered coefficients are uniquely determined, while the remaining locations require traversal to find the correct coefficient within a search space of [2,4]. We also get the probability of each quantity of **CPIO** through this test and listed the results in Table. 6. This indicates that most **CPIO** only require searching within two pairs.

Table 6: Probability of the searching space of CPIO

Quantity of CPIO	2	3	4
Probability	74.72%	19.37%	4.59%

However, notice that the higher security level of Kyber, that is, larger value of k, leads to less **CPIO**s. Because more *basemul* will increase more constraints into the factor graph, which indicates that the additional *basemul* may decrease the probability of the incorrect candidate in **CPIO**s to be zero.

There is also another method that can be utilized to decrease the false positive **CPIO**. Referring to algorithm. 3, line 2 and line 3 represent the computation procedure and intermediate variables of r_0. According to Figure 4 and 5, the side channel leakage of these computation instructions after the loading operation still show the relationship with our target secret key (referring to the correlation value). The side channel leakage of those variables can still be utilized, even though the template matching accuracy of those instructions is lower than loading operation. But that means we need to add thousands of templates for different location within a single trace and also large memory storage. So it will be a trade-off between searching space and storage space.

4.4 Analysis of Accuracy for the Complete Attack

In this section, we first recall the two phases of the proposed attack. In the first phase, template matching is performed on the trace, and the top 5 most likely matched Hamming Weight for each variable are retained. In the second phase, the correct candidate value is sought among the retained $\mathbf{HW_5}$. As long as the Hamming Weights of the correct candidates retained in the first phase are correct, it can be guaranteed that they can be found in the second phase, and only the size of the searching space needs to be considered. Therefore, in this part, we discuss how large the overall theoretical searching space is in the second phase, assuming that the first phase is correct.

Referring to Section. 4.3, the probability of the appearance of **CPIO** is 5% to 8% for a specific value of $\boldsymbol{A}$. We denote this probability as $p(CPIO)$, which means that for our attack, approximately $p(CPIO)$ of n coefficients may have **CPIO**. We denote the quantity of **CPIO** within one coefficient as $\#(CPIO)$, which means the theoretical searching space will be $\#(CPIO)$ for one specific coefficient that has **CPIO**. And according to Table. 6, we can compute the average quantity of each **CPIO** as:

$$2 * 74.72\% + 3 * 19.37\% + 4 * 4.59\% \approx 2.2$$

We use the average value of $p(CPIO) = 6.5\%$ with $\#(CPIO) = 2.2$ to represent the ideal average search space for each **CPIO**. However, if we need to search for the correct value in $\mathbf{HW_5}$, the searching space will increase to $2.2 * 5 = 11$. Therefore, the total searching space for recovering one polynomial with $n = 256$ coefficients under $\mathbf{HW_5}$ might be:

$$\{5 * \#(CPIO)\}^{p(CPIO)*n} = 11^{6.5\%*n} = 11^{16.64} \approx 2^{57.5} \ (n = 256)$$

This searching space is somewhat close to a threshold 2^{64} that indicate the theoretical searching space ability with common computational resource.

But assuming the case that we search the correct coefficient just in the correct **HW**, the total searching space will decrease to:

$$3^{6.5\%*n} = 2.2^{16.64} \approx 2^{21} \ (n = 256)$$

This searching space is much smaller than the threshold 2^{64}. That means, if we can use more accurate side channel collection method or other methods like deep learning, to increase the **Matching Success Rate**, rather than **Total Success Rate**, the searching space of our attack can decrease significantly. And note that since our attack can recover each polynomial in different computation device separately, we do not need to compute the k power of the searching space like $(2^{21})^k$. So this searching space is still practical for different value of k.

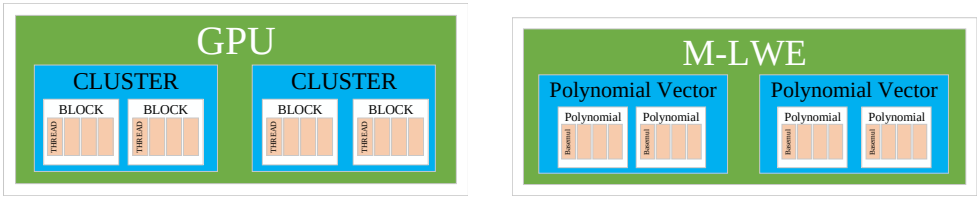
5 Accelerating Enumeration Phase by CUDA

In section 4, three *basemul* functions with same secret input are analyzed as a whole for eliminating the correct input value. These three *basemul* are independent of other *basemul* with different inputs. Therefore, the attacker can independently recover two coefficients with our attack. Taking advantage of this feature, parallelization is the most suitable method for acceleration. In this section, we provide a parallelized version of the Key Enumeration phase as a DEMO to demonstrate the practicability of acceleration.

5.1 CUDA

In the computer architecture, a sequence of operations in the computer system can be called a *thread*. The CPU is designed for executing several threads as fast as possible. While the GPU aims at running thousands of threads together in parallel. Therefore, the GPU is usually specialized for highly parallel computations, such as image processing and machine learning. In GPU, plenty of *threads* gather together as a *thread block*. One *block* may contain up to 1024 *threads*. And *block* can also be a unit of *cluster*.

The NVIDIA CUDA (Compute Unified Device Architecture) Toolkit is a computing platform and programming model that leverages the parallel computing resources in NVIDIA GPU. In CUDA, a specific function, called *kernel*, can be defined to be executed N times in parallel by N *threads*. The architecture of GPU can be depicted as Figure 7(a).



(a) Architecture of GPU Unit (b) Architecture of M-LWE

Figure 7: Structure Comparison of GPU and M-LWE Problem

5.2 Acceleration Implementation

Fortunately, the M-LWE problem can also be described in the same structure, as shown in Figure. 7(b). We recall the parameters in Kyber in section 2, that $n = 256$, $k = 2, 3, 4$ respectively. Remember that our attack recovers two coefficients in one function. So recovering one polynomial containing 256 coefficients can be assigned to one block, with

128 threads in parallel. Different blocks can be in charge of different polynomials of the secret key.

Furthermore, GPU can also be utilized to eliminate the wrong candidate caused by **CPIO**. Different guess of one secret polynomial can be loaded into one *thread* and then run the inverse NTT on this polynomial, only the correct polynomial has the result of inverse NTT that satisfying the distribution of secret key.

One advantage of our attack is that the time of the GPU-based enumeration phase will not be affected by the security level of the M-LWE problem. In the M-LWE problem, the higher security level indicates more polynomials in secret key that need to be recovered. While in our attack, larger k means executing more polynomial multiplications, along with more instructions to be leveraged in template building and higher attack accuracy. With the help of GPU, additional polynomials can be solved by configuring more blocks in parallel.

Without other optimization, the parallelized analysis only takes 2 seconds for one polynomial with 256 coefficients. For the comparison, in [PP19], it will take approximately 8min for a full iteration of BP. Our attack is hundreds faster than the unstable BP.

When executing polynomial multiplication $\hat{\mathbf{A}} \circ \hat{\mathbf{s}}$, each polynomial of $\hat{\mathbf{s}}$ will multiply k polynomials in one column of $\hat{\mathbf{A}}$ separately. That means in the memory of one block, k polynomials of constant matrix $\hat{\mathbf{A}}$ are required. Besides, in each *thread*, the probability distribution of candidates also needs space to be saved. Further optimization of memory sharing will be our future work.

6 Attack on Countermeasures against SCA

In this section, we will discuss the applicability of our attacks on protected implementations with certain countermeasures against SCA, such as masking and shuffling.

6.1 Masking

Masking is widely used for mitigating the threat from SCA attacks. In the masked implementation, the secret variables will be split into several shares and processed by carefully devised functions respectively. These shares will be re-combined together to compute the real output after the masked domain. The attackers may not have sufficient side channel information to recover every share on the masked implementation.

6.1.1 Generalized Masking

In the application of masking in lattice-based cryptography, functions such as NTT or polynomial multiplication do not require adjustment for multiple shares due to their linearity. For example, a bitslicing-based masking implementation was proposed in [BC22]. In this implementation, masking is applied to the Decapsulation in Kyber. The encryption and decryption functions in Decapsulation are reorganized and shown in algorithm. 4.

We also mark the variables and functions by the colors as used in [BC22]. The green parts do not require masking. The blue parts indicate the functions that can be applied linearly to the shares, while the red parts represent non-linear functions with masking gadgets. It is shown that our target is marked in blue and green, indicating that the traditional masking method will not modify the implementation of this function.

Our proposed attack is not suitable for the decryption function in Kyber, as the multiplication in decryption occurs between two polynomial vectors, rather than between one vector and one matrix. As a result, each polynomial in the vector will only be multiplied once, rather than k times. In this section, we discuss whether our attack still works when this masking method is applied on the Key Generation function.

Algorithm 4 Masked Implementation of Decapsulation in Kyber[BC22]

procedure KYBER.CPAPKE.DEC($\hat{s}, c$)
  $\boldsymbol{u} = Decompress(c_u)$
  $v = Decompress(c_v)$
  $\hat{z} = \hat{s}^T \circ NTT(u)$
  $\omega = v - NTT^{-1}(\hat{z})$
  $Compress(\omega)$
end procedure
procedure KYBER.CPAPKE.ENC(pk, m, σ)
  $\boldsymbol{r}, \boldsymbol{e_1}, e_2 = CBD(PRF(\sigma))$
  $\hat{\boldsymbol{r}} = NTT(\boldsymbol{r})$
  $\hat{\boldsymbol{u}} = NTT^{-1}(\hat{\boldsymbol{A}}^{-1} \circ \hat{\boldsymbol{r}}) + \boldsymbol{e_1}$
  $\hat{\boldsymbol{u}} = NTT^{-1}(\hat{\boldsymbol{t}}^{-1} \circ \hat{\boldsymbol{r}}) + \boldsymbol{e_2} + Decompress(m)$
  $c = Compress(\boldsymbol{u}, v)$
end procedure

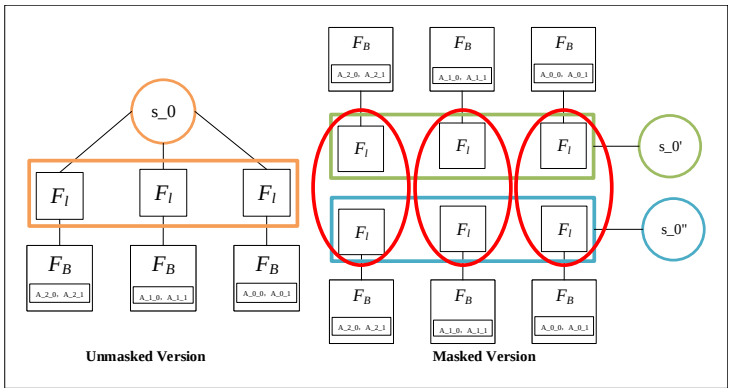


Figure 8: The Relationship among Variables and Leakage

We note that more side channel information will leak if the shares of one polynomial are **not** changed and regenerated for different *basemul*. For example, if one coefficient s_i is split into two shares s_i' and s_i'', for the first level masking, then *basemul* (s_i', A_i) and *basemul* (s_i'', A_i) will leak the side channel information of s_i' and s_i''. The recovery of these two shares can proceed in the same manner as the attack on s_i in an unmasked implementation. Additionally, in a masking implementation, we also have additional information that $s_i = s_i' + s_i''$.

Recall that the strategies discussed in Section. 3.3.3. The template matching success rate gets higher because of combining the leakage from multi-locations. Theoretically, that means the leakage from multi-locations can reduce the adverse effect from noise within single location. We depict the relationship among those leakages in Figure. 8.

On the left side of Figure 8, the template attack on reference implementation is re-drawn. Three leakages from *basemul* are combined to recover the s_i. We call those leakages from three F_B as horizontal leakage. In the masking scenario, the horizontal leakage still exists, shown on the right side of Figure 8. At least we can recover s_i' and s_i'' with same success rate as recovering s_i.

Note that the red circles in this figure indicate that $s_i' + s_i''$ should have a consistent result s_i. This correlation can be named as vertical leakage. Those vertical leakages have the same feature as the horizontal leakage that each of the leakage is mutually independent with others, however indicating same operation and processed data. Using this information, the success rate of our template matching phase can be closer to 100%.

6.1.2 Specific Masking

There is another kind of masking method proposed by Ravi *et al.*[RPBC20] in 2020. Their masking method especially aims at single trace attack on NTT like SASCA. The basic operation *butterfly unit* in NTT takes two inputs $(a, b) \in \mathbb{R}_q^2$ and a known twiddle constant ω with two outputs $(c, d) \in \mathbb{R}_q^2$. The operation of one *butterfly unit* can be represented as:

$$c = a + b \cdot \omega \tag{5}$$
$$d = a - b \cdot \omega \tag{6}$$

In that work, the *butterfly unit* of NTT is masked by adding a random mask ω_s on the twiddle constant like:

$$c' = c \cdot \omega_s \tag{7}$$
$$= (a + b \cdot \omega) \cdot \omega_s \tag{8}$$
$$= a \cdot \omega_s + b \cdot \omega \cdot \omega_s \tag{9}$$

Refer to line 3 in Algorithm. 3, the *zeta* also represents the twiddle constant used in NTT. In our attack, we only build the template on the input $b[2]$ and the output $r[2]$ in line 4 and 6 in Algorithm. 3. So if the mask is applied on the *zeta* in line 3 as $zeta \cdot \omega_s$, the result r_0 will be masked and we can not run the Key Enumeration just according to F_B as Equation. 4. That means this masking method could defend against our single trace attack.

6.2 Shuffling

Shuffling is also a commonly used countermeasure against SCA. In cryptographic algorithms, plenty of tiny operations are the same functions but processing different data. Shuffling means shuffling the order of those operations randomly so the attackers can not match the known information with collected traces.

In [RPBC20], Ravi *et al.* also proposed several shuffling methods for NTT. If those methods are applied on the *basemul*, the order of loading operation or the order of *basemul* in the whole polynomial multiplication may be shuffled. Under this countermeasure, it is not possible to build templates on several locations of loading operation directly, so our attack will also be mitigated by this kind of shuffling.

However, the attack against shuffling is also possible with more side channel information. Note that even though the order of operations is changed, the corresponding constant $A_{i,j}$ remains unchanged. So if the loading operation of $A_{i,j}$ can also be located and utilized by more templates or Deep Learning-based analysis as in [LZH+22], it may still be possible to locate the loading operation of secret key correctly.

7 Conclusion and Future Work

In conclusion, we propose a single trace attack on the key generation function in Kyber to recover the generated secret key. Our attack mainly targets the *basemul* function, which is used in the polynomial multiplication after NTT. And this k times leakage is caused by the structured M-LWE problem, in which one polynomial participates the multiplication k times. Although this location of leakage has been noticed before, their attack needs 20 to 200 traces, our attack only needs a single trace. Besides, larger k may also enhance our attack in analysis phase. We validate our attack on simulated traces by ELMO, and real traces of reference implementation and pqm4 implementation collected from STM32F3 board by oscilloscope. Because of the parallelization of *basemul*, we can utilize GPU to accelerate the key enumeration phase of the attack. Furthermore, we also investigate

the applicability of our attack on the traditional countermeasures against side channel attack, such as masking and shuffling. The incomplete implementation of masking can not effectively defend against our proposed attack. Moreover, we do not use the complex BP algorithm for key recovery. So much simpler analysis is possible.

As the future works, first, we could optimize our accelerated attack by better assignment of the shared memory.

Second, this attack may lead to the consideration that whether the structured LWE problem has more unnoticed leakages in the implementation.

Third, our attack discover a new leakage from M-LWE problem. One natural question is that whether the Dilithium has the same leakage. In this work, we take Kyber as an example because the pointwise multiplication in Kyber has double loading operations for every *basemul*. And this lead to double side channel leakage of secret key compared with Dilithium. However, we also demonstrate that in pqm4 implementation, there is only one loading operation in each *basemul* and we can also recover the secret key. So the attack on the Dilithium may also be effective, but that will be our future work.

References

[AAT+21] Furkan Aydin, Aydin Aysu, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange and encapsulation protocols. *ACM Trans. Embed. Comput. Syst.*, 20(6):110:1–110:22, 2021.

[BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):553–588, 2022.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking kyber: First- and higher-order implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):173–214, 2021.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):88–113, 2021.

[HKL+22] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Amber Sprenkels. First-order masked kyber on ARM cortex-m4. *IACR Cryptol. ePrint Arch.*, page 58, 2022.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):243 – 268, 2020.

[KPR+] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LZH+22] Yanbin Li, Jiajie Zhu, Yuxin Huang, Zhe Liu, and Ming Tang. Single-trace side-channel attacks on the toom-cook: The case study of saber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):285–310, 2022.

[Mac03] David J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. ELMO: Evaluating Leaks for the ARM Cortex-M0. https://github.com/sca-research/ELMO, 2017.

[MWK+22] Catinca Mujdei, Lennert Wouters, Angshuman Karmakar, Arthur Beckers, Jose Maria Bermudo Mera, and Ingrid Verbauwhede. Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Trans. Embed. Comput. Syst.*, nov 2022.

[NIS16] NIST. Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2016.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering*, Lecture Notes in Computer Science, pages 123–146, Cham, 2020. Springer International Publishing.

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[YWY+23] Yipei Yang, Zongyue Wang, Jing Ye, Junfeng Fan, Shuai Chen, Huawei Li, Xiaowei Li, and Yuan Cao. Chosen ciphertext correlation power analysis on Kyber. *Integration*, February 2023.