Practical Side-Channel Attack on Masked Message Encoding in Latticed-Based KEM

Jian Wang*†, Weiqiong Cao*, Hua Chen*, Haoyuan Li*†
*Institute of Software, Chinese Academy of Sciences, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
Email: {wangjian2019, caoweiqiong, chenhua, haoyuan2019}@iscas.ac.cn

***Abstract*—To defend against the rising threat of quantum computers, NIST initiated their Post-Quantum Cryptography(PQC) standardization process in 2016. During the PQC process, the security against side-channel attacks has received much attention. Lattice-based schemes are considered to be the most promising group to be standardized. Message encoding in lattice-based schemes has been proven to be vulnerable to side-channel attacks, and a first-order masked message encoder has been presented. However, there is still a lack of security evaluation for the first-order masked message encoder under different implementations. In this paper, we analyzed the security of the first-order masked message encoder of Kyber. We found although masked Kyber certainly is able to defend against the previous side-channel attacks, there still exist some exploitable leakages. With the help of the leakages, we propose a deep learning-based key recovery attack on message encoding of masked Kyber. We recover the original message from masked message encoding and then enable a chosen-ciphertext attack to recover the secret key. In our experiments, the whole secret key of masked Kyber768 was recovered with only 9 traces and the success rate of attack was close to 100%.**

***Index Terms*—Side-channel attack, Lattice-based cryptography, Kyber, Masking, Deep learning**

I. Introduction

With quantum computers, Shor's algorithm [1] can break the cryptosystems based on the classical computationally infeasible problems(e.g. RSA, ECC), which are playing important roles in today's communication, financial, IoT, etc. In recent years, more and more progress on quantum computers has been made by Google [2], USTC [3], IBM [4], etc. Updating cryptosystems with new cryptography primitives against quantum computers becomes urgent. In 2016, the National Institute of Standards and Technology(NIST) initiated their Post-Quantum Cryptography(PQC) standardization process, and received 69 algorithms from all over the world. Now, the PQC process has reached the third round, 7 finalists will continue to be reviewed for consideration for standardization at the conclusion of the third round, and NIST expects security in relation to side-channel attacks can have a larger focus [5] in this round.

In the third round of the PQC process, 4 of the 7 finalists are constructed from the lattice-based computationally infeasible problems, in which Kyber [6] is viewed as one of the most promising Key Encapsulation Mechanism(KEM) [5]. Accordingly, many studies on side-channel security of lattice-based schemes have been proposed over the years. In [7], Primas et al. proposed the first side-channel attack on the number theory transform(NTT) of RLWE-based schemes. They used a combination of template attack and belief propagation to achieve a single-trace key recovery attack. Subsequently, this attack got improved by Pessl [8] and Hamburg [9] et al. The correlation power analysis on polynomial multiplication of secret key has been discussed in [10]. Besides the classical side-channel attacks on key-dependent operation, the side-channel assisted chosen-ciphertext attacks have attracted much attention. In these attacks, the side-channel information is used as oracles to reveal some information about secret key, and these oracles can be categorized as plaintext check oracle [11] [12] [13], message recovery oracle [14] [15] and decryption failure oracle [16] [17]. To defend against side-channel attacks, masking is an efficient method that is firstly proposed in [18]. In [19], Oder et al. proposed the first first-order masking for the CCA-secured lattice-based schemes. Subsequently, improvements on masked binomial sampler [20], and comparison [21] [22] [23] were proposed. Furthermore, first-order and higher-order masked implementations of Kyber [24] [25] and Saber [26] [27] have been achieved.

A. Related work

1) Message Recovery Attack on Message Encoding: Due to the session key between the two communicating parties derived from the input message, the key can be deduced if the message is recovered. In [28], Amiet et al. proposed the first side-channel message recovery attack on latticed-based schemes. in which the message encoding of NewHope(a second-round algorithm in NIST's PQC process) was targeted. They employed simple power analysis(SPA) to recover the message from NewHope with an compiler optimization flag "-O0" and template attack to recover the message from "-O3" compilation result respectively. Next, Sim et. al [29] implemented the message recovery attack on multiple lattice-based schemes in different compilation optimization by clustering or neural network algorithms.

2) Message Recovery based Chosen-Ciphertext Attack : In 2020, Ravi et al. [12] used electromagnetic leakage from error correct codes or FO-transform as a plaintext check oracle, proposed a generic key recovery attack on lattice-based KEMs. However, the plaintext check oracle can only distinguish one bit of the message, and thereby only recover one secret coefficient every time. Hence, recovering the whole

secret key needs thousands of traces. After that, Xu et al. [14] used SPA on the message encoding to recover the whole message with only a single trace. Based on the single-trace message recovery, they can recover the whole secret key of Kyber512 [1] with 8 traces, which is 7680 in [12]. Furthermore, Ravi et al. [15] improved the ciphertext-chosen method in [14], reduced the number of needed traces to 6.

3) Masked Message Encoding: Since message encoding is vulnerable to side-channel attacks, implementing secure message encoders is important in CCA-secure lattice-based KEMs. In [19], Oder et al. firstly proposed a first-order masked message encoder for NewHope KEM. In [30], Heinz et al. introduced the masked encoder of [19] into their first-order masked Kyber implementation, and it was also applied to Saber as introduced in [26]. In addition, converting Boolean masking of the message to Arithmetic masking bit-wisely [24] is another implementation of a secure masked message encoder, but it costs too much for a first-order masking scheme.

4) High-order Attack on Masked Lattice-Based Schemes: In [31], based on the approach introduced in [32], Ngo et al. proposed a key recovery attack on the message decoding in masked Saber [26]. They combined the side-channel leakage when decoding two shares of one bit as input to training a neural network, which can recover the original message bit with several traces. This attack not only doesn't need any profiling device but can improve accuracy by using more than one trace without considering the randomness caused by re-masking. In [33], Ngo et al. achieved a key recovery attack on masked and shuffled Saber by a similar method.

To sum up, the message encoding in lattice-based schemes is vulnerable to side-channel attacks. Accordingly, the masked encoder proposed in [19] is an effective countermeasure to defend against side-channel attacks and has been implemented in multiple masked implementations of lattice-based schemes. However, the security of masked encoder still lacks enough evaluation, a practical attack on masked decoder has been proposed in [31]. Whether there are high-order attacks that can threaten the masked encoder is still an open question.

B. Contributions

In this paper, we target the masked Kyber implementation in [30] and propose a practical side-channel attack on its masked message encoding. Our contributions can be summarized as follows:

- The proposed attack is the first practical attack on the masked message encoding. Although the previous attacks [28] [29] [14] on the message encoding without masking can be extended to analyze the masked encoding, it will be infeasible once the implementation of masked encoding is under the weak leakage case of low Signal-Noise-Ratio(SNR), unevenness or masking (which is common in actual implementations). On the contrary, the restrictions above do not affect our attack performed successfully.
- The leakage model of our attack is more general. The single-bit leakage model used in the previous attacks [28] [29] [14] may be broke by compilation optimization. Our attack is based on a byte leakage model. Even though there is no exploitable leakage to distinguish a single bit(i.e., the power trace is not regular to distinguish the bit iteration), our model can still employ all the leakage of bits in one byte to recover the message without precisely locating the points in the trace.
- Our attack is validated on a Cortex-M4-based development board, and we were able to achieve close to 100% accuracy in both message recovery and key recovery for 1000 experiments.

The code and data used in this paper will be publicly available.

C. Organization

The remainder of this paper is organized as follows: We provide notations and basic principles of related work in Section II. In Section III, we introduce our 2-stage key recovery attack and show how we improve our attack with imperfect message recovery. We use realistic experiments to validate our methodologies in Section IV. Finally, we conclude this paper and future work in Section V.

II. PRILIMINARIES

A. Notations

Let q be prime and $\mathbb{Z}_q$ be the ring of integers modulo q. We define the ring of polynomials $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ for some interger n and denote lower case letters like v as polynomials over $\mathcal{R}_q$. Similarly, $\mathcal{R}_q^{k\times 1}$ represents vectors with k elements and $\mathcal{R}_q^{k\times l}$ matrices of dimension $k \times l$ over $\mathcal{R}_q$. The transpose of a vector $\mathbf{u}$ or a matrix $\mathbf{A}$ are denoted by $\mathbf{u}^T$ and $\mathbf{A}^T$, respectively. v_i and $\mathbf{u}[i]$ are denoted as the i-th coefficient in v and the i-th polynomial in vector $\mathbf{u}$, $\mathbf{A}[i][j]$ is denoted as the polynomials located in i-th row, j-th column of matrix $\mathbf{A}$. When we use NTT($\mathbf{a}$), we apply number theory transform(NTT) to each polynomial in $\mathbf{a}$. $\hat{v}$ represents a polynomial in NTT-domain, $\hat{\mathbf{u}}$ and $\hat{\mathbf{A}}$ represent a vector and a matrix whose polynomials are in NTT-domin. Multiplication in any ring is denoted by $\cdot$ operator whereas point-wise multiplication is denoted by $\circ$ operator.

For $x \in \mathbb{R}$, we write $\lfloor x \rceil$ to mean the closest interger to x. We use $\mathcal{U}$ to denote the uniform distribution on $\mathcal{R}_q$, whereas χ_η denotes a center binomial distribution with support $[-\eta...\eta]$. Byte arrays of length z are denoted as $\mathcal{B}^z$, and by $\{0,1\}^z$, we denote the set of z bits. In our paper, we let $m \in \mathcal{B}^{32}$, and we denote $m[i]$ as the i-th byte in m and m_i as the i-th bit in m. In masked cases, 2 shares of m are denoted as m' and m'', this is also applied to $m[i]$ and m_i.

B. Kyber

Kyber is one of the 7 finalists in the third round of the NIST's PQC standardization process [34]. In this paper, we use Kyber to showcase our attack. The security of Kyber bases

[1]The Kyber512 they used here is the version from the second round of PQC process with η_1 equals 2. In the third round, the η_1 of Kyber512 is increased to 3.

on the hardness of solving the learning-with-errors problem in module lattices(MLWE [35]). Kyber consists of a chosen-plaintext attack secured public-key encryption(CPAPKE) and a chosen-ciphertext attack secured key encapsulation mechanism(CCAKEM). CPAPKE consists of three parts: key generation(Algorithm 1), encryption(Algorithm 2) and decryption(Algorithm 3). FO-transformation [36] [37] is applied to CPAPKE to get CCAKEM, the core part of CCAKEM is key decapsulation(Algorithm 4) based on a decryption and re-encryption. The three parameter sets Kyber512, Kyber768 and Kyber1024 are claimed to the security of AES-128, AES-192 and AES-256 respectively. In this paper, we focus on Kyber768, but our approaches can also be applied to the other two sets. Parameters in Kyber768 are shown in Table I. $k = 3$ means secret key $\mathbf{s}$ has 3 polynomials, $\eta_1 = 2$ means the coefficients in $\mathbf{s}$ belong to $\{-2, -1, 0, 1, 2\}$.

Algorithm 1 CPAPKE.Gen

Output: Secret key sk
Output: Public key pk
$d \leftarrow \mathcal{B}^{32}$
$(seed_A, r) \leftarrow G(d)$
$\hat{\mathbf{A}} \leftarrow \mathcal{U}(R_q^{k\times k}, seed_A)$
$(\mathbf{s}, \mathbf{e}) \leftarrow \chi_{\eta_1}(R_q^{k\times 1}; r) \times \chi_{\eta_1}(R_q^{k\times 1}; r)$
$\hat{\mathbf{s}} \leftarrow$ NTT($\mathbf{s}$), $\hat{\mathbf{e}} \leftarrow$ NTT($\mathbf{e}$)
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
$sk \leftarrow \hat{\mathbf{s}}$, $pk \leftarrow (\hat{\mathbf{t}}||seed_A)$
return (sk, pk),

Algorithm 2 CPAPKE.Enc

Input: Public key pk
Input: Message $m \in \mathcal{B}^{32}$
Input: Random coins $r \in \mathcal{B}^{32}$
Output: Ciphertext s
$\hat{\mathbf{A}}^T \leftarrow \mathcal{U}(\mathcal{R}_q^{k\times k}, seed_A)$
$\mathbf{r} \leftarrow \chi_{\eta_1}(\mathcal{R}_q^{k\times 1}; r)$
$\mathbf{e_1} \leftarrow \chi_{\eta_2}(\mathcal{R}_q^{k\times 1}; r)$
$e_2 \leftarrow \chi_{\eta_2}(\mathcal{R}_q; r)$
$\hat{\mathbf{r}} \leftarrow$NTT($\mathbf{r}$)
$\mathbf{u} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v \leftarrow \text{NTT}^{-1}(\hat{\mathbf{b}}^T \circ \hat{\mathbf{r}}) + e_2 + \text{encode}(m)$
$c_1 \leftarrow$ Compress($\mathbf{u}, d_u$)
$c_2 \leftarrow$ Compress(v, d_v)
return $(c_1||c_2)$

Algorithm 3 CPAPKE.Dec

Input: Ciphertext $c/*c = (c_1||c_2)*/$
Input: Secret Key $\hat{\mathbf{s}}$
Output: Message m
$\mathbf{u} \leftarrow$ Decompress(c_1, d_u)
$v \leftarrow$ Decompress(c_2, d_v)
$m \leftarrow$ decode($v -$ NTT^{-1}(NTT($\mathbf{u}$)$\circ\hat{\mathbf{s}}$))
return m

Algorithm 4 CCAKEM.Dec

Input: Ciphertext $c/*c = (c_1||c_2)*/$
Input: Secret Key $\hat{\mathbf{s}}$
Output: Session key K
$m' \leftarrow$ CPAPKE.Dec($\hat{\mathbf{s}}, c$)
$\bar{K}', r' \leftarrow$ G($m'||$H(pk))
$c' \leftarrow$ CPAPKE.Enc(pk, m', r')
if $c = c'$ **then**
  return $K \leftarrow$ KDF($\bar{K}||'$H(c))
else
  return $K \leftarrow$KDF($z||$H(c))
end if

TABLE I
PARAMETER SETS OF KYBER768

parameters	n	q	k	(η_1, η_2)	(d_u, d_v)
values	256	3329	3	$(2, 2)$	$(10, 4)$

C. Side-Channel Attack on Message Encoding

In Kyber's algorithm specification [34], message encoding is considered as a special form of Decompression, it is a mapping between a message and a polynomial as shown in (1)

$$\mathcal{M} : \{0, 1\}^n \rightarrow \mathcal{R}_q \tag{1}$$

Concretely, message encoding maps each bit of the message to the corresponding coefficient in the polynomial as (2).

$$r[i] = \begin{cases} 0 & \textbf{if } m_i = 0 \\ \lfloor \frac{q}{2} \rceil & \textbf{if } m_i = 1 \end{cases} \tag{2}$$

In Kyber's reference implementation [34], message encoder is implemented as Listing 1.

```
void poly_frommsg(poly *r, const uint8_t msg[32]) {
    size_t i, j;
    int16_t mask;

    for (i = 0; i < 32; i++) {
        for (j = 0; j < 8; j++) {
            // mask = 0xffff/0x0000
            mask = -(int16_t)((msg[i] >> j) & 1);
            r->coeffs[8 * i + j] = mask & ((KYBER_Q
    + 1) / 2);
        }
    }
}
```

Listing 1. Kyber's reference message encoding

When a message bit is 0(or 1), the variable ***mask***(not related to the masking introduced in Section II-D) is **0x0000**(**0xffff**), respectively. The difference in hamming weight between the two possible values of ***mask*** is 16, which is a strong power leakage in microcontroller. In [29], Sim et al. defined the variables with strong hamming weight difference like ***mask*** here as **determiner**, we follow this definition in this paper. Using ***mask*** as a determiner, the attacker is able to distinguish whether a message bit in Kyber is 0 or 1 by observing power traces or leveraging some simple extracted features in traces [28] [29].

D. Masked Encoder

Masking [18] is a widely used countermeasure against side-channel attacks. It splits a secret intermediate value into multiple parts called shares and perform all the operations on each of the shares individually. A first-order masking splits any secret variable x into 2 shares x_1 and x_2, satisfying $x = x_1 + x_2$ in Arithmetic masking or $x = x_1 \oplus x_2$ in Boolean masking.

First masked message encoder for lattice-based scheme is proposed in [19] and applied to implementation of Kyber [30] and Saber [26]. Masked message encoder in Kyber is shown in Algorithm 5. The main idea is to encode two shares individually, then subtract round error correctly and securely. Our attack targets the separate encoding part, corresponding to **step 3, 4** in Algorithm 5. The subtraction part is not the focus of this paper, so we won't go into it here.

Algorithm 5 Masked Kyber.Encode

Input: m', m'' /* $m = m' \oplus m''$ */
Output: A_1, A_2 /* $A = A_1 + A_2, A = m * q/2$ */
1: $a', a'' \leftarrow reshare(m')$
2: $b', b'' \leftarrow reshare(m'')$
3: $A_1 = encode(m')$
4: $A_2 = encode(m'')$
5: $A_1 = A_1 - (a'b') - (a'b'') - (a''b') - (a''b'')$
6: **return** (A_1, A_2)

III. Methodologies

A. Analyzing mkm4

In [30], Heinz et al. presented the first open-source Cortex-M4 implementation of masked Kyber: **mkm4**. In mkm4, the masked encoder presented in [19] (see Section II-D) is implemented and the core code of masked encoder is listed in Listing 2, where the same operations as Listing 1 are applied to m' and m'' in sequence.

```
for (i = 0; i < KYBER_SYMBYTES; i++) {
    for (j = 0; j < 8; j++) {
      mask = -((msg->share[0].u8[i] >> j) & 1);
      r->polys[0].coeffs[8 * i + j] += (mask & ((
    KYBER_Q + 1) / 2));
    }
  }

for (i = 0; i < KYBER_SYMBYTES; i++) {
  for (j = 0; j < 8; j++) {
    mask = -((msg->share[1].u8[i] >> j) & 1);
    r->polys[1].coeffs[8 * i + j] += (mask & ((
    KYBER_Q + 1) / 2));
  }
}
/*code about subtracting m_1*m_2 is omitted*/
```

Listing 2. Masked encoder in mkm4

To evaluate its security, we intended to perform a SPA as in [28] at first. We selected a message containing 32 **0xff** bytes and encapsulated it as ciphertext. Then, we decapsulated the ciphertext in the device under attack and captured the power consumption trace of decapsulation. We expected there are

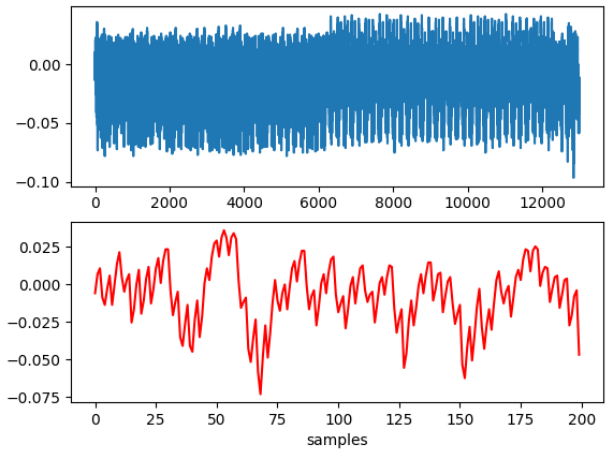

Fig. 1. Power consumption of masking message encoding(top) and amplification of the first peak(bottom)

two similar sets of 256 repeated peaks in the single trace, which indicates that the leakages in Section II-C are still exploitable. However, as shown in Fig. 1, we only found 32 peaks for each share(the upper one in Fig. 1) and didn't find any available leakage for SPA in each peak(the lower one in Fig. 1). To understand this phenomenon, we obtained its assembly code from the optimized compilation intermediate result. For simplicity, we omit unrelated code lines here. The related instructions for encoding one bit are shown in Listing 3. Since the code segment for encoding one byte are too long, we will not show it here.

```
ldrh    r7,     [r2, #4]
mov     r4,     #1665
// mask = -((msg->share[0].u8[i] >>2) & 1)
sbfx lr, r3, #2, #1
// coeffs += (mask & ((KYBER_Q + 1) / 2))
and lr, r4, lr
add     lr,     r7
strh    lr,     [r2, #4]
```

Listing 3. simplified assembly code of encoding a message bit in mkm4

In **step 7** of Algorithm 2, the encoded message is added to v , and $v = \mathbf{b}^T \cdot \mathbf{r} + e_2$. The simplified encoding process can be described as follows: the coefficients of v are loaded into registers of the microcontroller at first, and then the message will be encoded. Subsequently, the encoded results are added to the coefficients of v and then the coefficients of v are stored back in memory at the end. In previous works, there are usually the following assumptions: 1) message is encoded bit-by-bit; 2) during encoding one bit, loading ***mask*** from memory and storing ***mask*** to memory offer a strong and regular leakage about the value of the bit. However, there is no exploitable bit leakage in our analysis. By analyzing the assembly code of encoding **one byte**, we draw some reasons for the phenomenon as follows:

- **Low Signal-Noise-Ratio(SNR)**: The difference of power consumption between operations in registers and memory

can reach 1000 times [38], which means the only two message-dependent operations **sbfx** and **and** don't cause a strong leakage. When the encoded results are added to v, v is equivalent to a random masking, and thereby storing the coefficients of v can't leak useful information about message.

- **Unevenness**: Limited by register quantity, the microcontroller will process different bits and coefficients irregularly, and power traces are not even during encoding one byte. It is difficult to locate the points of encoding a specific bit.
- **Masking**: In the masked case, the two shares m' and m'' are changed every decapsulation, so we can't use specific traces to search points-of-interest(POI), build template as in [28] [29] or average multiple traces to improve SNR like [14].

To sum up, the masked encoder with compilation optimization breaks the correlation between message bits and power consumption efficiently. Natrually, the corresponding implementation of masked Kyber can resistant the previous side-channel attack on message encoding. However, we still found some exploitable leakages during the masked encoding and proposed a more general 2-stage side-channel attack to achieve key recovery. The following two subsections will introduce our attack in detail.

B. Message Recovery on Masked Message Encoding

As mentioned above, if recovering the original message in the decapsulation of masked Kyber, it is needed to recover m' and m'' with a single trace at first. However, in our case, it is difficult to obtain the points of interest or labels of templates due to the low SNR, unevenness and masking for the implementation of masked Kyber. In [32], Maghrebi et al. have proven that joining the trace points of each share to train a neural network can be used to recover the original secret key of AES. Using this method, we can also train a model to directly recover the original message without recovering m' and m''. Moreover, the deep learning-based side-channel attack performs well in low SNR environments and is therefore suitable for our scenario. In addition, we recover the message byte-wisely in order to avoid the unevenness.

Concretely, we introduce the deep learning technology into the side-channel attack on message encoding to achieve a feasible sing-trace attack. Benefiting from the powerful ability of deep learning, in profiling phase, we can profile the relation between m and power leakage from encoding m' and m''. We use unmasked message byte $m[i]$ as label and join power traces from encoding $m'[i]$ and $m''[i]$ as input of model. The method described here is similar to that in [31], but they target the incremental storage leakage of Saber's message decoding and are based on a bit model. Hence, they need to build 8 models for 8 bits in one byte, while we only need to train one model to recover all message bytes.

Without loss of generality, we use Multi-Layer Perception(MLP) as our profiling algorithm, which is one of the simplest deep learning models. The MLP structure we use here is shown in TableII. We use `ReLu` as activate the function of Dense layers and use `softmax` as activate function of the output layer, respectively. We select cross entropy as the loss function and use `adam` as our optimizer. In addition, to avoid overfitting, we add `Dropout` with a rate of 0.5 to drop 50% nodes of Dense layers during training.

TABLE II
MLP STRUCTURE IN MESSAGE RECOVERY ATTACK, $l = 190$ IN OUR CASE

Layer Type	(Input, Output)shape	Parameters
Batch Normalization 1	$(2l,\ 2l)$	$2l$*4
Dense1	$(2l,\ 512)$	195072
Batch Normalization 2	$(512, 512)$	2048
Dense2	$(512, 256)$	131328
Batch Normalization 3	$(256, 256)$	1024
Output	$(256, 256)$	65792

Since we can generate the ciphertext from any chosen message and decapsulate it by the device under attack, the device under attack can be viewed to be our profiling device without any extra profiling device.

The first step of the profiling phase is to collect some traces and the corresponding labels. For each trace capturing, we select random message m and encapsulate it into ciphertext ct. Then, we send ct to the device under attack and capture the traces during decapsulation. We store traces and denote $m[i]$ as labels, and with every decapsulation, we can get one trace and 32 byte-labels.

After capturing traces, we need to locate leakage points corresponding to every $m'[i]$ and $m''[i]$. The message encoding is located near the end of the decapsulation and searching from almost the end of the captured trace, we can find two similar 32-segment repeating patterns. All we need is to measure their length and divide them into 32 sub-traces corresponding to 32 bytes of m' and 32 bytes of m'' respectively. Joining every two sub-traces of $m'[i]$ and $m''[i]$, we set the joined sub-trace as input for our MLP model, the original message byte will be the label during training. We call this phrase preprocessing, and after we finish the trace preprocessing, we can train the neural network model $\mathcal{NN}$.

Once we have finished model training, we get a trained model $\mathcal{NN}^*$. The trained model has the ability to recover message byte value from power traces captured from the device under attack, which runs a masked Kyber decapsulation implementation. The whole message recovery process is shown in Algorithm 6

As the SNR is low, it is difficult to acquire a perfect model, there may be some false positive cases during message recovery. In [31], they decapsulate one ciphertext multiple times and get multiple recovered messages. For every message bit, they use majority voting of multiple recovered messages to improve accuracy. In our cases, 256 possible results exist, so majority voting doesn't fit here. Since the output of our MLP model is activated by the softmax function, we obtain a weight vector from the output of the neural network, and we select the class with the most weights as our recovery result.

Algorithm 6 Message Recovery Attack

/*Profiling Stage*/
for $i = 0$ to N **do**
  $msg \leftarrow$ RandomBytes(32)
  $ct \leftarrow$ Encaps(msg, pk)
  $Trace[i] \leftarrow$ Capture(Decaps(ct, sk))
  $Label[i] \leftarrow msg$
end for
$(X, y) \leftarrow$ PreProcess($Trace, Label$)
$\mathcal{NN}^* \leftarrow$ Train($\mathcal{NN}, X, y$)

/* Attack Stage*/
$malicious\_ct \leftarrow$ construct()
$Trace_{attack} \leftarrow$ Decaps($malicious_ct$, sk)
$X^*[32] \leftarrow$ PreProcess($Trace_{attack}$)
for $i = 0$ to 32 **do**
  $msg[i] \leftarrow$ argmax($\mathcal{NN}^*(X^*[i])$)
end for

We observed that in case of incorrect recoveries, although the correct results do not have the most weight, they are ranked high. With a high accuracy of our model, we can average several result vectors of d sub-traces to increase the ranking of correct candidates and decrease the ranking of incorrect candidates. In this way, we can achieve recovery accuracy very close to 100%, we'll show its efficiency in Section IV.

C. Key Recovery

We can simplify CPAPKE.Dec(Algorithm 2) as $m = \text{decode}(v - \mathbf{su})$, where $(\mathbf{u}, v)$ is ciphertext and $\mathbf{s}$ is secret key. That means the message can be seen as a linear combination of ciphertext and secret key. In our attacker model, we can select a message to construct ciphertext as we want, and if we construct some ciphertext with a special structure, we can recover some information about the secret key. In [14], Xu et al. treated it as a classification problem. They choose a fixed v and search several intervals of $\mathbf{u}$ to perform a One-vs-Rest(OvR) classification, only need 4 traces, they can recover one polynomial of $\mathbf{s}$. The theoretical number of traces required to achieve the N-categories classification problem is $\lceil \log_2 N \rceil$, which is 3 here. In [15], Ravi et al. reduced this number to 3 by random search of $(\mathbf{u}, v)$, our key recovery attack refers to [15].

Following Ravi et al. [15], we used a random search and selected 3 ciphertext pairs shown in Table III. Specifically, when we target i-th polynomial in $\mathbf{s}$, we construct ciphertext as $v = \sum_{j=1}^{256} k_v x^j$, and $u[i] = k_u$ where $u[i]$ is the i-th polynomials in $\mathbf{u}$ and other polynomials in $\mathbf{u}$ are 0. In this case, the message bit $m_j = \text{decode}(k_v - k_u * \mathbf{s}_j[i])$, we can recovery $\mathbf{s}[i][j]$ by the recovered 3 m_js.

In [14], Xu et al. used 960 traces to achieve a 98% accuracy rate, only 502 of 512 coefficients are recovered correctly. They need to exhaustive search $\binom{10}{512} * 5^{10}/2$ times to find the correct secret key. Our message recovery and average method in Section III-B reduce this search space to be negligible, we will show the details in Section IV.

TABLE III
MALICIOUS CIPHERTEXT PAIRS

Secret Coeffs	(k_u, k_v)		
	$(107, 2705)$	$(624, 1873)$	$(1252, 0)$
-2	O	O	O
-1	O	O	X
0	O	X	O
1	O	X	X
2	X	O	O

[a] O refers to the case of $m_j = 0$, X refers to $m_j = 1$

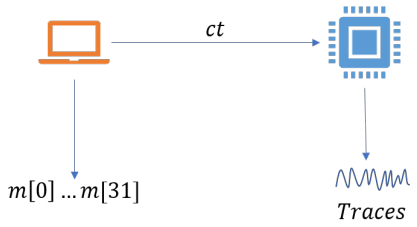

Fig. 2. Capture configuration

IV. EXPERIMENTS

A. Setup

For our experiments, we targeted an STM32F303 microcontroller featuring an ARM Cortex M4 core, which is the standard platform for evaluating embedded software implementations of the schemes running in NIST's PQC process [39]. The specific target board comes with the ChipWhisperer Lite [40], which is the equipment we used for measurement. The measurements done with the ChipWhisperer Lite will be voltage measurements over a shunt-resistor placed between the target processor and its supply.

The target device was programmed with the **mkm4** [30], which is the first open-source masked Kyber implementation based on the famous post-quantum cryptography test framework **pqm4** [41]. We used Kyber768 parameter set where $\eta_1 = 2$ and $k = 3$. We compiled the implementation using **arm-none-eabi-gcc** [42] with optimization flag "-O3".

B. Model Training

1) Capture Traces: We set the STM32F3 microcontroller as a server and our laptop as a client. Every time we selected a random message m and encapsulated m with the public key into ciphertext ct on the client, then we sent ct to the server through a serial port. During decapsulation, we captured power traces and saved $m[0]...m[31]$ as labels. Our capture configuration is shown in Fig. 2.

2) Traces Pre-Process: In this stage, we first divided each trace into two sub-traces, corresponding to the encoding of m' and m''. We named the first point of the first sub-trace the start-point. In each sub-trace, which contains 32 repeated

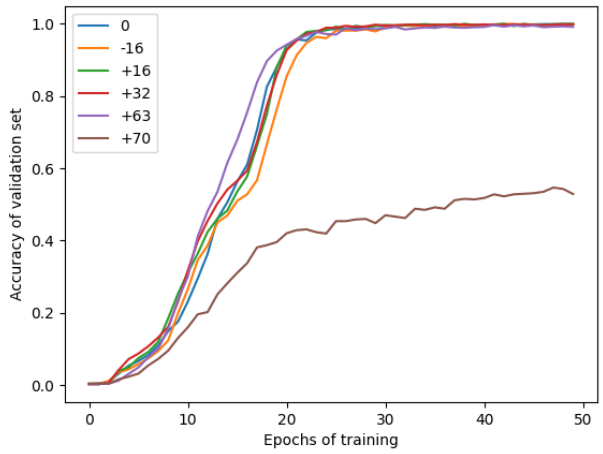

Fig. 3. Accuracy of validation set during train for different start-points, the labels in upper left mean distance from original start-point

segments, we spliced the i-th segments in each sub-trace and denote it as $X[i]$, respectively, we denoted $m[i]$ as $y[i]$. In this way, we obtained 32 pairs of training data like $(X[i], y[i])$ from each trace. In our experiments, we captured 1000 traces for training.

3) Model Training: With a training set of 32,000 data, we aimed to train a simple but still efficient neural network model to identify the 256 possible values of the message bytes encoded in masked Kyber. Following the structure and hyperparameters stated in Section III-B, we implemented an MLP model with Keras(version 2.8.0) [43], a widely used deep learning API written in Python. Since we had a trigger to enable trace capturing, we could precisely locate and divide traces. When we trained the model, getting satisfying results was to be expected. When training to 26-th epoch, the accuracy of the validation set had exceeded 99%, and when the number of epochs reached 47, the accuracy reached 100%. However, our attack doesn't rely on the trigger and exact division. To verify it, we set multiple experiments with different start-points, and the results are shown in Fig. 3. We found that the model achieves nearly 100% accuracy in an interval of up to 80 points, while it takes only 190 points to encode one byte on average.

C. key recovery

We used (k_u, k_v) in Table III to construct malicious ciphertext ct, and since $k = 3$, we needed to construct a total of 9 ciphertexts. We captured traces during decapsulating the 9 ciphertexts, and recover their corresponding messages in sequence. Following Table III, we use the i-th bit of every three messages to recover the i-th coefficient of polynomials of secret key $\mathbf{s}$. We repeated our key recovery experiment 1000 times. Finally, we found that there only were 239 error coefficients in all $1000*768$ coefficients, which means we only got 0.239 error coefficients in every key recovery. Our key recovery attack leaves a search space of only $\binom{1}{768} * 0.239 = 183.552$, much smaller than the results in [14].

To further narrow the search space, we applied our average method here. We captured 2000 traces and set $d = 2$, which means we averaged 2 output vectors to recover one message byte. In this case, we got 4 error coefficients during 1000 experiments. With the average method, to recover the secret key of masked Kyber768, we just need 18 traces and 3.072 times brute-force search.

The comparison with previous related work is summarized in Table IV.

TABLE IV
COMPARISON WITH PREVIOUS RELATED WORK

	Target	Number of traces		Success Rate
		Profiling	Attack	
Our work	Masked Kyber	1000	$3 * k$	99.97%
			$6 * k$	99.999%
[14]	Kyber	800	$480 * k$	98%
[31][a]	Masked Saber	1000	$36 * k$	99.37%

[a]They didn't give a success rate, we calculated this result based on their success rate on message recovery.

V. CONCLUSION

In this work, we analyzed the security of first-order masked message encoding and found it could defend against the previous proposed side-channel attacks on message encoding efficiently. However, exploitable leakage still exists, we introduced deep learning into side-channel attacks on message encoding and built a 2-stage attack that can recover the secret key with only 9 traces. Our approach combines information from two shares, and it is should be categorized as a high-order attack. Therefore, we didn't break the first-order security assumption, but as a general countermeasure, the first-order masked message encoder doesn't offer enough side-channel security.

In [24], Bos et al. implemented an arbitrary-order but high-cost masked encoder with secure conversions between Boolean and Arithmetic masking. Whether more cost leads to more security will be a meaningful topic that we will continue to explore in the future.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China (2020YFA0309704).

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[3] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, *et al.*, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020.

[4] C. Jerry, D. Oliver, and G. Jay, "Ibm quantum breaks the 100-qubit processor barrier," 2021. https://research.ibm.com/blog/127-qubit-quantum-processor-eagle.

[5] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, *et al.*, "Status report on the second round of the nist post-quantum cryptography standardization process," *US Department of Commerce, NIST*, 2020.

[6] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 353–367, IEEE, 2018.

[7] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 513–533, Springer, 2017.

[8] P. Pessl and R. Primas, "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.

[9] M. Hamburg, J. Hermelink, R. Primas, S. Samardjiska, T. Schamberger, S. Streit, E. Strieder, and C. van Vredendaal, "Chosen ciphertext k-trace attacks on masked cca2 secure kyber," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 88–113, 2021.

[10] C. Mujdei, A. Beckers, J. Bermundo, A. Karmakar, L. Wouters, and I. Verbauwhede, "Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication," *Cryptology ePrint Archive*, 2022.

[11] J.-P. D'Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede, "Timing attacks on error correcting codes in post-quantum schemes," in *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, pp. 2–9, 2019.

[12] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on cca-secure lattice-based pke and kems.," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 307–335, 2020.

[13] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma, "Curse of re-encryption: A generic power/em analysis on post-quantum kems," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 296–322, 2022.

[14] Z. Xu, O. M. Pemberton, S. S. Roy, D. Oswald, W. Yao, and Z. Zheng, "Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber," *IEEE Transactions on Computers*, 2021.

[15] P. Ravi, S. Bhasin, S. S. Roy, and A. Chattopadhyay, "On exploiting message leakage in (few) nist pqc candidates for practical message recovery attacks," *IEEE Transactions on Information Forensics and Security*, 2021.

[16] Q. Guo, T. Johansson, and A. Nilsson, "A key-recovery timing attack on post-quantum primitives using the fujisaki-okamoto transformation and its application on frodokem," in *Annual International Cryptology Conference*, pp. 359–386, Springer, 2020.

[17] S. Bhasin, J.-P. D'Anvers, D. Heinz, T. Pöppelmann, and M. Van Beirendonck, "Attacking and defending masked polynomial comparison for lattice-based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 334–359, 2021.

[18] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*, pp. 398–412, Springer, 1999.

[19] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu, "Practical cca2-secure and masked ring-lwe implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 1, pp. 142–174, 2018.

[20] T. Schneider, C. Paglialonga, T. Oder, and T. Güneysu, "Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto," in *IACR International Workshop on Public Key Cryptography*, pp. 534–564, Springer, 2019.

[21] F. Bache, C. Paglialonga, T. Oder, T. Schneider, and T. Güneysu, "High-speed masking for polynomial comparison in lattice-based kems," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 483–507, 2020.

[22] J.-P. D'Anvers, D. Heinz, P. Pessl, M. van Beirendonck, and I. Verbauwhede, "Higher-order masked ciphertext comparison for lattice-based cryptography," *Cryptology ePrint Archive*, 2021.

[23] J.-P. D'Anvers, M. Van Beirendonck, and I. Verbauwhede, "Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations," *Cryptology ePrint Archive*, 2022.

[24] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking kyber: First-and higher-order implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 173–214, 2021.

[25] T. Fritzmann, M. Van Beirendonck, D. Basu Roy, P. Karl, T. Schamberger, I. Verbauwhede, and G. Sigl, "Masked accelerators and instruction set extensions for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, pp. 414–460, 2021.

[26] M. V. Beirendonck, J.-P. D'anvers, A. Karmakar, J. Balasch, and I. Verbauwhede, "A side-channel-resistant implementation of saber," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 2, pp. 1–26, 2021.

[27] S. Kundu, J.-P. D'Anvers, M. Van Beirendonck, A. Karmakar, and I. Verbauwhede, "Higher-order masked saber," *Cryptology ePrint Archive*, 2022.

[28] D. Amiet, A. Curiger, L. Leuenberger, and P. Zbinden, "Defeating newhope with a single trace," in *International Conference on Post-Quantum Cryptography*, pp. 189–205, Springer, 2020.

[29] B.-Y. Sim, J. Kwon, J. Lee, I.-J. Kim, T.-H. Lee, J. Han, H. Yoon, J. Cho, and D.-G. Han, "Single-trace attacks on message encoding in lattice-based kems," *IEEE Access*, vol. 8, pp. 183175–183191, 2020.

[30] D. Heinz, M. J. Kannwischer, G. Land, T. Pöppelmann, P. Schwabe, and D. Sprenkels, "First-order masked kyber on arm cortex-m4," *Cryptology ePrint Archive*, 2022.

[31] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked ind-cca secure saber kem implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 676–707, 2021.

[32] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 3–26, Springer, 2016.

[33] K. Ngo, E. Dubrova, and T. Johansson, "Breaking masked and shuffled cca secure saber kem by power analysis," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, pp. 51–61, 2021.

[34] A. Roberto, B. Joppe, D. Léo, K. Eike, L. Tancrède, L. Vadim, M. S. John, S. Peter, S. Gregor, and S. Damien, "Crystals-kyber (version 3.02) – submission to round 3 of the nist post-quantum project," *Submission to the NIST Post-Quantum Cryptography Standardization Project*, 2020. https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[35] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Designs, Codes and Cryptography*, vol. 75, no. 3, pp. 565–599, 2015.

[36] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Annual international cryptology conference*, pp. 537–554, Springer, 1999.

[37] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the fujisaki-okamoto transformation," in *Theory of Cryptography Conference*, pp. 341–371, Springer, 2017.

[38] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2017.

[39] A.-S. Jacob, "Programmable hardware, microcontrollers and vector instructions," 2018. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/_0mDoyry1Ao/m/Tt7yHpjSDgAJ.

[40] "Cw1173 chipwhisperer-lite - newae hardware product documentation." https://rtfm.newae.com/Capture/ChipWhisperer-Lite.

[41] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4." https://github.com/mupq/pqm4.

[42] "Arm gnu toolchain." https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain.

[43] "Keras." https://github.com/keras-team/keras.