Single-Trace Attacks on the Message Encoding of Lattice-Based KEMs*

Bo-Yeon Sim[1][0000−0002−6446−1020], Jihoon Kwon[3], Joohee Lee[3], Ij-Ju Kim[2], Taeho Lee[2], Jaeseung Han[2], Hyojin Yoon[3], Jihoon Cho[3], and Dong-Guk Han[1,2]

[1] Department of Mathematics, Kookmin University, Seoul, Republic of Korea
{qjdusls,christa}@kookmin.ac.kr
[2] Department of Financial Information Security, Kookmin University, Seoul, Republic of Korea {kimij2905,20141932,jae1115}@kookmin.ac.kr
[3] Security Research Center, Samsung SDS, Inc., Seoul, Republic of Korea
{jihoon.kwon,joohee1.lee,hj1230.yoon,jihoon1.cho}@samsung.com

Abstract. We propose single-trace side-channel attacks against lattice-based KEMs, the current candidates of the NIST's standardization project. More specifically, we analyze the message encoding in the encapsulation of lattice-based KEMs to obtain the ephemeral session keys, concluding that a single trace leakage implies a whole key recovery: our implementation on a ChipWhisperer UFO STM32F3 target board shows 100% success rates for Crystals-Kyber and Saber regardless of optimization level, and more than a 79% success rate for FrodoKEM. We further show that our attack methodologies are not restricted to the above algorithms but widely applicable to other NIST PQC candidates, including LAC, NewHope, NTRU Prime, and NTRU.

Keywords: Side-channel attack · Lattice-based cryptography · Key encapsulation mechanism · Message encoding · Single-trace attack.

1 Introduction

The key encapsulation mechanism (KEM) is a public-key cryptosystem that aims key sharing between two parties and widely adopted in Internet protocols for secure communications. In 1994, Shor [46] proposed an algorithm that effectively defeats the underlying hard problems of all the widely adopted public-key cryptosystems such as RSA, Diffie-Hellman, and ECC using a quantum computer. Hence, the KEMs become vulnerable once a large-scaled quantum computation is available, and experts estimated that RSA with 2000-bit of public-key size becomes insecure by 2030 [13,34,35].

National institute of standards and technology (NIST) thus launched a project to standardize post-quantum cryptography (PQC). We have 26 candidates in the 2nd round of NIST PQC project, and 17 of them are KEMs [36]. Lattice-based

* We submitted the paper to Asiacrypt 2020 and received the final notification 16 Aug 2020.

KEMs [1,6,8,9,10,14,15,29] have attracted a great amount of attention due to the balanced performance in terms of sizes and speed. Lattice-based KEMs can be categorized into two classes: the schemes of which security is based on the hardness of some variant of the learning with errors (LWE) problem [41], and the schemes using NTRU problem [22] as their underlying hard problems. The first class includes FrodoKEM, NewHope, Crystals-Kyber, LAC, Saber, Round5, and Three Bears, which have its security reduction to the respective variants of LWE, and the second class consists of NTRU and NTRU Prime which has its strength in sizes. Both classes adopted optimization techniques such as number theoretic transform (NTT) and advanced vector extensions (AVX), achieving relatively low computational costs among the second round PQC candidates.

Side-channel attacks (SCAs) [27] allow to extract cryptographic keys using side-channel information such as power consumption, electromagnetic radiation, and execution time when cryptographic devices are in operation. Power analysis is a well known category of SCAs, including simple power analysis (SPA), differential power analysis (DPA), and template attack [12,27,28,33]. SPA recovers secret keys using only one or a few traces obtained from running cryptographic algorithms. DPA analyzes multiple power consumption traces using statistical methods, for example, evaluating the correlation between actual and hypothetical power models. Template attack is a type of profiling attack (PA) that builds a profile of a programmable device, which is identical to a target device, and exploits this profile to obtain a secret key. Machine learning techniques such as k-means clustering and support vector machine are also adopted to improve the performance of SCAs [7,21,24,31].

Related Works. Lattice-based cryptography is believed to have quantum resistance, but their practical implementations still have vulnerabilities against SCAs. Silverman and Whyte [47] presented a timing attack against NTRUEncrypt, which is based on variation in the number of hash calls in decryption. The timing leakage information of error-correcting codes in the decoding algorithm allows to extract the entire secret key of LAC [16]. Park and Han [38] reported an SPA attack against the decryption of the ring-LWE encryption scheme performed on an AVR processor.

DPA attacks on NTRU implementation are also discussed [4,30]. Aysu *et al.* [5] mounted horizontal DPA attacks on hardware implementations of NewHope and FrodoKEM, extracting secret keys with over 99% success rates using a single trace. Bos *et al.* [11] extended this attack considering ring-less LWE-based constructions by utilizing a single-trace template attack and reported experimental results for Frodo key exchange protocol, which is later updated to FrodoKEM. More DPA attacks targeting the multiplications of decryption phase in lattice-based PKE can be found in [37,42,43].

Primas *et al.* [40] proposed a single-trace template attack on the NTT of the ring-LWE decryption phase, extracting the entire secret key. Pessl and Primas [39] presented an improved single-trace attack on the NTT targeting an optimized constant-time implementation of Crystals-Kyber. Various power analysis attacks are applied to NTRU Prime [25], which includes vertical correlation

power analysis, horizontal in-depth correlation power analysis, online template attack, and chosen-input SPA. Amiet *et al.* [2] recently proposed a single-trace attack against NewHope, targeting the message encoding of the encapsulation phase.

If an adversary obtains a shared ephemeral session key by exploiting vulnerabilities of KEM, it could eavesdrop all the communications encrypted using the key. When attacking the encapsulation phase of KEM, however, it would be only possible to use a single trace since it encrypts a random secret message every time. Several types of research exist to recover the random secret message using a single trace [2,5,11], but studies based on vulnerabilities in the message encoding of the encapsulation phase are sparse [2].

Our Contributions. In this work, we give a comprehensive analysis of several state-of-the-art lattice-based KEMs by targeting the message encoding operation of the encapsulation phase. The main contributions of this paper can be summarized as below.

1. **Novel single-trace attacks on Crystals-Kyber, Saber, and FrodoKEM**
   We propose single-trace attacks on the message encoding of the encapsulation phase. We target lattice-based KEMs of the second round candidates of the NIST PQC standardization project and divide attack methodologies into three types based on the computational characteristics of each algorithm. We present the attack methodologies and experimental results of Crystals-Kyber, Saber, and FrodoKEM as respective representatives for these three types. We demonstrate that our proposed attacks on Crystals-Kyber and Saber can recover the entire secret message with 100% success rates using only a single trace. Neither of them is affected by the optimization level. Hence, we can generate the shared ephemeral session key using the recovered secret message and public values. As for FrodoKEM, it is possible to recover the secret message with a success rate of more than 79%.

2. **Application to other 2nd round lattice-based KEMs**
   We show how the three types of single-trace attacks can be applied to other lattice-based KEMs of NIST PQC standardization. Since there are similar vulnerabilities in LAC, NewHope, Streamlined NTRU Prime, NTRU LPRime, and NTRU as in Crystals-Kyber, the proposed single-trace attack on Crystals-Kyber thus can be applied to them. As for Streamlined NTRU Prime and NTRU, they also have similar vulnerabilities with Saber, and it is thus possible to further extract the secret message by combining the attack methodologies which are applied to Crystals-Kyber and Saber.

3. **Countermeasures**
   We recommend countermeasures that increase the attack complexity. The countermeasure proposed by Amiet *et al.* [2], combining masking and shuffling schemes, can be applied not only to NewHope but also to Crystals-Kyber.

We present modifications of the existing countermeasure [2] to be applied to the computational structure of Saber and FrodoKEM.

Technical Overview. Our attack strategy depends on the existence of an intermediate value in the targeted reference C implementation, which is determined by a sensitive bit value. We name this intermediate value as determiner. It follows that the number of cases of the determiner is 2. In particular, the determiner is defined in a context that the difference between the Hamming weights of two determiner values is greater than or equal to 2, where the Hamming weight is the number of 1-bit of the intermediate value stored in a register.

Definition 1. *The* determiner *is the intermediate value that is determined by a sensitive bit value, and the difference between the Hamming weights of the elements of the* determiner *domain is greater than or equal to 2. The cardinal number of the* determiner *domain is 2.*

We present three types of attack methodologies and provides experimental results on Crystals-Kyber, Saber, and FrodoKEM, respectively. The first type of attack makes use of the determiner and can be applied to Crystals-Kyber. In this case, the clustering-based attack is used because of the significant differences in the side-channel leakage. The second type of attack is targeting algorithms that scan one sensitive bit at a time during encoding operations, and it thus can be applied to Saber. The third type of attack is targeting algorithms that scan multiple sensitive bits at a time during encoding operations, and it thus can be applied to FrodoKEM. For the second and third types of attacks, we apply the machine learning-based profiling attack (ML-based PA) since there does not exist the determiner.

Organization. The rest of the paper is organized as follows. In Section 2, we briefly describe the basics of lattice theory. We describe our proposed single-trace attack methodologies and experiment results on three lattice-based KEMs in Section 3, Section 4, and Section 5. In Section 6, we discuss the applicability of our proposed attacks to other lattice-based KEMs and recommend countermeasures. We finally give a conclusion in Section 7.

2 Preliminaries

2.1 Notation

- All logarithms are base 2 unless otherwise indicated.
- For a positive integer q, we use $\mathbb{Z} \cap (-q/2, q/2]$ as a representative of $\mathbb{Z}_q$.
- We use $x \leftarrow D$ to denote sampling x according to the distribution D. It denotes the uniform sampling when D is a finite set.
- For two bitstrings v and w, $(v \parallel w)$ denotes their concatenation.
- For an ℓ-bit bitstring b of which the i-th most significant bit is b_i for all $1 \leq i \leq \ell$, we denote $b = (b_\ell, \cdots, b_1)_2$.

2.2 Lattice-Based Key Encapsulation Mechanism

KEM can be constructed by applying the Fujisaki-Okamoto generic transformation [18,23,45] on a CPA-secure public key encryption to have a CCA-security in (quantum) random oracle model. Most of lattice-based KEMs in the NIST PQC standardization are following this approach, especially using the PKE scheme LPR [32] proposed by Lyubashevsky, Peikert, and Regev as their core algorithms. We briefly discuss the LPR encryption and the message encoding algorithms as follows.

Let n and q be positive integers. Let $\mathcal{R}$ be a base ring that can be represented as

$$\left\{ \sum_{i=0}^{n-1} a_i x^i : a_i \in \mathbb{Z},\ 0 \le i \le n-1 \right\},$$

and $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$. For example, $\mathcal{R}$ can be set as $\mathbb{Z}[x]/\langle x^n+1\rangle$ or $\mathbb{Z}[x]/\langle x^n-x-1\rangle$. Let χ_s, χ_r, χ_e, and $\chi_{e'}$ be certain distributions over $\mathcal{R}$ such that an element sampled from any of the distributions has small coefficients as a polynomial, respectively. Let the message space of PKE scheme be Σ_n of n components.

The message encoder $\mathsf{encode} : \Sigma_n \to \mathcal{R}_q$ and decoder $\mathsf{decode} : \mathcal{R}_q \to \Sigma_n$ are functions satisfying $\mathsf{decode}(\mathsf{encode}(\mu) + e \bmod q) = \mu$ for any 'small enough' $e \in \mathcal{R}$.

- KeyGen: Choose a uniformly random $a \leftarrow \mathcal{R}_q$. Let $s \leftarrow \chi_s$, and $e \leftarrow \chi_e$. Compute $b \leftarrow -a \cdot s + e \bmod q$. Output the public key $pk = (a, b)$, and secret key $sk = (s, 1)$.
- $\mathsf{Enc}(pk, \mu \in \Sigma_n)$: Let $r \leftarrow \chi_r$, and $e_0, e_1 \leftarrow \chi_{e'}$. Compute $c_0 \leftarrow r \cdot a + e_0 \bmod q$ and $c_1 \leftarrow r \cdot b + \mathsf{encode}(\mu) + e_1 \bmod q$. Output the ciphertext $ct = (c_0, c_1)$.
- $\mathsf{Dec}(sk, ct)$: Compute and output $\mu' = \mathsf{decode}(\langle sk, ct\rangle \bmod q)$.

In the LPR encryption, errors e, e_0, and e_1 are sampled from discrete Gaussian distributions to be secure under the ring-LWE assumption. Moreover, a public key pk is of the form of ring-LWE sample with a secret s and an error e. Likewise, since $pk = (a, b)$ is close to uniform random (under the ring-LWE assumption), an encryption of zero forms two ring-LWE samples with secret r. The ring-LWE samples in both key generation and encryption can be altered with that of other security assumptions such as standard LWE, learning with rounding (LWR), module-LWE, etc.

The lattice-based KEMs which are the current NIST PQC candidates and use the LPR encryption as their core algorithms can be classified based on the security assumptions of LWE, ring-LWE, module-LWE, LWR, ring-LWR, module-LWR, and integer module-LWR as in Table 1. They use encoders that send the message bits to the most significant bits of the modulo q space to derive cryptographically negligible decryption failure rates.

Table 1: The LPR-like second round submissions to the NIST's PQC standardization process. In the fourth column, "Encoder" denotes the output of the algorithm encode for given μ in the respective plaintext spaces. q is the ciphertext modulus before applying any compression techniques and B, b, t, ϵ_p are integers such that $0 < \epsilon_p, B < \log q$ and $0 < b, t < q$. ECC denotes an encoding of error-correcting code.

Scheme	Security assumption	Base polynomial ring	Encoder
Crystals-Kyber	Module-LWE	$\mathbb{Z}[x]/\langle x^{256}+1\rangle$	$\lfloor (q/2)\cdot\mu \rceil$
FrodoKEM	LWE	-	$(q/2^B)\cdot\mu$
LAC	Ring-LWE	$\mathbb{Z}[x]/\langle x^n+1\rangle$	$\lfloor q/2 \rceil \cdot \mathsf{ECC}(\mu)$
NewHope	Ring-LWE	$\mathbb{Z}[x]/\langle x^n+1\rangle$	$\lfloor q/2 \rfloor \cdot \mu$
NTRU LPRime	NTRU	$\mathbb{Z}[x]/\langle x^n-x-1\rangle$	$((q-1)/2)\cdot\mu$
Round5	LWR, Ring-LWR	-, $\mathbb{Z}[x]/\langle\sum_{i=0}^{n} x^i\rangle$	$(t/b)\cdot\mu$
Saber	Module-LWR	$\mathbb{Z}[x]/\langle x^{256}+1\rangle$	$2^{\epsilon_p-1}\cdot\mu$
Three Bears	Integer Module-LWR	-	$8\cdot\mu$

3 Proposed Single-Trace Attack on Crystals-Kyber

We describe the message encoding operation of Crystals-Kyber and a single-trace attack methodology on it. Besides, we show experimental results when the algorithm is operating on ARM Cortex-M4 processors.

3.1 Message Encoding of Crystals-Kyber

Crystals-Kyber is based on a polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n+1\rangle$ of the dimension $n = 256$ and modulus $q = 2^{13} - 2^9 + 1 = 3329$. For NIST security level 1, the first component of the ciphertext is of rank 2 over $\mathcal{R}_q$, *i.e.*, $k = 2$ in the Algorithm 1, and p and t for compress are 2^{10} and 2^3, respectively. The bit length ℓ of a message μ and shared key K is 256. Hash1 and Hash2 are SHA3-256 and SHA3-512, respectively. KDF is implemented using SHAKE-256. $\mathsf{Compress}_{q,\log p}(x)$ and $\mathsf{Compress}_{q,\log t}(x)$ take an element $x \in \mathbb{Z}_q$ and output $\log p$- and $\log t$-bit integers, respectively. The Algorithm 1 and Listing 3.1 show the message encapsulation and message encoding of Crystals-Kyber, respectively. In the Listing 3.1, byte array msg is the secret message μ.

Attack Methodology. As shown in steps 11 to 12 of the Listing 3.1, the mask value is determined based on the sensitive bit μ_i value, where the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$. The mask value is whether `0x0000` or `0xffff`; thus, the number of cases of the mask value is 2. Moreover, the difference of the Hamming weight between `0x0000` and `0xffff` is 16. Therefore, based on the Definition 1,

Algorithm 1 Message Encapsulation of Crystals-Kyber (refer to [10])

Input : Public key $pk = (a \in \mathcal{R}_q^{k \times k}, b \in \mathcal{R}_q^k)$
Output : Ciphertext $c \in \mathcal{R}_p^k \times \mathcal{R}_t$, shared key $K \in \{0,1\}^\ell$

1: /*Generate random message*/
2: $\mu \leftarrow \{0,1\}^\ell$
3: $\mu \leftarrow \mathsf{Hash1}(\mu)$
4: $(\bar{K}, seed) = \mathsf{Hash2}(\mu \parallel \mathsf{Hash1}(pk))$
5: /*Encryption*/
6: Sampling $r, e_1 \in \mathcal{R}_q^{k \times 1}$, and $e_2 \in \mathcal{R}_q$ using $seed$
7: $c_1 = \mathsf{Compress}_{q,\log p}(ar + e_1 \bmod q)$
8: $c_2 = \mathsf{Compress}_{q,\log t}(b^\intercal r + e_2 + \mathsf{encode}(\mu) \bmod q)$
9: $c = (c_1 \parallel c_2)$
10: /*Shared key derivation*/
11: $K = \mathsf{KDF}(\bar{K} \parallel \mathsf{Hash1}(c))$
12: **Return** c, K

```
1  // Convert 32-byte message to polynomial
2  void poly_frommsg(poly *r, const unsigned char msg[KYBER_SYMBYTES])
3  {
4    int i,j;
5    uint16_t mask;
6
7    for(i = 1; i < KYBER_SYMBYTES; i++)
8    {
9      for(j = 0; j < 8; j++)
10     {
11       mask = -((msg[i] >> j) & 1);
12       r ->  coeffs[8 * i + j] = mask & ((KYBER_Q + 1) / 2);
13     }
14   }
15 }
```

Listing 3.1: Message Encoding $\mathsf{encode}(\mu)$ of Crystals-Kyber (in C code)

the mask value is a sensitive bit-dependent determiner. Traditionally, in software implementations, the power consumption model is based on the Hamming weight of the intermediate value. The power consumption traces can be classified into two sets depending on the Hamming weight of the mask value, because the number of cases of the mask value is 2. Accordingly, the power consumption property in steps 11 to 12 of the Listing 3.1 can be categorized as follows.

Property 1. In a software implementation, the power consumption is affected by the Hamming weight of the intermediate value. The mask *value is a 16-bit integer* $-\mu_i$*; therefore, when* $\mu_i = 0$ *it is* `0x0000`*, and the power consumption is proportional to* 0*. Contrariwise, when* $\mu_i = 1$*, the* mask *value is* `0xffff`*, and the power consumption is proportional to* 16*, which is the Hamming weight of the* mask *value.*

Algorithm 2 Single-Trace Attack on the Message Encoding of Crystals-Kyber

Input : A trace T
Output : message μ
for $i = \ell - 1$ down to 0 **do**
  Select points of interest p_i associated with μ_i
end for
Classify p_i into two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, using the clustering algorithm
Calculate the average values $E(\mathbb{G}_1)$ and $E(\mathbb{G}_2)$, respectively, of $\mathbb{G}_1$ and $\mathbb{G}_2$
for $i = \ell - 1$ down to 0 **do**
  if $p_i \in \mathbb{G}_1$ **then** ▶ assume that $E(\mathbb{G}_1) > E(\mathbb{G}_2)$
    $\mu_i \leftarrow 0$ ▶ $\mu_i = 0$ when it follows the Property 1
  else
    $\mu_i \leftarrow 1$ ▶ $\mu_i = 1$ when it follows the Property 1
  end if
end for
Return $(\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$

There are significant differences in performances of analyses depending on the position of the attack. Hence, it is important to find specific points of interest (PoIs). For each μ_i, we select the points where the mask value is calculated, stored, and loaded ($0 \leq i < \ell$). We define those points as the PoIs. After selecting the PoIs, classify the PoIs into two groups using clustering algorithms. The clustering algorithms aim to partition ℓ elements, *i.e.*, the ℓ power consumption traces, into k classes of similar elements. There are several clustering algorithms, such as k-means, fuzzy k-means, mean-shift, density-based spatial clustering of applications with noise (DBSCAN), expectation-maximization (EM) using Gaussian mixture models (GMM), and hierarchical [3,17,19,44]. Using one of the clustering algorithms, it is possible to classify the power consumption traces which are proportional to the Hamming weight of the mask value, *i.e.*, the PoIs, into two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$. Here, $\mathbb{G}_1$ and $\mathbb{G}_2$ denote each clustered group.

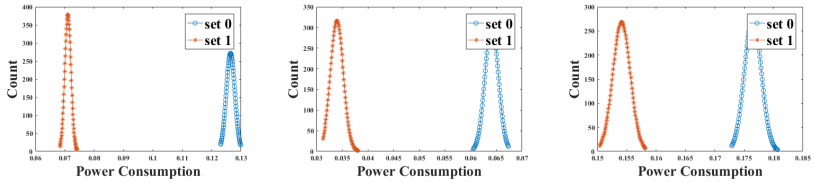
Because the power consumption is affected by the Hamming weight of the intermediate value, each average value of these groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are different. Therefore, if we assume that the larger the Hamming weight, the lower the power consumption, we can distinguish which μ_i value each group corresponds to based on the average value of two groups. This assumption is based on the design of ChipWhisperer-Lite mainboard, which uses for measuring the power consumption of the target board [25]. Thus, for example, the μ_i belonging to $\mathbb{G}_1$ is 0 and that belonging to $\mathbb{G}_2$ is 1 when $E(\mathbb{G}_1)$ is bigger than $E(\mathbb{G}_2)$, where $E(\mathbb{G}_1)$ and $E(\mathbb{G}_2)$ are the average values of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. As a results, we can recover the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ and generate secret shared key K. The Algorithm 2 describes the single-trace attack flow.

3.2 Experiment Results

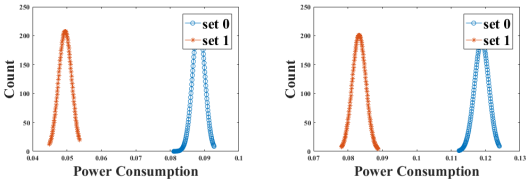
Our experimental results demonstrate that the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ can be extracted just using a single trace. We measured 500 power consumption

Table 2: Compiler option: optimization level

Optimization Level	Description
`-O0`	Turn off optimization
`-O1`	Optimize for speed (Low)
`-O2`	Optimize for speed (Medium)
`-O3`	Optimize for speed (High)
`-Os`	Optimize for size



(a) Optimization level 0 (b) Optimization level 1 (c) Optimization level 2



(d) Optimization level 3 (e) Optimization level s

Fig. 1: Distributions of the PoIs of the message encoding of Crystals-Kyber (set 0 and set 1 are $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively)

traces for different messages at 29.54 MS/s sampling rate when the Listing 3.1 is operating on the ChipWhisperer UFO STM32F3 target board [26]. Implementations are compiled with gcc-arm-none-eabi-6-2017-q2-update, and we use compiler options in Table 2.

To identify the PoIs, we calculate the sum of squared pairwise t-differences (SOST) [20] of the power consumption traces

$$SOST = \sum_{i,j=1}^{k} \left(\frac{E(\mathbb{G}_i) - E(\mathbb{G}_j)}{\sqrt{\frac{\sigma(\mathbb{G}_i)^2}{\#\mathbb{G}_i} + \frac{\sigma(\mathbb{G}_j)^2}{\#\mathbb{G}_j}}} \right)^2 \quad \text{for } i \geq j,$$

$E(\cdot)$, $\sigma(\cdot)$, $\#$, and k denote the mean, standard deviation, number of elements, and number of groups, respectively. We select points with high SOST values as the PoIs. The reader may refer to Appendix A.1 for a more detailed explanation.

Table 3: The averages and differences between two sets of Figure 1

Optimization Level	$E(\mathbb{G}_1)$	$E(\mathbb{G}_2)$	$\|\|E(\mathbb{G}_1) - E(\mathbb{G}_2)\|\|$	SOST value
`-O0`	1.264143e-01	7.105207e-02	5.237223e-02	237970
`-O1`	6.408548e-02	3.397242e-02	3.011306e-02	70684
`-O2`	1.764538e-01	1.541080e-01	2.234580e-02	29467
`-O3`	8.828891e-02	4.946987e-02	3.881904e-02	56015
`-Os`	1.188696e-01	8.322993e-02	3.563967e-02	38600

Figure 1 shows the distributions of the PoIs of 500 power consumption traces when the most significant bit $\mu_{\ell-1}$ is encoded. The distributions are distinctly separated irrespective of the optimization level. As shown in Table 3, when the optimization level is 0, *i.e.*, turn off optimization, the difference between $E(\mathbb{G}_1)$ and $E(\mathbb{G}_2)$ is the biggest. There is no error rate because the distributions of the PoIs are clearly distinguished, that is, we can recover the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ with a 100% success rate using the Algorithm 2. In this paper, we apply k-means clustering algorithm which is the most commonly used. As a result, we can generate the secret shared key K using the recovered μ, public key pk, and ciphertext c, as shown in the Algorithm 1.

4 Proposed Single-Trace Attack on Saber

We describe the message encoding operation of Saber and a single-trace attack methodology on it. Besides, we show experimental results when the algorithm is operating on ARM Cortex-M4 processors.

4.1 Message Encoding of Saber

Saber is based on a polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ of the dimension $n = 256$ and modulus $q = 2^{13}$. For NIST security level 1, the first component of the ciphertext is of rank 2 over $\mathcal{R}_q$, *i.e.*, $k = 2$ in the Algorithm 3, and the moduli p and t for rounding are 2^{10} and 2^3, respectively. The bit length ℓ of a message μ and shared key K is 256. Hash1 and Hash2 are SHA3-256 and SHA3-512, respectively. KDF is implemented using SHA3-256. $\mathsf{Rounding}_{q,p}(x)$ is rounding from a modulus q to modulus p. $\mathsf{Rounding}_{p,t}(x)$ is similarly defined. The Algorithm 3 and Listing 4.1 show the message encapsulation and message encoding of Saber, respectively. In the Listing 4.1, byte array message_received is the secret message μ.

Attack Methodology. As shown in steps 6 to 12 of the Listing 4.1, there is a sensitive bit μ_i identification phase to encode the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$. Notably, Saber scans one sensitive bit at a time during the message encoding.

Algorithm 3 Message Encapsulation of Saber (refer to [15])

Input : Public key $pk = (a \in \mathcal{R}_q^{k \times k}, b \in \mathcal{R}_p^k)$
Output : Ciphertext $c \in \mathcal{R}_p^k \times \mathcal{R}_t$, shared key $K \in \{0,1\}^\ell$

1: /*Generate random message*/
2: $\mu \leftarrow \{0,1\}^\ell$
3: $\mu \leftarrow \mathsf{Hash1}(\mu)$
4: $(\bar{K}, seed) = \mathsf{Hash2}(\mu \parallel \mathsf{Hash1}(pk))$
5: /*Encryption*/
6: Sampling $r \in \mathcal{R}_q^{k \times 1}$ using $seed$
7: $c_1 = \mathsf{Rounding}_{q,p}(ar \bmod q)$
8: $c_2 = \mathsf{Rounding}_{p,t}(b^\intercal(r \bmod p) - \mathsf{encode}(\mu) \bmod p)$
9: $c = (c_1 \parallel c_2)$
10: /*Shared key derivation*/
11: $K = \mathsf{KDF}(\bar{K} \parallel c)$
12: **Return** c, K

```
1  void indcpa_kem_enc(unsigned char *message_received, unsigned char *
       noiseseed, const unsigned char *pk, unsigned char *ciphertext)
2  {
3    uint16_t message[SABER_KEYBYTES*8];
4    // (omit)
5    // unpack message received
6    for(j = 0; j < SABER_KEYBYTES; j++)
7    {
8      for(i = 0; i < 8; i++)
9      {
10       message[8 * j + i] = ((message_received[j] >> i) & 0x01);
11     }
12   }
13
14   // message encoding
15   for(i = 0; i < SABER_N; i++)
16   {
17     message[i] = (message[i] << (SABER_EP - 1));
18   }
19   // (omit)
20 }
```

Listing 4.1: Message Encoding $\mathsf{encode}(\mu)$ of Saber (in C code)

Therefore, $\mathsf{message}[i] = \mu_{\ell-1-i}$ after steps 6 to 12 of the Listing 4.1, where $0 \leq i \leq \ell - 1$. In other words, each $\mathsf{message}[i]$ is 0 or 1. Since the moduli of Saber are the power of 2, steps 15 to 18 of the Listing 4.1 are the operation that shifts each sensitive bit μ_i by $p-1$ to the left. As a result, $\mathsf{message}[i]$ is whether 0x0000 or 0x0200 after steps 15 to 18 of the Listing 4.1. The number of cases of each $\mathsf{message}[i]$ is 2, however, the difference of the Hamming weight between cases of each $\mathsf{message}[i]$ is 1. Accordingly, the determiner does not exist, and the power

consumption property in steps 8 to 14 of the Listing 4.1 can be categorized as follows.

Property 2. The sensitive bit μ_i is 0 *or* 1*. Thus, if $\mu_i = 0$, the power consumption is proportional to* 0 *when extracting or manipulating the μ_i value. Likewise, if $\mu_i = 1$, then the power consumption is proportional to* 1*.*

The power consumption property in steps 15 to 18 of the Listing 4.1 is the same as steps 6 to 12 of the Listing 4.1. Unlike Crystals-Kyber, the difference between cases of each message$[i]$ is just 1; thus, we apply the ML-based PA. That is, we construct a template for each μ_i value in the profiling phase and calculate the probability that the power consumption trace belongs to each template in the extraction phase. After that, we find the μ_i value by selecting a template that has the highest probability to belong to. The ML-based PA automatically finds and combines specific PoIs by repeating the learning. Therefore, we divide the attack range into two areas, rather than selecting specific points; the computational range where μ_i is extracted, defined as first PoIs; the computational range where μ_i is shifted by $p - 1$ to the left, defined as second PoIs. As a results, we can recover the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ and generate secret shared key K.

4.2 Experiment Results

Our experiment shows that the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ can be extracted only using a single trace. We measured 10,000 power consumption traces for different messages at 29.54 MS/s sampling rate when the Listing 4.1 is operating on the ChipWhisperer UFO STM32F3 target board [26]. Implementations are compiled with gcc-arm-none-eabi-6-2017-q2-update, and we use compiler options in Table 2.

To identify the existence of the leakage based on the sensitive bit μ_i value, we calculate the SOST value of the power consumption traces. We divide the power consumption traces into two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, associated with $\mu_i = 0$ and $\mu_i = 1$, respectively. The reader may refer to Appendix A.2 for a more detailed explanation. Figure 2 and Figure 3 show the distributions of the PoIs of 500 power consumption traces when the most significant bit $\mu_{\ell-1}$ is encoded. Unlike Figure 1, the distributions are overlapped. On average, the SOST values in Table 5 and Table 6 are respectively 110 times and 40 times smaller than in Table 3. Accordingly, it is hard to perfectly divide into two groups by the clustering algorithms.

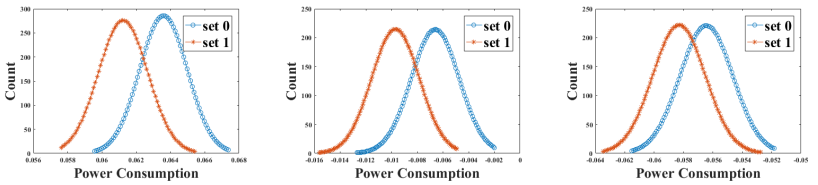
We construct network architecture as shown in Table 4 to apply the ML-based PA. x is the number of points of the PoIs and y is the number of classification labels. The power consumption based on the Property 2 occurs and information from multiple points is automatically combined while training. Therefore, we use three types of labels for training: 1-bit, 2-bit, and 8-bit values. In other words, we analyze in units of μ_i, $(\mu_i, \mu_{i-1})_2$, and $(\mu_i, \cdots, \mu_{i-7})_2$ at once for each type. Accordingly, y is 2, 4, and 256 for each label. We evaluate each ML model with 9-fold cross-validation; dividing a train and validation set into 8,000 and 1,000 traces, respectively. After each profiling, we carry out 1,000 single-trace attacks.

Table 4: ML-based PA

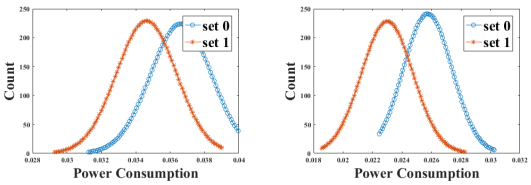
Layer	node (in, out)	kernel initializer
InputLayer	(x, x)	-
Batch Normalization	(x, x)	-
Dense	$(x, 32)$	he_uniform
Batch Normalization	$(32, 32)$	-
ReLU	$(32, 32)$	-
Dense	$(32, y)$	he_uniform
Softmax	(y, y)	-

* Input Normalization: all values are within the range of -1 and 1
* Loss function: categorical_crossentropy
* Optimizer: Nadam (lr=0.0001, epsilon=1e-08)
* Label encoding: one-hot encoding
* Batch size and epochs: 10 and maximum 200, respectively
* 9-fold cross-validation (training: 8000, validation: 1000)

In other words, the number of traces we attack is 1,000, and we show the average values of success rates of 9 profiled models in Table 4. Besides, the experiment results are the average values of the success rates of the attack for the first eight most significant bits $(\mu_{\ell-1}, \cdots, \mu_{\ell-8})_2$.



(a) Optimization level 0 (b) Optimization level 1 (c) Optimization level 2

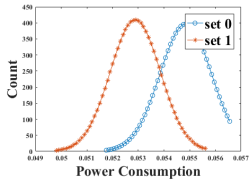


(d) Optimization level 3 (e) Optimization level s

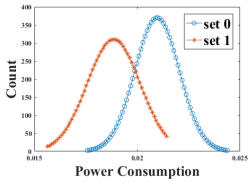
Fig. 2: Distributions of first PoIs of the message encoding of Saber (set 0 and set 1 are $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively)

Table 5: The averages and differences between two sets of Figure 2

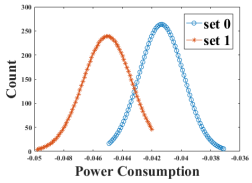
Optimization Level	$E(\mathbb{G}_1)$	$E(\mathbb{G}_2)$	$\|E(\mathbb{G}_1) - E(\mathbb{G}_2)\|$	SOST value
`-O0`	6.359049e-02	6.123422e-02	2.356270e-03	344
`-O1`	-6.598499e-03	-9.693878e-03	3.095379e-03	347
`-O2`	-5.644457e-02	-5.830823e-02	1.863660e-03	134
`-O3`	3.662884e-02	3.462072e-02	2.008120e-03	163
`-Os`	2.574050e-02	2.299686e-02	2.743640e-03	326



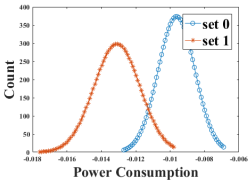
(a) Optimization level 0



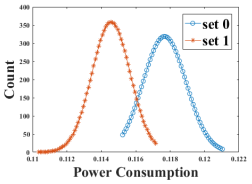
(b) Optimization level 1



(c) Optimization level 2



(d) Optimization level 3



(e) Optimization level s

Fig. 3: Distributions of second PoIs of the message encoding of Saber (set 0 and set 1 are $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively)

Table 6: The averages and differences between two sets of Figure 3

Optimization Level	$E(\mathbb{G}_1)$	$E(\mathbb{G}_2)$	$\|E(\mathbb{G}_1) - E(\mathbb{G}_2)\|$	SOST value
`-O0`	5.484788e-02	5.288561e-02	1.962270e-03	490
`-O1`	2.096737e-02	1.885364e-02	2.113730e-03	396
`-O2`	-4.135224e-02	-4.503360e-02	3.681360e-03	664
`-O3`	-9.629991e-03	-1.309303e-02	3.463039e-03	1025
`-Os`	1.176719e-01	1.145635e-01	3.108400e-03	865

On average, the SOST values in Table 6 are 2.6 times larger than that in Table 5. The cause is that more information is generated at the second PoIs

Table 7: The success rates of the single-trace attacks on Saber

Optimization	First PoIs				Second PoIs
Level	1-bit value	2-bit value	8-bit value	Combined	1-bit value
-O0	100%	-	-	-	100%
-O1	99.97%	99.89%	99.83%	99.97%	100%
-O2	99.90%	99.56%	99.53%	99.87%	100%
-O3	95.87%	83.12%	86.96%	96.71%	100%
-Os	99.99%	99.96%	99.96%	99.98%	100%

because the operation that shifts each sensitive bit μ_i by $p-1$ to the left. Therefore, in the case of using second PoIs, it is possible to extract each 1-bit value of $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ with a 100% success rate regardless of the optimization level (refer to Figure 9(b) and Figure 10(b) in Appendix A.2). Hence, we can generate secret shared key K using the recovered μ, public key pk, and ciphertext c, as shown in the Algorithm 3. Whereas, in the case of using first PoIs, it is hard to recover each 1-bit value of μ with a 100% success rate (refer to Figure 9(a) and Figure 10(a) in Appendix A.2). Accordingly, to find the 1-bit value μ_i, we combine the results of the 1-bit, 2-bit, and 8-bit value analysis, and apply the majority rule. Consequently, we can discover each 1-bit value μ_i with over a 96.71% success rate, and then apply an exhaustive search of candidates to recover the secret shared key K with a complexity 2^{11} ($\ell * 0.04 = 256 * 0.04 \approx 11$).

5 Proposed Single-Trace Attack on FrodoKEM

We describe the message encoding operation of FrodoKEM and a single-trace attack methodology on it. Besides, we show experimental results when the algorithm is operating on ARM Cortex-M4 processors.

5.1 Message Encoding of FrodoKEM

FrodoKEM is based on lattices over $\mathbb{Z}_q$. For NIST security level 1, the modulus $q = 2^{15}$ and the dimension $n = 640$. The secret key is consisted of $\bar{n} = 8$ vectors and the parameter $\bar{m} = 8$. The bit length ℓ of a message μ and shared key K is 128. Hash1 and KDF are SHAKE-128. The Algorithm 4 and Listing 5.1 show the message encapsulation and message encoding of FrodoKEM, respectively. In the Listing 5.1, in that is 16-bit integer array is the secret message μ.

Attack Methodology. As shown in steps 15 to 20 of the Listing 5.1, there is sensitive bits $(\mu_i, \mu_{i-1})_2$ identification phase to encode the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$. We denotes $(\mu_i, \mu_{i-1})_2$ as the *wvalue*. Notably, FrodoKEM

Algorithm 4 Message Encapsulation of FrodoKEM (refer to [9])

Input : Public key $pk = (a \in \mathbb{Z}_q^{n \times n}, b \in \mathbb{Z}_q^{n \times \bar{n}})$
Output : Ciphertext $c \in \mathbb{Z}_q^{\bar{m} \times n} \times \mathbb{Z}_q^{\bar{m} \times \bar{n}}$, shared key $K \in \{0,1\}^{\ell}$
1: /*Generate random message*/
2: $\mu \leftarrow \{0,1\}^{\ell}$
3: $(seed, \bar{K}) = \mathsf{Hash1}(\mathsf{Hash1}(pk) \parallel \mu)$
4: /*Encryption*/
5: Sampling $r, e_1 \in \mathbb{Z}^{\bar{m} \times n}$ and $e_2 \in \mathbb{Z}^{\bar{m} \times \bar{n}}$ using $seed$
6: $c_1 = ra + e_1 \bmod q$
7: $c_2 = rb + e_2 + \mathsf{encode}(\mu) \bmod q$
8: $c = (c_1 \parallel c_2)$
9: /*Shared key derivation*/
10: $K = \mathsf{KDF}(c \parallel \bar{K})$
11: **Return** c, K

```
1  void frodo_key_encode(uint16_t *out, const uint16_t *in)
2  {
3    // Encoding
4    unsigned int i, j, npieces_word = 8;
5    unsigned int nwords = (PARAMS_NBAR * PARAMS_NBAR) / 8;
6    uint64_t temp, mask = ((uint64_t) 1 << PARAMS_EXTRACTED_BITS) - 1;
7    uint16_t *pos = out;
8
9    for(i = 0; i < nwords i++)
10   {
11     temp = 0;
12     for(j = 0; j < PARAMS_EXTRACTED_BITS; j++)
13       temp|=((uint64_t)((uint8_t*)in)[i*PARAMS_EXTRACTED_BITS+j])<<(8*j);
14
15     for(j = 0; j < npieces_word; j++)
16     {
17       *pos=(uint16_t)((temp & mask)<<(PARAMS_LOGQ-PARAMS_EXTRACTED_BITS));
18       temp >>= PARAMS_EXTRACTED_BITS;
19       pos++;
20     }
21   }
22 }
```

Listing 5.1: Message Encoding $\mathsf{encode}(\mu)$ of FrodoKEM (in C code)

scans two sensitive bits at a time during the message encoding. Thus, the number of cases of the extracted sensitive bits *wvalue* is 4. Accordingly, the power consumption property in steps 15 to 20 of the Listing 5.1 can be categorized as follows.

Property 3. The wvalue is in $\{(00)_2, (01)_2, (10)_2, (11)_2\}$. *Thus, if wvalue* $= (00)_2$, *the power consumption related to* 0 *occurs when extracting or saving the*

wvalue value. Likewise, if wvalue is non-zero, then the power consumption is proportional to the Hamming weight of wvalue.

Similar to Saber, we apply the ML-based PA and select the computational range, where the *wvalue* is extracted, as the PoIs. As a results, we can extract the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ and generate secret shared key K.

5.2 Experiment Results

Our experiment shows that the message $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ can be recovered using a single trace. We measured 10,000 power consumption traces for different messages at 29.54 MS/s sampling rate when the Listing 5.1 is operating on the ChipWhisperer UFO STM32F3 target board [26]. Implementations are compiled with gcc-arm-none-eabi-6-2017-q2-update, and we use compiler options in Table 2.

To identify the existence of the leakage based on the sensitive bits *wvalue*, we calculate the SOST value of the power consumption traces. We divide the power consumption traces into four groups, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, and $\mathbb{G}_4$, associated with $wvalue = (00)_2$, $wvalue = (01)_2$, $wvalue = (10)_2$, and $wvalue = (11)_2$, respectively. The reader may refer to Appendix A.3 for a more detailed explanation. Figure 4 shows the distributions of the PoIs of 500 power consumption traces when the most significant consecutive two bits $(\mu_{\ell-1}, \mu_{\ell-2})_2$ are encoded. The four distributions according to the *wvalue* are overlapped, especially the distribution of $\mathbb{G}_2$ and $\mathbb{G}_3$ with the Hamming weight of 1 is almost the same. In case the optimization level is 0, *i.e.*, turn off the optimization, the SOST value is the largest because the distribution of $\mathbb{G}_2$ and $\mathbb{G}_3$ is distinguishable comparing to other optimization levels.

On average, the SOST values in Table 8 are 90 times smaller than in Table 3 in Section 3. Thus, we apply the ML-based PA which is the same analysis method used in Section 4, and labels for training are 2-bit, 4-bit, and 8-bit values. In other words, we analyze in units of $(\mu_i, \mu_{i-1})_2$, $(\mu_i, \cdots, \mu_{i-3})_2$, and $(\mu_i, \cdots, \mu_{i-7})_2$ at once for each type. Accordingly, y is 4, 16, and 256 for each label. Similar to the case using the first PoIs of Saber, it is hard to recover each *wvalue* with a 100% success rate (refer to Figure 12 in Appendix A.3). Hence, we combine the results of the 2-bit, 4-bit, 8-bit value analysis, and apply the majority rule.

Table 8: The averages and differences between two sets of Figure 4

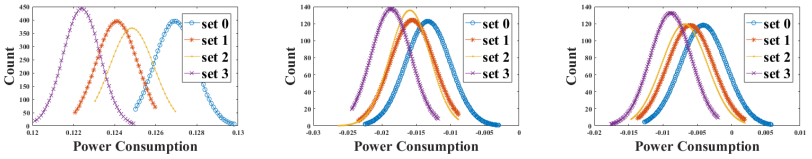
Optimization Level	$E(\mathbb{G}_1)$	$E(\mathbb{G}_2)$	$E(\mathbb{G}_3)$	$E(\mathbb{G}_4)$	SOST value
`-O0`	1.269287e-01	1.241036e-01	1.248361e-01	1.223927e-01	2635
`-O1`	-1.329385e-02	-1.557993e-02	-1.593018e-02	-1.867858e-02	395
`-O2`	-4.104393e-03	-6.006634e-03	-6.859564e-03	-8.867536e-03	281
`-O3`	8.939756e-02	8.808445e-02	8.822609e-02	8.636975e-02	481
`-Os`	-2.748210e-02	-3.013717e-02	-3.013611e-02	-3.290936e-02	353

Table 9: The success rates of the single-trace attacks on FrodoKEM

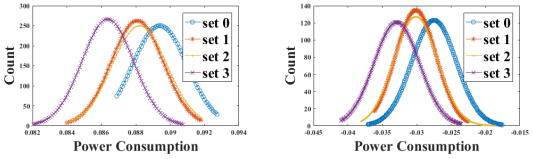
Optimization Level	2-bit value	4-bit value	8-bit value	2-bit HW
`-O0`	99.98%	99.93%	99.70%	99.96%
`-O1`	97.64%	95.50%	86.45%	99.66%
`-O2`	97.58%	99.84%	86.06%	94.88%
`-O3`	79.17%	73.60%	49.15%	85.75%
`-Os`	98.56%	96.95%	91.16%	99.90%

Optimization Level	Combined 1 (value)	Combined 2 (HW)
`-O0`	99.77%	99.95%
`-O1`	97.72%	99.75%
`-O2`	97.55%	99.80%
`-O3`	79.15%	90.57%
`-Os`	98.50%	99.85%

Consequently, we can discover the *wvalue* with a success rate over 79.15%. If the optimization level is 0, the success rate is increased to more than 99.77%. In the case of analyzing the Hamming weight value of *wvalue*, the success rate is over 90.57%. Thus, the secret shared key K can be recovered by applying an exhaustive search of candidates.



(a) Optimization level 0 (b) Optimization level 1 (c) Optimization level 2



(d) Optimization level 3 (e) Optimization level s

Fig. 4: Distributions of the PoIs of the message encoding of FrodoKEM (set 0, set 1, set2, and set3 are $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, and $\mathbb{G}_4$, respectively)

6 Discussion

6.1 Application to Other 2nd Round Lattice-Based KEMs

In this section, we discuss applicability of our proposed single-trace attacks against LAC, NewHope, NTRU, and NTRU Prime. We show how the three types of attacks described in Section 3, Section 4, and Section 5 can be applied to each scheme. Target operations for the message encoding of the encapsulation phase are described in Appendix B. There exist determiner in the operations of LAC, NewHope, NTRU, and NTRU LPRime. The determiner in the Listing B.1 of LAC, Listing B.2 of NewHope, and Listing B.5 of NTRU LPRime are message, mask, and r[i]*q12, respectively. Hence, it is possible to cluster the power consumption traces into two groups using the single-trace attack methodology which is applied to Crystals-Kyber. The clustered group $\mathbb{G}_1$ and $\mathbb{G}_2$ represent the sensitive bit μ_i value. Accordingly, we can recover the message μ using the Algorithm 2, and can generate secret shared key K using the recovered μ and public values.

$$\mathsf{message} = \begin{cases} \texttt{0x00}\ ,\ if\ \mu_i = 0; \\ \texttt{0x7d}\ ,\ if\ \mu_i = 1. \end{cases} \quad \mathsf{mask} = \begin{cases} \texttt{0x00000000}\ ,\ if\ \mu_i = 0; \\ \texttt{0xffffffff}\ ,\ if\ \mu_i = 1. \end{cases}$$

$$\mathsf{r[i]*q12} = \begin{cases} \texttt{0x0000}\ ,\ if\ \mu_i = 0; \\ \texttt{0x0906}\ ,\ if\ \mu_i = 1. \end{cases}$$

In the case of NTRU, μ_i is the sensitive coefficient belonging to the set $\{0, 1, 2\}$, and there are determiner in the Listing B.3 of NTRU as follows.

$$(-(\mathsf{r} \rightarrow \mathsf{coeffs[i]} \gg 1)) = \begin{cases} \texttt{0x0000}\ ,\ if\ \mu_i = 0, 1; \\ \texttt{0xffff}\ ,\ if\ \mu_i = 2. \end{cases}$$

Thus, the power consumption traces can be classified into two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$. However, unlikely to the previous schemes, in the case of NTRU, $\mathbb{G}_1$ includes both $\mu_i = 0$ and $\mu_i = 1$. Therefore, $\mathbb{G}_1$ should be divided into two groups, which requires additional attack methodology. The results of step 6 in the Listing B.3,

$$\mathsf{r} \rightarrow \mathsf{coeffs[i]} \mid ((-(\mathsf{r} \rightarrow \mathsf{coeffs[i]} \gg 1))\&(\mathsf{NTRU\_Q} - 1)) = \begin{cases} \texttt{0x0000}\ ,\ if\ \mu_i = 0; \\ \texttt{0x0001}\ ,\ if\ \mu_i = 1; \\ \texttt{0x03ff}\ ,\ if\ \mu_i = 2; \end{cases}$$

can be divided into three groups. Thus, it is possible to cluster each μ_i into three groups by combining the attack methodologies which are applied to Crystals-Kyber and Saber.

In the Listing B.4 of Streamlined NTRU Prime, the secret message μ is f and $f[i]$ is belonging to the set $\{0, 1, -1\}$. Because f is an 8-bit integer array, $f[i]$ is as below.

$$f[i] = \begin{cases} \texttt{0x00}\ ,\ if\ \mu_i = 0; \\ \texttt{0x01}\ ,\ if\ \mu_i = 1; \\ \texttt{0xff}\ ,\ if\ \mu_i = -1. \end{cases}$$

Hence, similar to NTRU, combining the attack methodologies which are applied to Crystals-Kyber and Saber is needed. Figure 5 shows the proposed attack flowchart.

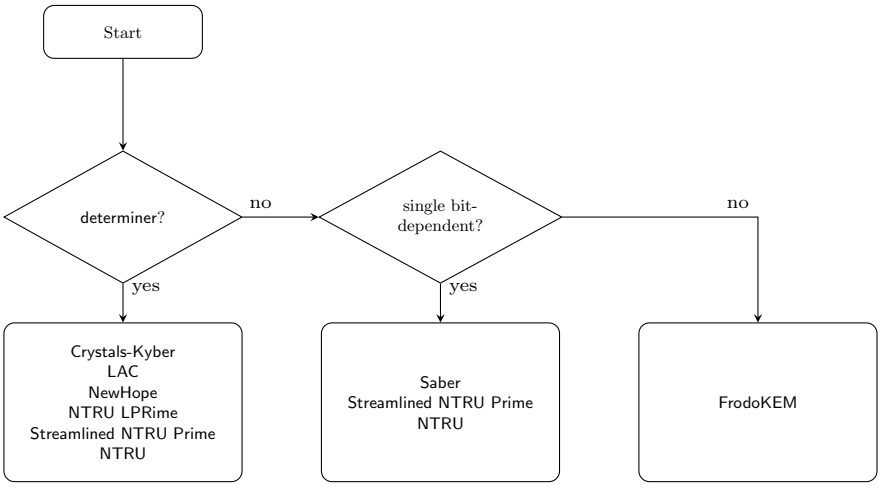


Fig. 5: Flowchart of our proposed attack

Remark. In this paper, Round5 and Three Bears are excluded from the attack target. In the case of Round5, there is an operation that is adding μ to a random variable before extracting μ_i from the string μ. Therefore, there is no operation to extract μ_i from the string μ and store it in the register. It does not satisfy our attacking assumptions. In the case of Three Bears, there is an operation that concatenates 8-bit characters to a 64-bit string before hashing. Thus, 8-bit value-dependent leakage occurs and it can be extracted by ML-based PA which is a more deep network than the network shown in Table 4. It is not covered in this paper and we leave it as further work.

6.2 Countermeasures

We here recommend countermeasure that increases the attack complexity, *i.e.*, makes the attack more difficult. The masking scheme, which splits secret message μ into two random messages μ' and μ'' [37], is not a perfectly secure countermeasure. Since it is possible to recover μ' and μ'' by applying the proposed single-trace attack twice, thus, we can calculate $\mu = \mu' \oplus \mu''$. On the other hand, the shuffling scheme can properly counteract against the proposed attack. The Listing 6.1 proposed by Amiet *et al.* [2] is for NewHope, but it can also be applied to Crystals-Kyber. By modifying the Listing 6.1, we can construct the Listing 6.2 and Listing 6.3 for Saber and FrodoKEM, respectively. Similarly, the shuffling scheme can be applied to LAC, NTRU, Streamlined NTRU Prime, and NTRU LPRime.

```
1 b = (fyList[i] >> 3); s = (fyList[i] & 7);
2 mask = -((msg[b] >> s) & 0x01);
```

Listing 6.1: Message Encoding with Fisher-Yates Shuffle for NewHope [2]

```
b = (fyList[i] >> 3); s = (fyList[i] & 7);
// unpack message received
message[8 * b + s] = ((message_received[b] >> s) & 0x01);
// message encoding
message[b] = (message[b] << (SABER_EP - 1));
```

Listing 6.2: Message Encoding with Fisher-Yates Shuffle for Saber

```
b = (fyList[i] >> 4);
temp |= ((uint64_t)((uint8_t*)in)[b*PARAMS_EXTRACTED_BITS + j])<<(8*j);
```

Listing 6.3: Message Encoding with Fisher-Yates Shuffle for FrodoKEM

Even though the proposed single-trace attacks can still be applied, however, it is impossible to determine the location of the extracted bits because bits are encoded in random order. Additional attacks on the shuffling scheme are required. Thus, as mentioned in [2], if you combine the shuffling with masking, it can increase the attack complexity.

7 Conclusion

We proposed three types of single-trace attacks against Crystals-Kyber, Saber, and FrodoKEM, targeting the message encoding of the encapsulation phase. Experiment results show that it is possible to recover entire secret messages with 100% success rates for Crystals-Kyber and Saber regardless of the optimization level. The message recovery success rate of FrodoKEM is over 79%. When the optimization level is 0, the success rate increases to more than 99.77%. We also showed that our attack methodologies are applicable to others including LAC, NewHope, Streamlined NTRU Prime, NTRU LPRime, and NTRU. Lastly, we recommended countermeasure which is combining shuffling and masking schemes to increase the attack complexity.

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: 25th USENIX Security Symposium, USENIX Security 16. pp. 327–343. USENIX Association (2016)
2. Amiet, D., Curiger, A., Leuenberger, L., Zbinden, P.: Defeating newhope with a single trace. In: International Conference on Post-Quantum Cryptography. pp. 189–205. Springer (2020)
3. Anzai, Y.: Pattern Recognition & Machine Learning. Elsevier (1992), https://doi.org/10.1016/c2009-0-22409-3
4. Atici, A., Batina, L., Gierlichs, B., Verbauwhede, I.: Power analysis on ntru implementations for rfids: First results. RFIDSec 2008 pp. 128–139 (2008)

5. Aysu, A., Tobah, Y., Tiwari, M., Gerstlauer, A., Orshansky, M.: Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In: IEEE International Symposium on Hardware Oriented Security and Trust. pp. 81–88. IEEE Computer Society (2018)
6. Baan, H., Bhattacharya, S., Fluhrer, S.R., García-Morchón, Ó., Laarhoven, T., Rietman, R., Saarinen, M.O., Tolhuizen, L., Zhang, Z.: Round5: Compact and fast post-quantum public-key encryption. In: International Conference on Post-Quantum Cryptography. pp. 83–102. Springer (2019)
7. Bartkewitz, T., Lemke-Rust, K.: Efficient template attacks based on probabilistic multi-class support vector machines. In: International Conference on Smart Card Research and Advanced Applications. pp. 263–276. Springer (2012)
8. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime: Reducing attack surface at low cost. In: International Conference on Selected Areas in Cryptography. pp. 235–260. Springer (2017)
9. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1006–1018. ACM (2016)
10. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In: IEEE European Symposium on Security and Privacy. pp. 353–367. IEEE (2018)
11. Bos, J.W., Friedberger, S., Martinoli, M., Oswald, E., Stam, M.: Assessing the feasibility of single trace power analysis of frodo. In: International Conference on Selected Areas in Cryptography. pp. 216–234. Springer (2018)
12. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 13–28. Springer (2002)
13. Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on Post-Quantum Cryptography. `https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf` (2016)
14. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR. In: International Conference on Security and Cryptography for Networks. pp. 160–177. Springer (2018)
15. D'Anvers, J., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure KEM. In: International Conference on Cryptology in Africa. pp. 282–305. Springer (2018)
16. D'Anvers, J., Tiepelt, M., Vercauteren, F., Verbauwhede, I.: Timing attacks on error correcting codes in post-quantum schemes. In: Proceedings of ACM Workshop on Theory of Implementation Security Workshop. pp. 2–9. ACM (2019)
17. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. pp. 226–231. AAAI Press (1996)
18. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999)
19. Fukunaga, K., Hostetler, L.D.: The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans. Inf. Theory **21**(1), 32–40 (1975)

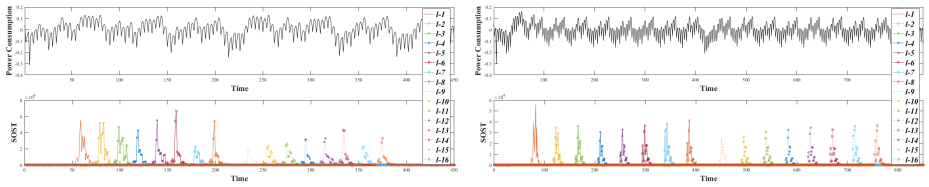
20. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 15–29. Springer (2006)
21. Heyszl, J., Ibing, A., Mangard, S., Santis, F.D., Sigl, G.: Clustering algorithms for non-profiled single-execution attacks on exponentiations. In: International Conference on Smart Card Research and Advanced Applications. pp. 79–93. Springer (2013)
22. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: International Algorithmic Number Theory Symposium. pp. 267–288. Springer (1998)
23. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Theory of Cryptography Conference. pp. 341–371. Springer (2017)
24. Hospodar, G., Gierlichs, B., Mulder, E.D., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptographic Engineering **1**(4), 293–302 (2011)
25. Huang, W., Chen, J., Yang, B.: Power analysis on NTRU prime. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 123–151 (2020)
26. Inc, N.T.: ChipWhisperer UFO. `https://wiki.newae.com/CW308T-STM32F`
27. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Annual International Cryptology Conference. pp. 104–113. Springer (1996)
28. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference. pp. 388–397. Springer (1999)
29. Lee, J., Kim, D., Lee, H., Lee, Y., Cheon, J.H.: Rlizard: Post-quantum key encapsulation mechanism for iot devices. IEEE Access **7**, 2080–2091 (2019)
30. Lee, M., Song, J.E., Choi, D., Han, D.: Countermeasures against power analysis attacks for the NTRU public key cryptosystem. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **93-A**(1), 153–163 (2010)
31. Lerman, L., Bontempi, G., Markowitch, O.: Side channel attack: an approach based on machine learning. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 29–41. Springer (2011)
32. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–23. Springer (2010)
33. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
34. Mariantoni, M.: Building a superconducting quantum computer (2014), `https://www.youtube.com/watch?v=wWHAs--HA1c`
35. Mosca, M.: Cybersecurity in an era with quantum computers: Will we be ready? IEEE Secur. Priv. **16**(5), 38–41 (2018)
36. NIST: Post-Quantum Cryptography, Round 2 Submissions, NIST Computer Security Resource Center. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions` (2019)
37. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 142–174 (2018)
38. Park, A., Han, D.: Chosen ciphertext simple power analysis on software 8-bit implementation of ring-lwe encryption. In: IEEE Asian Hardware-Oriented Security and Trust. pp. 1–6. IEEE Computer Society (2016)

39. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: International Conference on Cryptology and Information Security in Latin America. pp. 130–149. Springer (2019)
40. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 513–533. Springer (2017)
41. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing. pp. 84–93. ACM (2005)
42. Reparaz, O., de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Additively homomorphic ring-lwe masking. In: International Conference on Post-Quantum Cryptography. pp. 233–244. Springer (2016)
43. Reparaz, O., Roy, S.S., de Clercq, R., Vercauteren, F., Verbauwhede, I.: Masking ring-lwe. J. Cryptographic Engineering **6**(2), 139–153 (2016)
44. Rokach, L., Maimon, O.: Clustering methods. In: The Data Mining and Knowledge Discovery Handbook, pp. 321–352. Springer (2005)
45. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 520–551. Springer (2018)
46. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society (1994)
47. Silverman, J.H., Whyte, W.: Timing attacks on ntruencrypt via variation in the number of hash calls. In: The Cryptographers' Track at the RSA Conference. pp. 208–224. Springer (2007)

A Experiment Results

A.1 Crystals-Kyber

Figure 6(a) and Figure 6(b) show the SOST value between two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, of each μ_i, when the optimization level is 3 and s, respectively. They also show that the PoIs are regularly distributed.

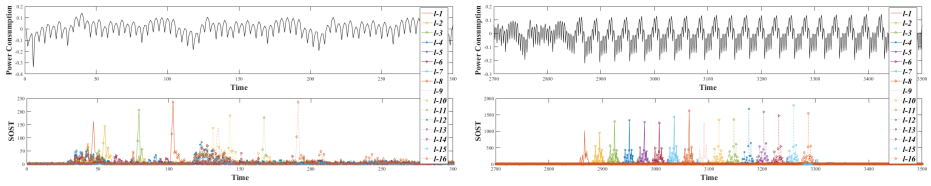


(a) Optimization level 3 (b) Optimization level s

Fig. 6: The power consumption trace of the message encoding of Crystals-Kyber (top) and the SOST value between two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, of each μ_i (bottom)

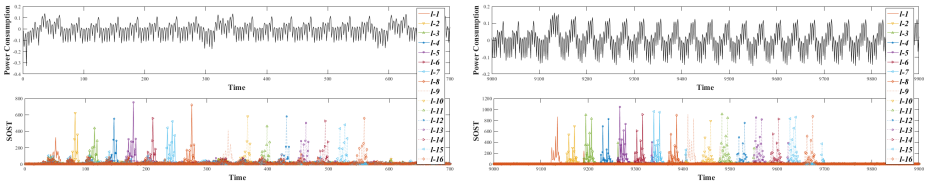
A.2 Saber

To identity the existence of the leakage based on the sensitive bit μ_i value, we calculate the SOST value of the power consumption traces. Figure 7 and Figure 8 show the SOST value between two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, of each μ_i, when the optimization level is 3 and s, respectively. As shown in Figure 7(a), there is no regularity at the first PoIs when the optimization level is 3. However, when the optimization level is not 3, points with high SOST values in the first PoIs are regularly distributed as shown in Figure 8(a).



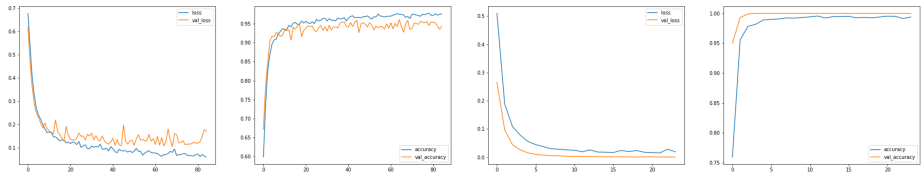
(a) First PoI (b) Second PoI

Fig. 7: The power consumption trace of the message encoding of Saber when optimization level is 3 (top) and the SOST value between two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, of each μ_i (bottom)



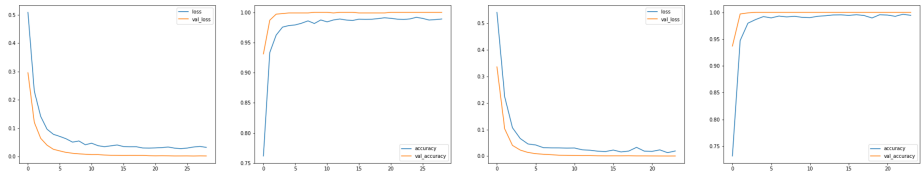
(a) First PoI (b) Second PoI

Fig. 8: The power consumption trace of the message encoding of Saber when optimization level is s (top) and the SOST value between two groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, of each μ_i (bottom)



(a) First PoIs (b) Second PoIs

Fig. 9: Loss (left) and accuracy (right) over the epochs for training and validation (Optimization level 3, Saber, 1-bit $\mu_{\ell-1}$ value)



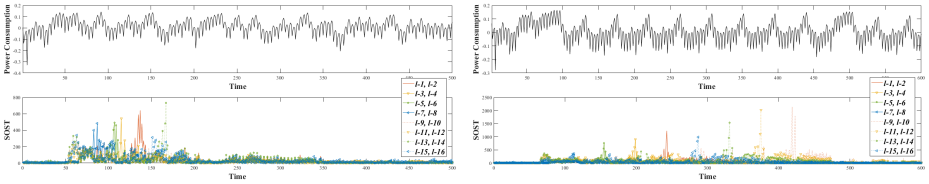
(a) First PoIs (b) Second PoIs

Fig. 10: Loss (left) and accuracy (right) over the epochs for training and validation (Optimization level s, Saber, 1-bit $\mu_{\ell-1}$ value)

Figure 9 and Figure 10 show the ML-based PA results. In the case of using the second PoIs for profiling 1-bit value, the validation accuracy is 1 in all the optimization level. Hence, 1-bit values of $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ are recovered with a 100% success rate as shown in Table 7. Whereas, in the case of using the first PoIs for profiling 1-bit value, the validation accuracy is over 0.95. Thus, 1-bit values of $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ are recovered with over a 95% success rate as shown in Table 7.

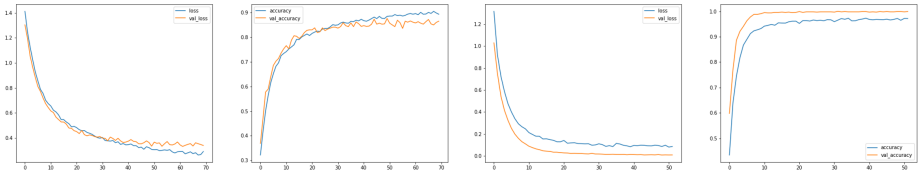
A.3 FrodoKEM

To identify the existence of the leakage based on the sensitive *wvalue*, we calculate the SOST value of the power consumption traces. Figure 11 shows the SOST value between four groups when the optimization level is 3 and s. As shown in Figure 11(a), there is no regularity when the optimization level is 3. However, when the optimization level is not 3, points with high SOST values are regularly distributed as shown in Figure 11(b).



(a) Optimization level 3 (b) Optimization level s

Fig. 11: The power consumption trace of the message encoding of FrodoKEM (top) and the SOST value between four groups, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, and $\mathbb{G}_4$, of each *wvalue* (bottom)



(a) Optimization level 3 (b) Optimization level s

Fig. 12: Loss (left) and accuracy (right) over the epochs for training and validation (Optimization level 3 and s, FrodoKEM, 2-bit value)

Figure 12 shows the ML-based PA results. For profiling 2-bit value, the validation accuracy is higher than 0.78 and 0.97, when the optimization level is 3 and s, respectively. Hence, 2-bit values of $\mu = (\mu_{\ell-1}, \cdots, \mu_1, \mu_0)_2$ are recovered with over 78% and 97% success rates, respectively, as shown in Table 9.

B Message Encoding

The secret message μ in Listing B.1, Listing B.2, Listing B.3, Listing B.4, and Listing B.5 are p_code, msg, r→coeffs, f, and r, respectively.

```
int pke_enc_seed(const unsigned char *pk, const unsigned char *m, unsigned
    long long mlen, unsigned char *c, unsigned long long *clen, unsigned
    char *seed)
{
  // (omit)
  // compute the length of c2
  c2_len = (mlen + ECC_LEN) * 8 * 2;
  // generate error vector e2
  gen_psi_std(e2, c2_len, seeds + 2 * SEED_LEN);

  int vec_bound = c2_len / 2;
  int8_t message;
  // compute code * q/2 + e2
  for(i = 0; i < vec_bound; i++)
  {
    // RATIO = q/2, add code * q/2 to e2
    message = RATIO * ((p_code[i / 8] >> (i % 8)) & 1);
    e2[i] = e2[i] + message;
    // D2 encode, repeat at i + vec_bound
    e2[i + vec_bound] = e2[i + vec_bound] + message;
  }
  // (omit)
}
```

Listing B.1: Message Encoding encode(μ) of LAC

```
void poly_frommsg(poly *r, const unsigned char *msg)
{
  unsigned int i, j, mask;
  for(i = 0; i < 32; i++) // XXX: MACRO for 32
  {
    for(j = 0; j < 8; j++)
    {
      mask = -((msg[i] >> j) & 1);
      r->coeffs[8*i+j+  0] = mask & (NEWHOPE_Q/2);
      r->coeffs[8*i+j+256] = mask & (NEWHOPE_Q/2);
#if (NEWHOPE_N == 1024)
      r->coeffs[8*i+j+512] = mask & (NEWHOPE_Q/2);
      r->coeffs[8*i+j+768] = mask & (NEWHOPE_Q/2);
#endif
    }
  }
}
```

Listing B.2: Message Encoding encode(μ) of NewHope

```
// Map {0, 1, 2} -> {0, 1, q-1} in place
void poly_Z3_to_Zq(poly *r)
{
  int i;
  for(i = 0; i < NTRU_N; i++)
    r->coeffs[i] = r->coeffs[i] | (-(r->coeffs[i] >> 1) & (NTRU_Q - 1));
```

```
7 }
```

Listing B.3: Message Encoding encode(μ) of NTRU

```
1  static void Small_encode(unsigned char *s, const small *f)
2  {
3    small x;
4    int i;
5
6    for(i = 0; i < p/4; ++i)
7    {
8      x = *f++ + 1;
9      x += (*f++ + 1) << 2;
10     x += (*f++ + 1) << 4;
11     x += (*f++ + 1) << 6;
12     *s++ = x;
13   }
14   x = *f++ + 1;
15   *s++ = x;
16 }
```

Listing B.4: Message Encoding encode(μ) of Streamlined NTRU Prime

```
1  #define q12 ((q-1)/2)
2  static void Encrypt(Fq *B, int8 *T, const int *r, const Fq *G, const Fq *A
       , const small *b)
3  {
4    Fq bG[p];
5    Fq bA[p];
6    int i;
7
8    Rq_mult_small(bG, G, b);
9    Round(B, bG);
10   Rq_mult_small(bA, A, b);
11   for(i = 0; i < I; ++i)
12     T[i] = Top(Fq_freeze(bA[i] + r[i] * q12));
13 }
```

Listing B.5: Message Encoding encode(μ) of NTRU LPRime