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

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

[1] Zhejiang University, Hangzhou, China,
[2] Temasek Laboratories, Nanyang Technological University, Singapore
yangbolin@zju.edu.cn prasanna.ravi@ntu.edu.sg fanzhang@zju.edu.cn

Abstract.
In this work, we propose a novel single-trace key recovery attack targeting side-channel leakage from the key-generation and encryption 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 and secret key-related computations, referred to as STAMPs appearing on a single side channel trace, compared to the Ring-LWE problem of similar security level. We exploit leakage from the pointwise multiplication operation 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 countermeasures against traditional SCA such as masking and shuffling. We believe our work motivates more research towards SCA resistant implementation of key-generation and encryption 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, and announced the list of algorithms for Public Key Encryptoin (PKE), Key Encapsulation Mechanisms (KEMs) and Digital Signatures schemes (DSS) that are intended to be standardized [NIS23] in August 2023. 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 (M-LWE) 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 divided 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+22a].

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 [PPM17] [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. 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 between a polynomial matrix and polynomial vector. 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. Those repeated and secret key-related computations lead to k times leakages within a single side channel trace with different time-stamp, which are called as STAMPs in this work. We exploit leakage from the pointwise multiplication operation in the key-generation (encryption) procedure, and take advantage of the properties of the Module-LWE instance to enable potential single trace key (message) recovery attacks. Thus, our work demonstrates an additional side-channel vulnerability in the key-generation and encryption 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 and encryption 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 and encryption procedure of Kyber KEM. The use of Module-LWE problem for Kyber KEM ensures that the secret key is manipulated k

times during the pointwise multiplication operation where k is the dimension of the module. While the computation is performed in single coefficients and involve very few operations, we show that there is still enough leakage to perform single trace attacks.

2. Our work can be divided into two phases. In the first phase, we construct HW templates exploiting the STAMP behaviour, showing that the attacker can construct improved templates due to repeated leakage of the secret key. We validate the efficiency of improved template construction in both the reference and optimized implementations of Kyber KEM from the well known open-source pqm4 library. For the second and final phase of our attack, we propose a novel key enumeration algorithm, that is capable of exploiting available leakage from multiple intermediate variables for practical key recovery. Our approach is simpler than the Belief Propagation approach, traditionally used for single trace attacks on the polynomial multiplication operation.

3. The experimental validation of our attack reveals practical key recovery attacks, particularly on higher parameter sets of Kyber KEM, owing to higher number of STAMPs for higher security levels. Our attack on the reference implementation of Kyber1024 results in perfect key recovery and message recovery, while our attack on the optimized implementation yields a very small brute force complexity of 2^5 and 0 for key recovery and message recovery respectively. We also report practical attacks for the other parameter sets of Kyber KEM, and our experiements clearly indicate higher success rate for higher parameter sets of Kyber KEM.

4. We also analyze the applicability of our attack to masking and shuffling countermeasures. We show that generalized masking can not be used to defend against our attack but only increase the offline searching space. Like for all single-trace attacks, shuffling countermeasure provides concrete protection against our attack.

5. Our work demonstrates an additional side-channel vulnerability in the key-generation and encryption 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 provide a concise overview of the two stages involved in the proposed attack. In Section. 4, we demonstrate the first phase of our attack for evaluating the side channel leakages from STAMPs. In Section. 5, we provide the novel Key enumeration method as our second phase to recover the secret key. 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, we denote the polynomial ring $\mathbb{Z}_q[x]/\phi(x)$ as R_q, where $\phi(x) = x^n + 1$. The variables and functions are written in italic style. A polynomial in R_q is denoted with regular letters as $a \in R_q$, and the i^{th} coefficient of this polynomial can be denoted as $a[i]$. A vector of polynomials in R_q is represented by lowercase bold letters (i.e.) $\boldsymbol{a} \in R_q^k$. A

matrix of polynomials in R_q is denoted by italicizes bold uppercase letters as $\boldsymbol{A} \in R_q^{k \times k}$, with each polynomial in the i^{th} row and j^{th} column of the matrix denoted as $A_{i,j}$. The expression $\boldsymbol{A} \circ \boldsymbol{s}$ represents the pointwise multiplication of a polynomial with a polynomial vector.

2.2 M-LWE Problem

Majority of the Lattice-based schemes including the NIST PQC Standard Kyber KEM is based on variants of the Learning With Error (LWE) problem. In a standard LWE problem, a matrix $\boldsymbol{A}$ is sampled uniformly at random in $\mathbb{Z}_q^{m \times n}$ and the secret $\boldsymbol{s} \in \mathbb{Z}_q^n$ and error $\boldsymbol{e} \in \mathbb{Z}_q^m$ are sampled from a narrow Gaussian distribution respectively. Subsequently, a vector $\boldsymbol{b} \in \mathbb{Z}_q^m$ is computed as $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$ $(mod\, q)$. The search LWE problem requires the attacker to recover $\boldsymbol{s}$ from the LWE instance $(\boldsymbol{A}, \boldsymbol{b})$. However, it should be noted that performing operations on matrices require significant memory resources due to security requirements in large dimensions.

Structured variants of the LWE problem such as Ring LWE (R-LWE) and Module LWE (M-LWE) problem were proposed to enhance the performance of LWE based cryptographic schemes. In structured LWE problems, the fundamental element is the polynomial ring as $R_q = \mathbb{Z}_q[x]/\phi(x)$. We focus on the M-LWE problem as Kyber is based on the M-LWE problem. In the M-LWE problem, $\boldsymbol{s}$, $\boldsymbol{e}$ are polynomial vectors sampled from R_q^k with small coefficients, where k denotes the number of polynomials in a single vector. The polynomial matrix $\boldsymbol{A}$ is uniformly sampled from $R_q^{k \times k}$. The security level of the M-LWE problem can be modified simply by changing value of k, while retaining the same underlying polynomial arithmetic operations performed over R_q. While the theoretical security of the M-LWE problem increases with increasing k, our research demonstrates that larger values for k may leak additional side channel information and lead to insecure implementations.

2.3 Kyber

Kyber is an IND-CCA2 secure key encapsulation mechanism (KEM), based on the M-LWE problem. In its core, it contains an IND-CPA-secure public key encryption (PKE) scheme that can encrypt messages of a fixed length of $n = 256$ bits. It employs the well-known Fujisaki-Okamoto (FO) transform, to convert the IND-CPA secure Kyber PKE scheme into an IND-CCA secure KEM, that is used to establish a shared session key between two parties. The polynomial ring underlying Kyber is $R_q^k = \mathbb{Z}_q[x]/(x^n + 1)$, with $q = 3329$ and $n = 256$. Kyber has three security levels, Kyber-512 (NIST Level 1), Kyber-768 (NIST Level 3) and Kyber-1024 (NIST Level 5), each associated with a parameter $k = 2, 3, 4$, respectively. The simplified version of IND-CPA secure PKE scheme of Kyber is described in Algorithm. 1. We do not describe IND-CCA secure Kyber KEM, as it is not the focus of our attack.

In Algorithm. 1, the function Parse, XOF, CBD, and PRF functions are used to sample random polynomials. The functions Encode and Decode are used to transform the representation of variables in R_q or R_q^k into binary data (without any data loss). However, the functions Compress (resp. Decompress) are used to lossily compress (decompress) a given polynomial from a given modulus p to another modulus q, one coefficient at a time. While we do not provide in-depth description of these functions, we refer the reader to the specification document in [SAB+22].

2.3.1 Number Theoretic Transform(NTT)

This paper primarily focuses on the pointwise multiplication operation performed in the key generation procedure and encryption procedure of IND-CPA secure Kyber PKE scheme. The polynomial multiplication operation in Kyber KEM is carried out using the Number

Algorithm 1 Simplified Kyber.CPA.PKE

1: **procedure** CPAPKE.KeyGen
Input: None
Output: Public Key pk, Secret Key sk
2: $(\rho, \sigma) = G(d)$ ▷ randomly sampled d
3: $\hat{\boldsymbol{A}} = \mathsf{Parse}(\mathsf{XOF}(\rho))$
4: $(\boldsymbol{s}, \boldsymbol{e}) = \mathsf{CBD}(\mathsf{PRF}(\sigma))$
5: $\hat{\boldsymbol{s}} = \mathsf{NTT}(\boldsymbol{s}), \hat{\boldsymbol{e}} = \mathsf{NTT}(\boldsymbol{e})$
6: $\hat{\boldsymbol{w}} = \hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}}$
7: $\hat{\boldsymbol{t}} = \hat{\boldsymbol{w}} + \hat{\boldsymbol{e}}$
8: $(pk, sk) = \mathsf{Encode}(\hat{\boldsymbol{t}}||\rho), \mathsf{Encode}(\hat{\boldsymbol{s}})$
9: **end procedure**
10:
11: **procedure** CPAPKE.Encrypt
Input: Public Key pk, Message $m \in \{0,1\}^*$, Random Seed $r \in \{0,1\}^*$
Output: Ciphertext ct
12: $\hat{\boldsymbol{A}} = \mathsf{Parse}(\mathsf{XOF}(\rho))$ ▷ $\rho \leftarrow pk$
13: $\boldsymbol{r}, \boldsymbol{e_1}, e_2 = \mathsf{CBD}(\mathsf{PRF}(r)), \hat{\boldsymbol{r}} = \mathsf{NTT}(\boldsymbol{r})$
14: $\hat{\boldsymbol{w'}} = \hat{\boldsymbol{A}} \circ \hat{\boldsymbol{r}}$
15: $\hat{\boldsymbol{w''}} = \hat{\boldsymbol{t}} \circ \hat{\boldsymbol{r}}$ ▷ $\hat{\boldsymbol{t}} \leftarrow pk$
16: $\boldsymbol{u} = \mathsf{NTT}^{-1}(\hat{\boldsymbol{w'}}) + \boldsymbol{e_1}$
17: $v = \mathsf{NTT}^{-1}(\hat{\boldsymbol{w''}}) + e_2 + \mathsf{Decompress}(m)$
18: $ct = (c_1||c_2) = \mathsf{Encode}(\mathsf{Compress}(\boldsymbol{u}, v))$
19: **end procedure**
20:
21: **procedure** CPAPKE.Decrypt
Input: Secret key sk, Ciphertext ct
Output: Message m
22: $(\boldsymbol{u}, v) \leftarrow \mathsf{Decode}(ct)$
23: $\hat{\boldsymbol{u}} = \mathsf{NTT}(\boldsymbol{u})$
24: $m = \mathsf{Compress}(v - \mathsf{NTT}^{-1}(\hat{\boldsymbol{s}} \circ \hat{\boldsymbol{u}}))$
25: **end procedure**

Theoretic Transform (NTT), which is an integer variant of the well-known Discrete Fourier Transform (DFT) over a prime field. Within the key generation procedure, we target leakage from the pointwise multiplication operation of $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ and $\hat{\boldsymbol{s}} \in R_q^k$ in the NTT domain (Line 6 in CPAPKE.KeyGen). Within the encryption procedure, we target leakage from the pointwise multiplication operation between $\hat{\boldsymbol{A}} \in R_q^{k \times k}$, $\hat{\boldsymbol{t}} \in R_q^k$ and $\hat{\boldsymbol{r}} \in R_q^k$ in the NTT domain (Line 14 and 15 in CPAPKE.Encrypt), respectively. In the following, we will cover background on the NTT operation. In Kyber, NTT of a polynomial f in R_q is represented as:

$$NTT(f) = \hat{f} = (\hat{f}_0 + \hat{f}_1 X, \hat{f}_2 + \hat{f}_3 X, ..., \hat{f_{254}} + \hat{f_{255}} X)$$

When multiplying polynomials f and g, both of them are converted into the NTT domain as $\hat{f}$ and $\hat{g}$ respectively, and they are then pointwise multiplied as $\hat{h} = \hat{f} \circ \hat{g}$ in the following manner:

$$\hat{h}_{2i} + \hat{h}_{2i+1} X = (\hat{f}_{2i} + \hat{f}_{2i+1} X)(\hat{g}_{2i} + \hat{g}_{2i+1} X) mod X^2 - \zeta^{2\boldsymbol{br}(i)+1}$$

for $i \in \{0, (n/2-1)\}$. Consequently, the product of $h = f \cdot g$ can be obtained by computing the inverse NTT on $\hat{h}$.

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. These attacks have targeted several operations within the different procedures of Kyber KEM. Refer to [RCDB23] for a detailed survey on the various SCA performed on Kyber KEM. The polynomial multiplication operation has been the main target for several side-channel attacks, and on a high level, these attacks can be split into two categories - (1) Multi-trace attacks and (2) Single-trace attacks. We will now look into attacks targeting the polynomial multiplication reported in literature.

2.4.1 Multi-Trace Attacks on Polynomial Multiplication

Multi-trace attacks are only applicable to the decapsulation procedure of Kyber KEM, since the attacker can observe manipulation of the same secret key for multiple executions. In this respect, 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 (Line 22 in CPAPKE.Decrypt procedure in Alg.1). They demonstrated the possibility to perform full key recovery (in simulations) with only $k \in [2, 4]$ traces. Apart from the NTT operation, Mujdei *et al.* [MWK+22] proposed a Correlation Power Analysis (CPA) targeting the pointwise multiplication operation between the NTT transformed secret $\hat{\boldsymbol{s}}$ and the NTT transformed ciphertext component $\hat{\boldsymbol{u}}$ (Line 23 in CPAPKE.Decrypt). They were able to perform full key recovery in about 200 traces using their proposed CPA approach. Recently, Yang *et al.* [YWY+23] also proposed a CPA style attack targeting the pointwise multiplication operation in a chosen-ciphertext setting, requiring a few hundred traces to perform full key recovery.

2.4.2 Single-Trace Attacks on Polynomial Multiplication

Kyber KEM received more attention from single trace attacks compared to multi-trace attacks. One reason is that many sensitive variables are generated randomly and used only once, which indicates that the attacker cannot obtain enough traces. This particularly applies to the key-generation and encryption procedure, which are probabilistic in nature. Single-trace attacks aim to exploit maximum available information corresponding to multiple variables from a single side-channel trace, while multi-trace attacks exploit leakage corresponding to a single variables over multiple traces.

In 2017, Primas *et al.* [PPM17] proposed the first single trace attack based on the Soft Analytical Side-Channel Attack (SASCA) strategy, targeting the inverse NTT operation in the decryption procedure to recover its input (i.e.) $\hat{\boldsymbol{s}} \cdot \hat{\boldsymbol{u}}$, whose knowledge can be used to recover the secret key $\boldsymbol{s}$ (Line 23). They build templates using leakage from all intermediate variables computed in the INTT operation, and then they build a factor graph for the INTT operation in which the obtained leakage information is integrated, and solved using the Belief Propagation (BP) algorithm to find the input to the INTT operation. While this attack required 100 million traces to build the required templates for the attack, the same authors in [PP19] subsequently improved the attack to only utilizing a few hundred templates, and also an improved BP algorithm for key recovery. They also expanded the attack surface to exploit leakage from the NTT operations on the secret $\boldsymbol{s}$ in the key generation procedure (Line 5) and the ephemeral secret key $\boldsymbol{r}$ (Line 13) in the encryption procedure. Recently, 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.

There have also been single trace attacks targeting other types of polynomial multiplication and in this respect, Aysu *et al.* [AAT+21] demonstrated a single trace attack

exploiting horizontal leakage from the school-book multiplication operation in FrodoKEM. More recently, Bock *et al.* [BBB+23] proposed a template attack targeting the pointwise multiplication operation from the decryption procedure for key recovery. However, their attack requires either $q = 3329$ or $q^2 \approx 11$ million templates to perform full key recovery, and utilization of such high number of templates for a single trace attack, questions the applicability of their attack in a practical setting.

Those prior works are tabulated and categorized in Table. 1, and the target procedures decryption, encryption and key generation are denoted as Dec, Enc, KeyGen respectively.

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

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

2.5 Motivation

From Table. 1, we observe that there are several works targeting the NTT in the Decryption or Encryption procedure with single trace, while leakage from the pointwise multiplication operation has been exploited by multiple trace attacks. A natural question that arises is whether there are any additional vulnerabilities from the pointwise multiplication operation in the key generation or encryption procedure. So in this paper, we investigate the susceptibility of key generation and encryption procedure to single-trace attacks.

In this work, we bridge this gap by demonstrating the feasibility of single-trace attacks targeting the pointwise multiplication in the key generation and encryption procedure in Kyber KEM. Due to the M-LWE structure of Kyber KEM, we observe that the secret key is manipulated multiple times (i.e.) k times within a single execution of the key generation procedure and encryption procedure. We demonstrate that an attacker can exploit this inherent behaviour of the M-LWE problem, and exploit multiple repeated leakages of the sensitive variables, to perform efficient key recovery attacks. As a result, we show that Kyber at higher theoretical security levels with a higher value of k exhibits more leakage compared to Kyber at lower theoretical security levels. Throughout this work, we refer to these repeated key-dependent computations with different time-stamp, which are called as STAMPs in this work.

3 Side-Channel Attack on STAMPs

In this work, we start by demonstrating the inherent behaviour of the M-LWE problem that results in the existence of STAMPs (i.e.) repeated leakage of sensitive key-dependent variables within the pointwise multiplication operation. Subsequently, we demonstrate our novel attack methodology that can exploit the obtained leakage information from the STAMPs, to perform key recovery and message recovery attacks. Refer to Algorithm 2 for the pseudo-code of our proposed attack. Our attack works in two phases.

Phase 1: Building Side-Channel Templates for STAMPs: In the first phase of our attack, we use the classical template matching technique to exploit repeated leakage of the input variables of the pointwise polynomial multiplication operation. This enables us to build templates with higher accuracy on the secret coefficient $\boldsymbol{s}$, compared to templates from a single operation. Moreover, we also exploit leakage from several other intermediate variables within the pointwise multiplication, to maximize the leakage information available for key recovery.

Phase 2: Key Enumeration: In the second phase of our attack, we employ a novel key enumeration technique that combines the available leakage information from multiple repeated operations in a simpler manner compared to the traditional BP algorithm, to perform key recovery. We show that the success rate of our attack can be enhanced with higher security levels of Kyber KEM, since the higher dimension of the M-LWE problem results in more side-channel leakage and constraints of the secret key.

Adversary Model

The attacker targets the key-generation procedure of Kyber KEM to recover the secret key sk, or targets the encryption procedure of Kyber KEM to recover the message m from the ciphertext ct. We assume the following attacker capabilities:

- Physical access to DUT performing key-generation and encryption procedure for power/EM measurements.
- Access to a clone device that runs the key-generation and encryption procedure of Kyber KEM, and also obtain access to any intermediate variable within the target procedure.

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 probabilities for HW(s) $\leftarrow$ Template attack on STAMPs(l) ▷ Phase 1
2: $\hookrightarrow$ Template Matching on Collected Traces from Targeted Device
3: $\hookrightarrow$ Increasing the accuracy of F_l by leveraging leakage from STAMPs
4: Specific value of $s \leftarrow$ Key Enumeration(F_l,$\boldsymbol{A}$) ▷ Phase 2
5: $\hookrightarrow$ Use Factor Graph to search the Correct Value of s within F_l
6: $\hookrightarrow$ Decreasing the Searching Space by additional information (multiple $\boldsymbol{A}$)

We first explain the first phase of our attack below, which involves building side-channel templates for the STAMPs, followed by the second phase of our attack, in which we use our novel key enumeration algorithm for practical key recovery and message recovery attacks.

4 Phase 1: Building Side-Channel Templates for STAMPs

4.1 Analyzing PointWise-Multiplication for Repeated Secret Leakage

In this work, we utilize Kyber-1024 ($k = 4$) as a representative parameter set of Kyber for our analysis. Each row i of matrix $\boldsymbol{A}$ consists of k polynomials represented by $\boldsymbol{A}_i$, and the polynomial in row i and column j is denoted as $A_{i,j}$. The result of multiplication between a polynomial matrix $\boldsymbol{A} \in R_q^{k \times k}$ and a polynomial vector $\boldsymbol{s} \in R_q^k$ is indicated as $\boldsymbol{p} \in R_q^k$.

This multiplication operation between the $\boldsymbol{A}$ and $\boldsymbol{s}$ is implemented in k steps using the *polyvec_basemul* function as follows:

$$polyvec\_basemul(\boldsymbol{A}_0, \boldsymbol{s}) = A_{0,0} \circ s_0 + A_{0,1} \circ s_1 + A_{0,2} \circ s_2 + A_{0,3} \circ s_3 = p_0 \quad (1)$$
$$polyvec\_basemul(\boldsymbol{A}_1, \boldsymbol{s}) = A_{1,0} \circ s_0 + A_{1,1} \circ s_1 + A_{1,2} \circ s_2 + A_{1,3} \circ s_3 = p_1 \quad (2)$$
$$polyvec\_basemul(\boldsymbol{A}_2, \boldsymbol{s}) = A_{2,0} \circ s_0 + A_{2,1} \circ s_1 + A_{2,2} \circ s_2 + A_{2,3} \circ s_3 = p_2 \quad (3)$$
$$polyvec\_basemul(\boldsymbol{A}_3, \boldsymbol{s}) = A_{3,0} \circ s_0 + A_{3,1} \circ s_1 + A_{3,2} \circ s_2 + A_{3,3} \circ s_3 = p_3 \quad (4)$$

Due to the nature of the M-LWE problem, we observe that every polynomial of $\boldsymbol{s}$ is involved in the computation k times. The coefficients of each polynomial of $\boldsymbol{s}$ are loaded from memory k times during the pointwise polynomial multiplication operation (highlighted in red). These four loading operations, which we refer to as STAMPs, can be observed at independent time instances in a single execution. We can combine this repeated leakage to reduce the effect of noise and improve the success rate of template matching of the STAMPs.

Applicability of Analysis to Encryption Procedure: While the attacker can observe leakage from STAMPs for k manipulations of the secret key in the key-generation procedure, the same analysis can also be applied to the encryption procedure. In this respect, the attacker can target the pointwise multiplication operations within the encryption procedure to recover the ephemeral secret $\boldsymbol{r} \in R_q^k$, which enables trivial recovery of the message from the ciphertext. The attacker can observe leakage from two pointwise multiplication operations that manipulate the NTT of the ephemeral secret $\boldsymbol{r} \in R_q^k$: (1) $\hat{\boldsymbol{w}'} = \hat{\boldsymbol{A}} \circ \hat{\boldsymbol{r}}$ (Line 14 of CPAPKE.Encrypt procedure in Alg.1) and (2) $\hat{\boldsymbol{w}''} = \hat{\boldsymbol{t}} \circ \hat{\boldsymbol{r}}$ (Line 15 of CPAPKE.Encrypt procedure in Alg.1), where $\hat{\boldsymbol{t}} \in R_q^k$. Compared to the key-generation procedure that leaks through k pointwise multiplication operations, the attacker can observe leakage from an additional pointwise multiplication operation. Taking advantage of this additional information, this ensures that attacking the encryption procedure for message recovery for $k = k_0$, is equivalent to attacking the key-generation procedure for $k = k_0 + 1$.

4.1.1 Analysing the Operation of BaseMul Function

Each of the pointwise multiplication operations between the single polynomials, is computed using multiple applications of the BaseMul function, on pairs of coefficients of the input polynomials. We now perform a detailed analysis of the BaseMul function. The BaseMul function takes the following inputs: two coefficients together denoted as $A[2]$ from $A_{0,0}$ and two coefficients together denoted as $s[2]$ from s_0 that have the same index as $A[2]$ are given as input to the BaseMul function. It generates two coefficients together denoted as $r[2]$ (i.e.) $r[0]$, $r[1]$ of the output polynomial r. Within the BaseMul function, another function called FqMul is used to compute the product of two coefficients and subsequently compute the montgonmery reduction to obtain the final product. The pseudo-code of the BaseMul function is shown in Algorithm. 3.

We will first start with the analysis of the reference implementation, after which we will separately analyse the assembly optimized implementation in Section 4.2.2. A closer observation of the BaseMul function reveals that, every secret coefficient (i.e.) $s[0]$ and $s[1]$ is utilized by two FqMul functions (i.e.) $s[0]$ is manipulated by FqMul functions in Lines 6 and 9, and $s[1]$ is manipulated by FqMul functions in Lines 4 and 7 (highlighted in red in Algorithm 3). Despite being contained within a single instance of the BaseMul function, two distinct loading operations for s_0 exist for two FqMul when compiled at the -O3 optimization level. In a single pointwise multiplication operation between $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ and $\hat{\boldsymbol{s}} \in R_q^k$, a single secret polynomial is involved in k polynomial multiplication operations (BaseMul). However, in every BaseMul function, every secret coefficient is manipulated two

Algorithm 3 Pseudo Code of BaseMul and FqMul Function

1: **procedure** BASEMUL($\&r[2], \&A[2], \&s[2], zeta$) ▷ $\&A[2]$ represents address of two coefficients from A
2: Input: First two coefficients of $A_{0,0}$(A[2]), s_0(s[2]) and a constant(zeta)
3: Output: The result of basemul function(r[2])
4: $r[0]_1 = \mathsf{FqMul}(A[1], s[1])$ ▷ *load* $s[1]$, *store* $r[0]_1$
5: $r[0]_2 = \mathsf{FqMul}(r[0]_1, zeta)$ ▷ *store* $r[0]_2$
6: $r[0]_3 = r[0]_2 + \mathsf{FqMul}(A[0], s[0])$ ▷ *load* $s[0]$, *store* $r[0]_3$
7: $r[1]_1 = \mathsf{FqMul}(A[0], s[1])$ ▷ *load* $s[1]$, *store* $r[1]$
8: $r[1]_2 += \mathsf{FqMul}(A[1], s[0])$ ▷ *load* $s[0]$, *store* $r[1]$
9: Return $r[0], r[1]$
10: **end procedure**
11: **procedure** FQMUL(a, b)
12: $m = \mathsf{montgomery\_reduce}(a * b)$
13: Return m
14: **end procedure**

times. Thus, in total, every secret coefficient is manipulated $2 * k$ times. For Kyber1024, it totals upto 8 times for a single pointwise polynomial multiplication operation. Thus, we can build templates on these 8 loading operations for more accurate templates with lower noise.

Apart from building templates for loading of the secret coefficients due to STAMPs, we also observe storage of the final output of the BaseMul function $r[0]_3$ (Line 6) and $r[1]_2$ (Line 8), which are highlighted in blue. Moreover, we can also observe storage of the intermediate variables $r[0]_1$ (Line 4), $r[0]_2$ (Line 5) and $r[1]_1$ (Line 7), which are also dependent on the secret coefficients, that are highlighted in green. The CPA values of these intermediate variables also exhibit a comparable level of significance as loading operations. Thus, we can also build templates for all these intermediate and output variables of the BaseMul function, to enhance the success rate for practical attacks. We remark that the compiler could have optimized some of these repeated memory load and store operations of the intermediate variables, but we observed that the compiler did not optimize them out, and this enabled us to leverage the leakage of these intermediate variables.

4.1.2 Evaluating Impact of STAMPs on Simulated Leakage

We start by validating the presence of additional leakage due to STAMPs, using ELMO [MOW17], which simulates leakage for an ARM Cortex-M0 processor. We generate noise-free simulated traces for the BaseMul function from the reference implementation of Kyber KEM, compiled based on the O3 optimization level.

In order to limit the number of templates, we choose to build templates on the Hamming Weight (**HW**) of the data rather than its specific value. This is also because we only target leakage from the loading and storing of the sensitive variables within the target operation, rather than the whole computation.

We now analyze four BaseMul functions ($k = 4$) that manipulate the same secret coefficients $s[2]$ (corresponding to the first two coefficients of secret polynomial s_0), but different coefficients of A, corresponding to the first two coefficients of $A_{0,0}$, $A_{1,0}$, $A_{2,0}$ and $A_{3,0}$ respectively. The generated simulated trace for these four *basemul* operations is shown in Figure 1, where each point on the trace corresponds to a single assembly instruction. In this figure, each BaseMul function is framed in red. The upward peaks represent the memory load operations while the downward peaks represent the memory store operations.

Those simulation results preliminarily demonstrate existence of the leakage targets.

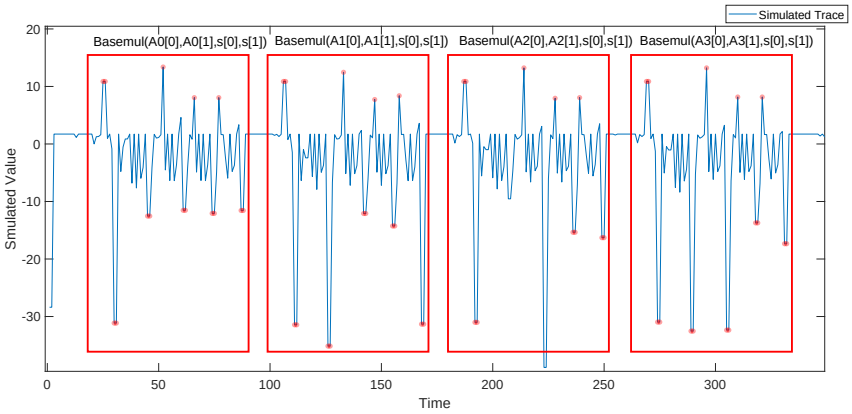


Figure 1: The Simulated Trace for Four Basemul Function

We then build templates on those simulated traces to verify the success rate of template matching on the $2 * k$ loading operations corresponding to the secret key coefficients. We add Gaussian noise with an incremental standard deviation σ to imitate the real traces and then build templates for the Hamming Weight of the secret coefficients in the NTT domain (i.e.) for values between $[0, 3329]$, using leakage from the memory load operation of the secret coefficients.

Refer to Table 2 for the matching accuracy of our template attack, utilizing leakage from different number of BaseMul operations, and different noise levels. The results clearly demonstrate increased success rate for the HW templates, with increasing number of BaseMul functions. These observations can also be reasoned from a more theoretical perspective, wherein targeting on multiple points results in a more solid noise covariance matrix, used for building templates, than that on a single point.

While we demonstrated construction of improved templates for the loading operation of secret coefficients, the other intermediate variables $r[0]_1$, $r[0]_2$, $r[1]_1$ (highlighted in green in Algorithm.3) and the final results $r[0]_3$ and $r[1]_2$ of the BaseMul operation (highlighted in blue in Algorithm.3) are only manipulated once, and thus the accuracy for storing these variables corresponds to the result for a single BaseMul function in Table.2.

Table 2: Template Matching accuracy for k Loading Operations in Simulation

σ of Noise	SR of 4 *basemul*	SR of 3 *basemul*	SR of 2 *basemul*	SR of 1 *basemul*
0.1	100.00%	99.87%	98.49%	96.28%
0.2	99.26%	98.21%	95.13%	89.56%
0.3	96.33%	93.77%	89.64%	83.08%
0.4	92.64%	88.54%	85.18%	76.56%
0.5	88.51%	83.97%	80.41%	71.64%

4.2 Practical Side-Channel Leakage Experiments

In this section, we perform experimental validation of our proposed attack using real side-channel leakage, obtained from the reference implementation and optimized implementation of Kyber KEM.

Experiment Setup

We choose the ChipWhisperer CW308-STM32F3 based on the STM32F3 microcontroller as our target board, and rely on power side-channel measurements for our experiments. We perform experiments on the reference and assembly optimized implementations of Kyber KEM (Kyber1024 parameter set) from the well known *pqm4* [KPR+] open-source library for benchmarking PQC on ARM Cortex-M4 microcontrollers. A +20dB amplifier is used to amplify the power side-channel traces, and we use a KeySight DSOX3034T oscilloscope to capture the power traces. We will first detail our experiments on the reference implementation of Kyber KEM in section 4.2.1, followed by our analysis of the assembly optimized implementation of Kyber KEM in section 4.2.2.

4.2.1 Leakage Analysis of Reference Implementation

Refer to Figure 2(a) for a sample trace corresponding to the target k BaseMul functions corresponding to a single pair of secret coefficients $s[0]$ and $s[1]$.

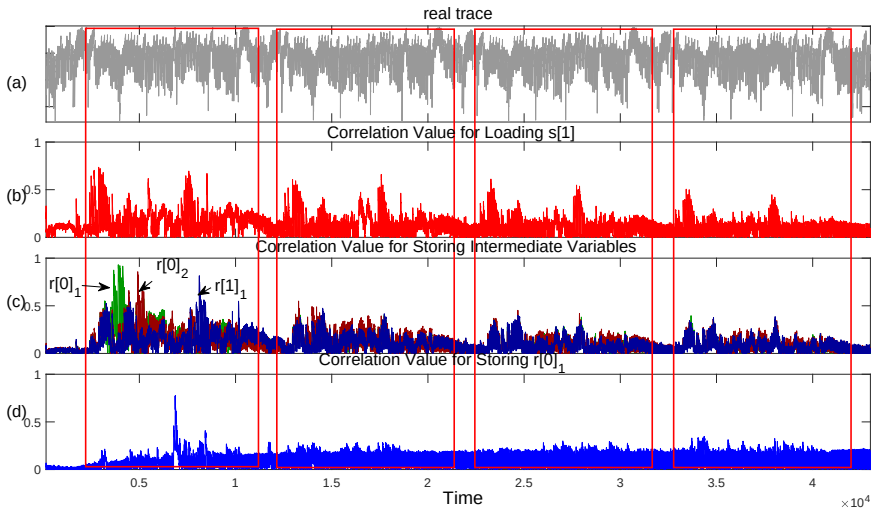


Figure 2: Correlation Location of Loading/Storing VS. Real Trace for Ref Implementation

Firstly, we observe that the repeating patterns of instructions in the power trace (gray) are not as distinct as those in the simulated trace (Figure 1). However, we can utilize Simple Power Analysis (SPA), to divide the trace into $k = 4$ (four) segments, highlighted in red. Each segment enclosed by a red rectangle represents one instance of the BaseMul function.

To locate the points corresponding to the different target variables, we utilize Correlation Power Analysis on the template traces and choose those points with the highest correlation value above a certain empirical threshold (parameter of the experimental setup). The correlation plots for the different targeted variables within the BaseMul function is shown in Figure 2. Firstly, Figure 2(b) indicates the correlation plot for loading of the secret coefficient $s[1]$, where we can clearly observe two load operations within a single BaseMul function. Apart from leakage due to the memory load operations, we also observe leakage corresponding to the computations. However, we observe that the correlation peaks diminish in height with increasing number of BaseMul functions. Similarly, Fig.2(c) shows the overlaid correlation plots of the intermediate variables $r[0]_1$, $r[0]_2$ and $r[1]_1$, that are stored in memory during the BaseMul computation. Fig.2(d) shows the correlation plot for the final output $r[0]_3$ of the BaseMul computation.

Accuracy of HW Template Matching

A higher dimension of the module k for parameter sets of Kyber with higher security, results in more side-channel leakage for the secret coefficients, which has been verified through simulations in Table 2. In the following, we validate the same using real side-channel leakage experiments on the reference implementation of Kyber KEM compiled with the O3 optimization level. We then demonstrate the ability to recover the HW of the targeted variables for different parameter sets (i.e.) $k = 2$ to $k = 4$. We use a set of 500 traces to bulid templates for each HW and assess the matching accuracy on 1000 traces.

We use the notation $\mathsf{HW}_{\mathsf{GE}=0}$ to indicate the likelihood that the correct HW has a Guessing Entropy (GE) of 0 (i.e.) first rank among all HW candidates. We use the notation $\mathsf{HW}_{\mathsf{GE}<5}$ to indicate the likelihood that the correct HW has a Guessing entropy of less than 5 (i.e.) among top five candidates. In this way, we consider the candidates whose HW is ± 2 of the correct HW. Table 3 shows $\mathsf{HW}_{\mathsf{GE}=0}$ and $\mathsf{HW}_{\mathsf{GE}<5}$ for different parameter sets of Kyber KEM, for a single BaseMul function (indicated as $k = 1$) up to four BaseMul functions for $k = 4$. We observe that $\mathsf{HW}_{\mathsf{GE}=0}$ starts from 84% for $k = 1$ and increases upto 96% for $k = 4$. However, $\mathsf{HW}_{\mathsf{GE}<5}$ is already at a high 98% for $k = 1$ and increases up to 99.92% for $k = 4$. Thus, we can clearly see that a large value of k significantly increases the success rate of template matching. While this demonstrates the improved leakage of the secret coefficients (due to multiple load instructions), the success rate for the other target variables which are manipulated only once, corresponds to that shown for $k = 1$.

Table 3: Template Matching Results for Real Traces

k	4	3	2	1[1]
$\mathsf{HW}_{\mathsf{GE}=0}$	96.39%	95.41%	93.35%	83.93%
$\mathsf{HW}_{\mathsf{GE}<5}$	99.92%	99.02%	98.86%	98.39%

4.2.2 Leakage Analysis of Optimized Implementation

We also studied leakage from the assembly optimized implementation of Kyber KEM, based on the implementations reported in [KPR+][2]. Refer to Algorithm 4 which shows the assembly code snippet corresponding to the target BaseMul function.

We observed that the reference implementation involves multiple loading operations of the secret coefficients and they are manipulated one coefficient at a time. However, the assembly optimized implementation takes advantage of the 32-bit registers, and loads two secret coefficients (signed 16-bit integers) into a single register in a single load instruction (Line 2 in Algorithm 4). Moreover, the output $r[0]_3$ is computed using a single assembly instruction (smulwt instruction in Line 5), instead of multiple FqMul functions as in the reference implementation. Thus, we observe that a pair of secret coefficients are only loaded once within the BaseMul function, and both coefficients are also loaded together. This reduces the number of leaky operations by half (i.e.) k compared to $2 \cdot k$ leakages for the reference implementation. Refer to Figure 3(b) for the correlation plot corresponding to the combined loading of the two secret coefficients $s[0]$ and $s[1]$. It can be seen that there exists a more pronounced pattern in the optimized implementation, compared to the reference implementation. However, we can only observe the combined HW of the two secret coefficients. We list the $\mathsf{HW}_{\mathsf{GE}=0}$ and $\mathsf{HW}_{\mathsf{GE}<5}$ of our template attack on optimized implementation in Table 4. In this experiment, we use 900 traces for each HW and test the success rate on 1000 traces.

[1] Only take points for one *basemul* to build templates and test the success rate.

[2] Our analysis and experiments were carried out on the implementation of Kyber-1024 corresponding to the commit hash 1eeb74e4106a80e26a9452e4793acd6f191fe413 in the *pqm4* library

Algorithm 4 Assembly optimized Basemul Function

1: **procedure** BASEMUL_ASM_CACHE_16_32
2: ldr \poly0, [\aptr], #4 ▷ Load $(s[1]||s[0])$
3: ldr \poly1, [\bptr] ▷ Load A[0]||A[1]
4: ldr \zeta, [\zetaptr], #4 ▷ Load ζ
5: smulwt \tmp, \zeta, \poly0 ▷ Compute $s[1] \cdot \zeta$
6: smlabb \tmp, \tmp, \q, \qa ▷ Reduce $s[1] \cdot \zeta$
7: pkhbt \tmp, \poly0, \tmp ▷ Pack $(s[1] * \zeta)||s[0]$
8: str \tmp, [\aprimeptr], #4 ▷ Store $(s[1] * \zeta)||s[0])$
9: smultt \tmp2, \tmp, \poly1 ▷ Compute $s[1] \cdot \zeta \cdot A[1]$
10: smlabb \tmp2, \poly0, \poly1, \tmp2 ▷ Compute $r[0] = s[1] \cdot \zeta \cdot A[1] + s[0] \cdot A[0]$
11: smuadx \tmp, \poly0, \poly1 ▷ Compute $r[1] = s[1] \cdot A[0] + s[0] \cdot A[1]$
12: str.w \tmp, [\rptr_tmp, #4] ▷ Store $r[1]$
13: str \tmp2, [\rptr_tmp], #8 ▷ Store $r[0]$
14: **end procedure**

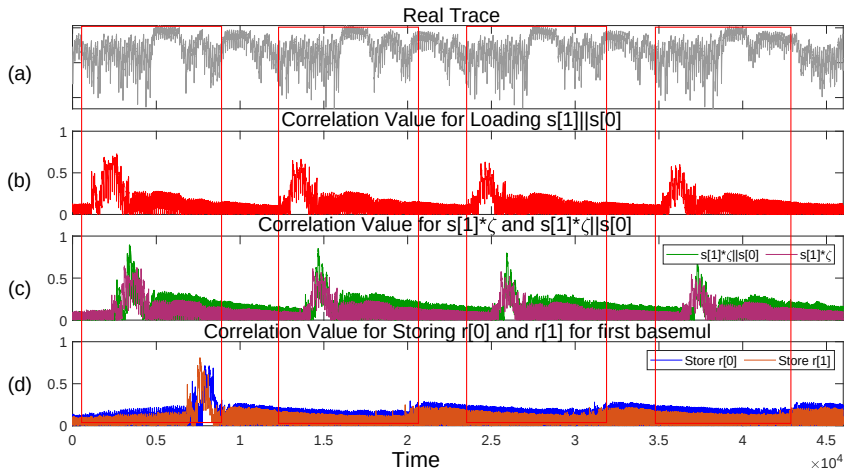


Figure 3: Correlation Location of Loading Secret Key VS. Real Trace for Optimized Implementation

Table 4: Accuracy of Template Matching for pqm4 Traces

k	4	3	2	1
$\mathsf{HW}_{\mathsf{GE}=0}$	91.93%	87.26%	81.60%	75.45%
$\mathsf{HW}_{\mathsf{GE}<5}$	99.95%	99.04%	98.83%	97.48%

Apart from the combined HW leakage of secret key coefficients $s[0]$ and $s[1]$, we also observe leakage of intermediate variable $s[1] \cdot \zeta$ (Line 5) and storing $(s[1] * \zeta)||s[0])$ (Line 8) in the first BaseMul, where ζ is a public constant known to the attacker. The variable $(s[1] * \zeta)||s[0])$ then will be loaded every time in later BaseMul. We observe a correlation of ≈ 0.85 for $s[1] \cdot \zeta||s[0]$ in each BaseMul function, and also obtain a $\mathsf{HW}_{\mathsf{GE}<5} \approx 99\%$, clearly indicating leakage of this intermediate variable. Combing these two additional leakages, we can get side channel information equivalent to targeting reference implementation but without $r[0]_1$, $r[0]_2$ and $r[1]_1$. Moreover, we also observe leakage due to storing the final outputs of the BaseMul function $r[0]$ and $r[1]$ (i.e.) $r[0]_3$ and $r[1]_2$. Here we only plot $r[0]$ and $r[1]$ for first BaseMul.

We have thus demonstrated the leakage from the different types of intermediate vari-

ables the attack can obtain for key recovery in the second phase of our attack. More importantly, we observe more leakage of intermediate variables in the reference implementation, compared to the optimized implementation. However, in the following section, we show that there is enough leakage available that allows for practical key recovery and message recovery attacks on both of these implementations.

5 Phase 2: Key Enumeration

In this section, we explain our novel key enumeration approach, which can be used to recover the value of the secret key coefficients, and in particular, increased number of STAMPs, reduces the searching space of the secret key coefficients significantly.

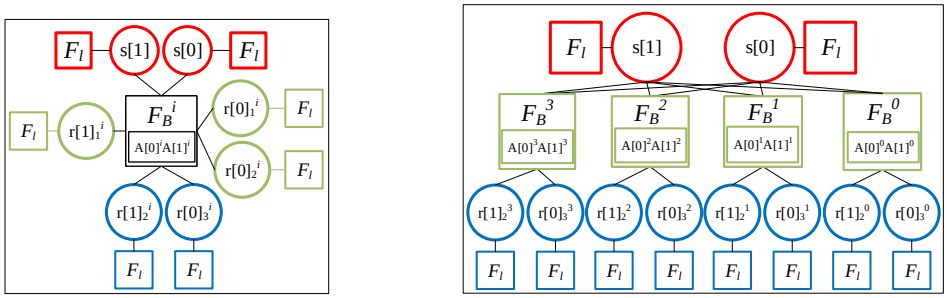
Several works have attempted to utilize the BP algorithm in the second phase of the attack to combine the leakage information to identify the key with the highest rank. However, using BP algorithm comes with several practical problems, e.g., the convergence of the BP algorithm, careful choice of parameters and the large memory usage. It is known that imperfect factor graphs will make BP harder to converge. Considering the aforementioned issues, we attempt to provide a more simpler key enumeration algorithm to recover the secret coefficients.

5.1 Factor Graph for BaseMul Function

The factor Graph is the fundamental part of Belief Propagation and the detailed description of BP can be found in [Mac03]. In a 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.

The factor graph we use for our attack is represented in Figure 4, and it shows the relation between the different variables whose leakage is observed using side-channels. Since one of the inputs to the pointwise multiplication operation $\hat{\boldsymbol{A}}$ is part of the public key, we assume the attacker's knowledge of the same. There are two kinds of factor nodes in this figure, F_l indicates the side channel leakage of the connected variable. As stated earlier, our templates are built on the HW of data, so F_l indicates the rank of possible HW of the connected variable nodes. Every possible candidate for this coefficient with the same HW will have the same initial probability. Our factor graph contains another factor node F_B, which is explained in Equation. 5, where i indicates the index of multiple STAMPs.

$$F_B^i = \begin{cases} p(s[0], s[1], A[0], A[1], r[0], r[1] | F_l) & r[0], r[1] = basemul(s[0], s[1], A[0], A[1]) \\ 0, & else \end{cases} \tag{5}$$



(a) Factor Graph for one basemul

(b) Factor Graph for k basemul

Figure 4: Factor Graph for one basemul and k basemul

When the inputs and outputs of the BaseMul function satisfy the computation, then the side-channel related posterior joint probability distribution is assigned for these candidates. A value of 0 is assigned to that candidate that does not satisfy this relation.

As can be seen in our factor graph in Figure 4, the variable nodes are highlighted in different colors in the following manner. We remark that we utilize the same notations as used in the pseudo-code in Algorithm 3. The input coefficients $s[0]$ and $s[1]$ are marked in red, indicating leakage from their multiple memory load operations. The outputs $r[0]_3$ and $r[1]_2$ of the BaseMul operation are marked in blue. The intermediate variables $r[0]_1$, $r[0]_2$ and $r[1]_1$ are marked in green. We mark the F_B in green to represent these intermediate variables for conciseness in Figure 4(b). Despite being stored only once, we aim to recover their HW values, through template attack on the storing operation. The coefficients of $\mathbf{A}$ relevant to the factor nodes F_B are marked appropriately, and these values are assumed to be known to the attacker.

5.2 Attack Methodology

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

In contrast to the complexity of NTT, which contains $n \cdot logn$ interconnected butterfly units, our target factor graph corresponding to the BaseMul function is relatively simple and only consists of k connected F_B functions. Consequently, there is no need to utilize message scheduling in Belief Propagation for this scenario. Instead, the joint probability distribution of candidates can be directly deduced using Equation 5. In the following, we explain the detailed steps of our key enumeration algorithm to recover a pair of secret coefficients, combining leakage from k BaseMul functions, and the same process can be repeated to recover the full secret key $\hat{\boldsymbol{s}}$ in the NTT domain two coefficients at a time. Once the attacker recovers $\hat{\boldsymbol{s}}$, the secret key $\boldsymbol{s}$ can be easily computed by simply computing the inverse NTT operation over $\hat{\boldsymbol{s}}$. The detailed steps of our key enumeration algorithm are listed below:

1. Reserve the top five most probable HW values in F_l for later enumeration, and we denote this set as HW_5, and one set each is generated for every targeted sensitive variable in the BaseMul function. The choice of top five values is motivated by our side-channel experiments shown in Section 4.2.1, where we observe that the correct HW value is present in the top five possible candidates with a very high probability. The initial probability of HW has been assigned in Phase 1, and all the candidate values with same HW will share the same initial probability.

2. Examine every possible value of the tuple $(s[0], s[1])$ that satisfies the set HW_5 and compute the output tuple $(r[0], r[1])$ for the k BaseMul functions. Assign the initial probabilities of those outputs based on information from F_l, that is connected to the output variable node.

3. Update the probabilities of $s[0]$ and $s[1]$ by multiplying the joint probabilities for the k BaseMul functions based on Equation 5 as: $\prod_{i=1}^{k} p(F_B^i)$.

4. Identify the correct candidate according to the largest updated probabilities.

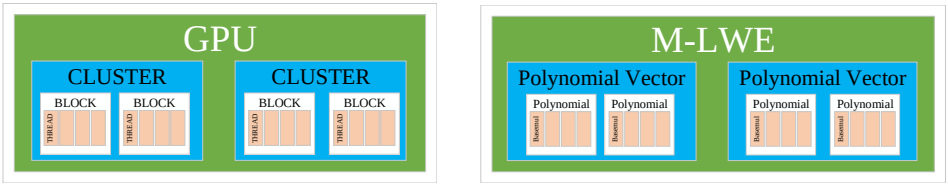
Considering more computations for F_B results in a decrease in the number of incorrect candidates for the tuple $(s[0], s[1])$, since each additional computation imposes constraints on the input value. Based on Step 3, the probabilities for the candidates for the tuple $(s[0], s[1])$ are updated only when all of the k $p(F_B^i)$ are non-zero, because only those

variable tuples that satisfy all k F_B^i conditions in Figure 4(b) can be considered as potential candidate. We can adopt a greedy approach, where in upon observance of $p(F_B^i) = 0$ for a given candidate tuple, we disregard leakage information from the subsequent BaseMul functions ($i+1$ to k) to reduce the amount of calculation.

5.2.1 Accelerating Key Enumeration using GPU

We observe that leakage from four BaseMul functions are used to recover the corresponding secret tuple $(s[0], s[1])$, and these tuples can be recovered in an independent manner, in a divide and conquer approach for full key recovery. This therefore enables parallelization to accelerate our attack and in this section, we explain our implementation of the key enumeration phase using the NVIDIA GPU based on the CUDA toolkit.

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 independent *threads*. *Threads* form thread *blocks* within the GPU framework, with each block accommodating up to 1024 individual threads. These *blocks* can also serve as units within a larger *cluster*. The architecture of GPU can be depicted as Figure 5(a).



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

Figure 5: Structure Comparison of GPU and M-LWE Problem

Acceleration Methodology: We observe that the M-LWE problem can be described in the same manner as the GPU architecture, as shown in Figure. 5(b). We can assign each block to recover one polynomial containing 256 coefficients, with 128 threads operating in parallel, each used to recover a corresponding tuple of the secret coefficients. Additionally, GPU can also be utilized to eliminate wrong candidates, in case the attack requires offline key search. The possible guesses for a secret polynomial can be loaded into separate threads to run the inverse NTT on each polynomial to check the distribution of coefficients in the secret key.

One primary advantage of our approach is that the time required for key recovery is not affected by the security level of the M-LWE problem. This is because a higher security level simply indicates more polynomials in the secret key. With the help of GPU, additional polynomials can be recovered by simply configuring more blocks in parallel.

Our parallelized implementation of the key-enumeration algorithm only takes two minutes for a single polynomial with 256 coefficients. For comparison, [PP19] reported a time of 8 minutes for a single iteration of the Belief Propagation algorithm. This clearly demonstrates the improved speed of our attack, compared to the BP algorithm.

5.3 Experimental Evaluation of Key Recovery

In order to test the effectiveness of our approach, we performed experiments on set of four BaseMul functions, to recover the corresponding coefficient tuple $(s[0], s[1])$. The attacker can independently target sets of four BaseMul functions to recover the entire secret key

two coefficients at a time, in a divide and conquer approach. We analyze the possibility of recovering a single coefficient tuple $(s[0], s[1])$, for random values of the input $A[0]$, $A[1]$. We observed that even after incorporating leakage information from all the targeted intermediate variables, we were not able to distinguish between certain candidates for $s[0]$ and $s[1]$, as they all satisfied the obtained conditions in the factor graph. We refer to the set of candidates for $(s[0], s[1])$, which cannot be distinguished as Collision_Set. The presence of Collision_Set for the secret coefficient tuples denotes that the attacker has to perform offline brute-force key search to recover the correct key.

Table 5: Brute-Force Complexity for Key Search for the reference and assembly optimized Implementations for all parameter sets of Kyber KEM

Implementation		Reference			Optimized		
Kyber Parameter Set		1024	768	512	1024	768	512
Total Searching Space	CPAPKE.KeyGen	0	2^3	2^{30}	2^5	2^{40}	$\sim$
	CPAPKE.Encrypt	0	0	2^3	0	2^5	2^{40}

We ran our attack to recover 50 random secret key polynomials of Kyber with different parameter sets, and since all the secret polynomials of Kyber can be recovered independently, we evaluated the offline brute-force search required to recover a single secret key polynomial, whose results are shown in Table 5 for different parameter sets of Kyber KEM. This number can then be used to evaluate the key search complexity to recover the entire secret key of Kyber.

We separate discuss our results for the reference as well as optimized implementations.

Reference Implementation: Referring to results for the key generation procedure in Table 5, we observe that the brute-force search for Kyber512 is about 2^{30} for a single polynomial, and hence recovering the entire secret key amounts to 2^{60}, which is still within the brute-force capability of 2^{64}, which is considered to be practical. However, for the higher parameter sets of Kyber768 and Kyber1024, we observe only a negligible brute-force complexity for the secret key polynomial, which clearly demonstrates the ability of our attack to exploit leakage from STAMPs involving repeated manipulation of the secret key. The results for the encryption procedure are better compared to the key generation procedure for every parameter set of Kyber, and this is primarily due to existence of additional leakage corresponding to the ephemeral secret in the encryption procedure.

Optimized Implementation: Referring to results for the key generation procedure in Table 5, the total searching space for Kyber512 is too large, and so it is not listed here. However, the searching space reduces significantly for higher values of k, with 2^{40} for Kyber768, and only 2^5 for Kyber1024. Similar to the case for the reference implementation, our attack on the encryption procedure is better for all parameter sets of Kyber, compared to the key generation procedure.

Thus, we can clearly observe that the attack on the reference implementation remains practical for all parameter sets of Kyber, except for Kyber512, primarily due to exploitation of leakage of multiple intermediate variables within the target BaseMul computation. However, in case of the optimized implementation, the reduced number of intermediate variables for exploitation results in a larger brute-force complexity for key search, for the optimized implementation.

6 Applicability to SCA Countermeasures

Thus, we have clearly demonstrated the ability of our attack exploiting leakage from STAMPs, for key recovery and message recovery attacks. In the following, we discuss the applicability of our attack to the masking and shuffling countermeasures for Kyber KEM.

6.1 Masking

Masking is widely used for mitigating the threat from SCA attacks. In the masked implementation, secret variables are partitioned into multiple shares and processed by carefully devised functions, respectively. These shares will be re-combined together to compute the actual output after the masked domain. The attackers will do not have sufficient side channel information to recover every share on the masked implementation in the traditional SCA methods, especially for CPA and DPA.

6.1.1 Generalized Masking

In lattice-based cryptography, masking gadgets are commonly designed for compress/decompress, sampling and polynomial comparison functions. As both NTT and pointwise multiplication are linear functions, they can be directly applied to each share. In this section, we explore whether our attack remains effective when this masking technique is employed during Key Generation or Encryption procedures.

We note that the side channel information still exist if the shares of one polynomial are **not** changed and regenerated for different BaseMul. For example, if one coefficient s_i is divided 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, which means we can recover the coefficients of s'_i and s''_i separately.

Whereas, for attacking masked version, the searching space for one coefficient equals to the product of the searching spaces for s'_i and s''_i. We conducted 50 tests on the masked Kyber-768 implementation[HKL+], an open source library for [HKL+22b], using the same method as described in section 4.2.2 to evaluate its searching space. However, the assembly implementation of BaseMul from *mkm4* [HKL+] has fewer leakage than the implementation from *pqm4* [KPR+], they use pkhtb instruction to combined the $r[0]$ and $r[1]$ and only use one storing instruction, which caused an exponential increasing number of searching space. The average searching space required to attack the encryption procedure in Kyber-768 equals to $2^{19.6}$ on this masked and optimized implementation. In terms of the success rate, we get an overall success rate as 90% for recovering a complete polynomial, influenced by individual success rates on each share. However, for key generation in [HKL+], the running time required for searching would be impractical, since the effectiveness of our attack on the unmasked assembly version of key generation in Kyber-768 has already been quite weak.

6.1.2 Specific Masking

There is an alternative masking technique proposed by Ravi *et al.*[RPBC20] in 2020, specifically designed to address single trace attacks on NTT such as SASCA. The fundamental operation, known as the *butterfly unit*, within the NTT framework involves taking two inputs $(a, b) \in \mathbb{R}_q^2$ and a twiddle constant ω, resulting in two outputs $(c, d) \in \mathbb{R}_q^2$. The operation of one *butterfly unit* can be represented as:

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

$$d = a - b \cdot \omega \tag{7}$$

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

$$c' = c \cdot \omega_s \tag{8}$$
$$= (a + b \cdot \omega) \cdot \omega_s \tag{9}$$
$$= a \cdot \omega_s + b \cdot \omega \cdot \omega_s \tag{10}$$

Refer to line 3 in Algorithm. 3, the *zeta* also represents the twiddle constant used in NTT. In our attack on optimized version, we only build the templates on loading $s[1]||s[0]$, computing $s[1] * \zeta$ and storing $s[1] * \zeta||s[0]$ on line 2,5,8 in Algorithm. 4. So if the ζ is masked in line 5 as $\zeta \cdot \omega_s$, the intermediate variable $s[1] * \zeta$ will also be masked and we can not run the Key Enumeration just according to F_B as Equation 5. That means this masking method could defend against our single trace attack on optimized implementations.

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 and encryption procedure in Kyber KEM as key recovery and message recovery attack. Our attack mainly targets the BaseMul function, which is used for the polynomial multiplication in NTT domain. Our target operations, referred to as STAMPs, repeat k times caused by the structured M-LWE problem. We validate our attack on simulated traces by ELMO, and real traces of reference implementation and pqm4 implementation collected from STM32F3 board. The experimental results of our attack demonstrate that Kyber at higher theoretical security levels with a higher value of k exhibits more leakage compared to Kyber at lower theoretical security levels. Due to 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.

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 have more unnoticed leakages in the implementation.

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.

[BBB+23] Estuardo Alpirez Bock, Gustavo Banegas, Chris Brzuska, Łukasz Chmielewski, Kirthivaasan Puniamurthy, and Milan Šorf. Breaking dpa-protected kyber via the pair-pointwise multiplication. Cryptology ePrint Archive, Paper 2023/551, 2023. https://eprint.iacr.org/2023/551.

[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.

[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+] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Daan Sprenkels. First-order masked kyber on ARM Cortex-M4. https://github.com/masked-kyber-m4/mkm4.

[HKL+22a] 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.

[HKL+22b] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Daan Sprenkels. First-order masked kyber on arm cortex-m4. Cryptology ePrint Archive, Paper 2022/058, 2022. https://eprint.iacr.org/2022/058.

[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.

[NIS23] NIST. Post-quantum cryptography standardization. https://csrc.nist.gov/News/2023/three-draft-fips-for-post-quantum-cryptography, 2023.

[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.

[RCDB23] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D'Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. *ACM Trans. Embed. Comput. Syst.*, jun 2023. Just Accepted.

[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.