

# Secret Key Recovery Attacks on Masked and Shuffled Implementations of CRYSTALS-Kyber and Saber

Linus Backlund, Kalle Ngo, Joel Gärtner, Elena Dubrova  
KTH Royal Institute of Technology, Stockholm, Sweden  
`{lbackl,kngo,jgartner,dubrova}@kth.se`

**Abstract**—Shuffling is a well-known countermeasure against side-channel analysis. It typically uses the Fisher-Yates (FY) algorithm to generate a random permutation which is then utilized as the loop iterator to index the processing of the variables inside the loop. The processing order is scrambled as a result, making side-channel analysis more difficult. Recently, a side-channel attack on a masked and shuffled implementation of Saber requiring 61,680 power traces to extract the secret key was reported. In this paper, we present an attack that can recover the secret key of Saber from 4,608 traces. The key idea behind the 13-fold improvement is to recover FY indexes directly, rather than by extracting the message Hamming weight and bit flipping, as in the previous attack. We capture a power trace during the execution of the decapsulation algorithm for a given ciphertext, recover FY indexes 0 and 255, and extract the corresponding two message bits. Then, we modify the ciphertext to cyclically rotate the message, capture a power trace, and extract the next two message bits with FY indexes 0 and 255. In this way, all message bits can be extracted. By recovering messages contained in  $k * l$  chosen ciphertexts constructed using a new method based on error-correcting codes with length  $l$ , where  $k$  is the security level, we recover the long term secret key. To demonstrate the generality of the presented approach, we also recover the secret key from a masked and shuffled implementation of CRYSTALS-Kyber, which NIST recently selected as a new public-key encryption and key-establishment algorithm to be standardized.

**Index Terms**—Public-key cryptography, post-quantum cryptography, CRYSTALS-Kyber, Saber, side-channel attack, power analysis.

## I. INTRODUCTION

The National Institute of Standards and Technology (NIST) has recently selected one of the third round finalists in the Post-Quantum Cryptography (PQC) project, a Learning With Errors (LWE)-based scheme CRYSTALS-Kyber [1], as a Public-Key Encryption (PKE) and Key Encapsulation Mechanism (KEM) to be standardized [2]. The other two lattice-based finalists, an NTRU-based scheme NTRU [3] and a Learning With Rounding (LWR)-based scheme Saber [4], were dropped from the competition.

The motivation behind the PQC process is to choose new cryptographic primitives that will remain secure after the potential construction of a large scale quantum computer. The first two rounds of the NIST PQC selection process mainly focused on the theoretical security of the candidate designs and their implementation efficiency. The third round, however, put the spotlight on the security of the actual implementations. The resistance to side-channel attacks, which are considered as one of the main security threats to implementations of cryptosystems at present, has received particular attention. This topic is the focus of this paper.

An attack on a cryptosystem is usually either a Chosen Plaintext Attack (CPA) or a Chosen Ciphertext Attack (CCA). Indistinguishability under CPA (IND-CPA) means that one cannot distinguish two ciphertexts based on the messages they encrypt. This property is a basic requirement for most provably secure PKE schemes. Indistinguishability under adaptive CCA (IND-CCA2) extends this requirement further. By allowing the use of a decryption oracle that can decrypt any ciphertexts except the given ones, one cannot improve

the guess. The respective PKEs of the CRYSTALS-Kyber and Saber KEMs are IND-CPA secure. Utilizing a variation of the Fujisaki-Okamoto (FO) transform, they achieve IND-CCA2 security [5].

By re-encrypting the decrypted message and comparing the resulting ciphertext with the one received (yielding the true decrypted message only if they match) the IND-CCA2 schemes are protected against CCAs in theory. However, this may not hold for the implementations of the schemes. By recovering the decrypted message during algorithm execution via side-channels, the theoretical security provided by the FO transform can be bypassed. All cryptosystems run on physical devices that leak information through non-primary channels such as timing [6], power consumption [7], or electromagnetic (EM) emanations [8]. No physical device free of side-channels has yet been constructed. Instead countermeasures such as masking [9], shuffling [10], random delays insertion [11], etc. are used to combat side-channel leakage.

Masking [9] attempts to eliminate the leakage by partitioning a secret variable into two or more shares and executing all operations separately on the shares. Since the shares are randomized at each execution, none of them is expected to contain any exploitable information about the secret variable they mask. There are two main types of masking: Boolean and arithmetic. The Boolean masking uses the XOR to combine the shares into the original secret. The arithmetic masking uses the arithmetic addition.

Shuffling [10] is another well-known countermeasure applicable to operations on secret variables that are not dependent on each other. It typically uses the Fisher-Yates (FY) algorithm to create a random permutation which is then utilized as the loop iterator to index the processing of the variables inside the loop. By shuffling the order of the operations, the correlation between executed instructions and time can be hidden. Combined with masking, shuffling was previously believed to provide a sufficient protection against side-channel attacks. However, in [12] an attack defeating the combined protection was shown, which uses the bit flipping ciphertext malleability of LWE/LWR PKE/KEMs discovered in [13]. The attack requires 61,680 power traces to extract the secret key from a first-order masked and shuffled implementation of Saber on an ARM Cortex-M4.

A ciphertext malleability is a means of modifying the unknown message contained in a given ciphertext without re-encrypting it. The work of [13] presents two ways to accomplish this utilizing the fact that ciphertexts of LWE/LWR PKE/KEM schemes like e.g. CRYSTALS-Kyber and Saber correspond to polynomials in negacyclic polynomial rings. The first one is a method of flipping an individual message bit by adding a fixed value to the corresponding polynomial coefficient. The second one is a method of cyclically rotating the message by rotating the ciphertext polynomials.

In a KEM, a successfully recovered decrypted message trivially leads to a recovery of the shared session key (because it is derived from the message using hash functions). Furthermore, since the

security of the FO-transform is bypassed using side-channels, a set of Chosen Ciphertexts (CCTs) can be used to recover the long term secret key from the decrypted messages. A method for constructing CCTs assuming a perfect message recovery was presented in [13]. In [14], an Error-Correcting Code (ECC)-based CCT construction method was presented, which waives the requirement for a perfect message recovery.

**Our contributions:** In this paper, we demonstrate a side-channel attack on a first-order masked and shuffled implementation of Saber on an ARM Cortex-M4 which requires 4,608 power traces to extract the secret key. The key ideas behind the 13-fold improvement over the attack of [12] are:

- *A message recovery method using 0 and 255 FY indexes and cyclic rotations.* Rather than extracting the message Hamming Weight (HW) and bit flipping, as in [12], we recover “corner” FY indexes 0 and 255, extract the corresponding two bits of the message, and then cyclically rotate the message by modifying the ciphertext. In this way, all message bits can be extracted.
- *An incremental CCT construction method.* We construct CCTs iteratively, so that CCTs based on an ECC with code distance  $d + 1$  are a superset of the CCTs based on an ECC with code distance  $d$ . For a given implementation, there is an optimal code that minimizes the number of traces required for the attack. However, the optimal code is not known in advance. The presented CCT construction method reduces the total capture time for CCTs based on different codes.
- *A method for error-free cyclic rotation of CCTs.* It has previously been discussed how to cyclically rotate a message by modifying the corresponding ciphertext [13], [15]. However for arbitrary ciphertexts, the rotation may flip a wrapped-around message bit. We offer a solution tailored to the specifics of the presented CCT construction method.

To demonstrate the generality of the presented approach, we also extract the secret key from a first-order masked and shuffled implementation of CRYSTALS-Kyber, which has been recently selected for standardization by NIST [2]. Given that the National Security Agency (NSA) has included CRYSTALS-Kyber in the suite of cryptographic algorithms recommended for national security systems [16], it is important to thoroughly assess the security of CRYSTALS-Kyber implementations in order to improve the future versions.

In this paper we focus on the parametrizations of Saber and CRYSTALS-Kyber that use module rank  $k = 3$  and target NIST security level 3. The other security levels with  $k = 2$  and 4 can be treated similarly.

The rest of this paper is organized as follows. Section II describes previous work on protected implementations of Saber and CRYSTALS-Kyber as well as related side-channel attacks. Section III presents the Saber and CRYSTALS-Kyber algorithms. Section IV-A presents the equipment used for trace acquisition and the target implementations are defined in IV-B. Sections V and VI describe the profiling and the attack stages, respectively. Section VII summarizes the experimental results. Section VIII concludes the paper and describes future work.

## II. PREVIOUS WORK

This section describes previous work on protected implementations of Saber and CRYSTALS-Kyber and related side-channel attacks.

### A. Implementations

Several implementations of Saber [17], [18] and CRYSTALS-Kyber [19]–[21] protected by masking have been presented.

Masking brings a significant CPU time overhead to software implementations. Linear operations are repeated twice. Non-linear operations require even more complex solutions that decrease the speed substantially. For Saber, the most lightweight implementation, presented by Van Beirendonck et al. [17], has an overhead factor of 2.5 on an ARM Cortex-M4 compared to an unmasked one. This implementation employs a first-order masking of the Saber CCA-secure decapsulation algorithm. It is based on masked logical shifting on arithmetic shares and a masked binomial sampler.

The vulnerabilities discovered in the early version of the Saber implementation of Van Beirendonck et al. [17] helped improve the subsequently released versions of the implementation, as well as the higher-order masked implementation of Saber by Kundu et al. [18]. Some procedures with the bit-dependent leakage (easiest to exploit) were patched by re-implementing them to accumulate the message bits into a register before writing the entire byte into a memory. This results in a byte-dependent leakage, making side-channel analysis more difficult. Additionally, in the implementation [18], the procedure performing arithmetic to boolean conversion of shares was re-implemented to work in a bitsliced fashion. Stronger mitigation techniques against side-channel attacks, such as those presented in [22]–[24], were proposed to strengthen side-channel resistance of the NIST PQC candidates.

To the best of our knowledge, the implementation of Saber presented in [12] is the only available masked and shuffled implementation of an LWE/LWR PKE/KEM scheme.

### B. Attacks

Since both CRYSTALS-Kyber and Saber are based on module lattices, they have many similarities. Side-channel attacks on one are typically applicable to the other [13], [25], [26].

In [13], the authors discussed how to attack a masked implementation of LWE/LWR PKE/KEMs in two steps by recovering each share separately using templates created on traces with known masks, and then combining the shares.

The first attack on a first-order masked implementation of Saber KEM was presented in [14]. It recovers a message from a single trace with a high probability using a neural network trained at the profiling stage. The key idea is to recover messages in one step, without explicitly extracting random masks at each execution. It was also shown in [14] how to recover the secret key from 24 power traces captured during the execution of the decapsulation procedure for CCTs. The ciphertexts are constructed using an ECC-based method which can correct some errors in the recovered messages. In [27] the attack was extended to the implementation of Saber from [18].

An attack applying the method of [14] to a first-order masked implementation of CRYSTALS-Kyber was presented in [25], targeting the message encoding vulnerability found in [28]. In [29], it was demonstrated that the method of [14] can also be used to break a higher-order masked implementation of Saber. It was shown that the neural networks are capable to recover more than two shares and then XORs them to get the message. In [30], side-channel attacks on two implementations of masked polynomial comparison were demonstrated on the example of CRYSTALS-Kyber.

The closest related work to the presented one is [12] where a side-channel attack on a first-order masked and shuffled implementation of Saber is described. The attack requires  $257 \times N$  power traces to recover a message  $m$  and  $24 \times 257 \times N$  power traces to recover the secret key, where  $N$  is the number of repetitions of the same measurement. In [12],  $N = 10$  is used. Similarly to the method of [14], a neural network trained at the profiling stage is used to

<b>CPA-PKE.KeyGen()</b> 1: $seed_A \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 2: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k \times k}; seed_A)$ 3: $\mathbf{s} \leftarrow \chi_1(R_q^{k \times 1})$ 4: $\mathbf{e} \leftarrow \chi_2(R_q^{k \times 1})$ 5: $\mathbf{b} = [\mathbf{As} + \mathbf{e}]_{p_1}$ 6: $pk = (seed_A, \mathbf{b}), sk = \mathbf{s}$ 7: <b>return</b> $(pk, sk)$	<b>CPA-PKE.Enc(<math>pk = (seed_A, \mathbf{b}), m, r</math>)</b> 1: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k \times k}; seed_A)$ 2: $\mathbf{s}' \leftarrow \chi_1(R_q^{k \times 1}; r)$ 3: $\mathbf{e}' \leftarrow \chi_3(R_q^{k \times 1}; r)$ 4: $\mathbf{e}'' \leftarrow \chi_3(R_q^{k \times 1}; r)$ 5: $\mathbf{u} = [\mathbf{As}' + \mathbf{e}']_{p_2}$ 6: $\mathbf{v}' = [\mathbf{b} \cdot q/p_1] \mathbf{s}' + \mathbf{e}''$ 7: $v = [v' + \text{encode}(m)]_{p_3}$ 8: <b>return</b> $c = (\mathbf{u}, v)$
<b>CPA-PKE.Dec(<math>s, c = (\mathbf{u}, v)</math>)</b> 1: $x = [v \cdot q/p_3] - s \lfloor u \cdot q/p_2 \rfloor$ 2: $m' = \text{decode}(x)$ 3: <b>return</b> $m'$	

Fig. 1: Pseudocode of CPA-PKE algorithms.

<b>CCA-KEM.KeyGen()</b> 1: $z \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 2: $(pk, s) = \text{CPA-PKE.KeyGen}()$ 3: $sk = (s, pk, \mathcal{H}(pk), z)$ 4: <b>return</b> $(pk, sk)$	<b>CCA-KEM.Decaps(<math>sk = (s, pk, \mathcal{H}(pk), z), c</math>)</b> 1: $m' = \text{CPA-PKE.Dec}(s, c)$ 2: $(\hat{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$ 3: $c' = \text{CPA-PKE.Enc}(pk, m', r')$ 4: <b>if</b> $c = c'$ <b>then</b> 5: <b>return</b> $K = \text{KDF}(\hat{K}, \mathcal{H}(c))$ 6: <b>else</b> 7: <b>return</b> $K = \text{KDF}(z, \mathcal{H}(c))$ 8: <b>end if</b>
<b>CCA-KEM.Encaps(<math>pk</math>)</b> 1: $m \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 2: $(\hat{K}, r) = \mathcal{G}(m, \mathcal{H}(pk))$ 3: $c = \text{CPA-PKE.Enc}(pk, m, r)$ 4: $K = \text{KDF}(\hat{K}, \mathcal{H}(c))$ 5: <b>return</b> $(c, K)$	

Fig. 2: Pseudocode of CCA-KEM algorithms.

recover each bit of  $m$ . However, since the bits are shuffled, their order is unknown. Thus, only the HW of  $m$ ,  $HW(m)$ , can be deduced in this way. To find the values of individual bits of  $m$ , 256 additional traces are captured for messages  $m'$ , in which one bit of  $m$  is complemented using the bit flipping method of [13]. The  $HW(m')$  is then recovered and  $m$  is deduced by comparing  $HW(m)$  and  $HW(m')$ .

### III. SABER AND CRYSTALS-KYBER ALGORITHMS

In this section, we briefly describe CRYSTALS-Kyber and Saber algorithms. For more details, the reader is referred to [1] and [4].

Fig. 1 and 2 show pseudocodes of CPA-PKE and CCA-KEM algorithms, respectively, which are applicable to both CRYSTALS-Kyber and Saber [31]. CPA-PKE contains three algorithms: key generation, CPA-PKE.KeyGen; encryption, CPA-PKE.Enc; and decryption, CPA-PKE.Dec. CCA-KEM also contains three algorithms: key generation, CCA-KEM.KeyGen; encapsulation, CCA-KEM.Encaps; and decapsulation, CCA-KEM.Decaps.

The ring  $R_q$  in CPA-PKE is the quotient ring  $\mathbb{Z}_q[X]/(X^{256} + 1)$ , where  $\mathbb{Z}_q$  is the ring of integers modulo a positive integer  $q$ . Sampling  $v$  from a distribution  $\chi_i$  over a set  $S$  is denoted by  $v \leftarrow \chi_i(S)$  while  $v \leftarrow \chi_i(S; r)$  denotes deterministic sampling from  $\chi_i$  using seed  $r$ . The uniform distribution is denoted by  $\mathcal{U}$  and  $\lfloor v \rfloor_p = [v \cdot (p/q)]$ .

The encode function in CCA-PKE encodes a message to a polynomial by letting each coefficient of the polynomial to be equal to the corresponding bit of the message times  $\lfloor p_3/2 \rfloor$ . Similarly, the decode function decodes a polynomial to a message by letting each bit of the message be determined by the corresponding polynomial coefficient. If the coefficient is closer to  $\lfloor q/2 \rfloor$  than to 0, the message bit is 1. Otherwise, the message bit is 0.

Both CRYSTALS-Kyber and Saber can be seen as different parametrizations of CCA-KEM, with the security levels of the schemes mainly differing by the rank  $k$  of the module that they use.

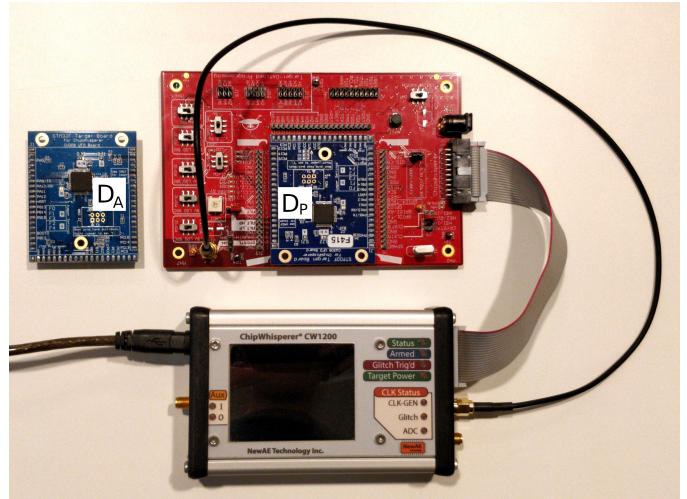


Fig. 3: Equipment for trace acquisition.

For the parametrizations considered in this paper, both CRYSTALS-Kyber and Saber use  $k = 3$ .

The most significant difference between CRYSTALS-Kyber and Saber is in the type of distributions  $\chi_i$  that the schemes use and in rounding parameters  $p_1, p_2, p_3$ . In CRYSTALS-Kyber, all distributions  $\chi_i$  are centered binomial distributions with  $p_1 = 1, p_2 > 1$  and  $p_3 > 1$ . In Saber, only  $\chi_1$  is a centered binomial distribution while samples from the other distributions  $\chi_i$  are always equal to 0. This is compensated by using larger rounding, with  $p_1 = p_2$  greater than 1 and  $p_3$  significantly greater than 1.

### IV. EXPERIMENTAL SETUP

This section presents the equipment which we use for trace acquisition and the target implementations of Saber and CRYSTALS-Kyber.

#### A. Equipment

For trace acquisition, we use a ChipWhisperer-Pro, a CW308 UFO board and two CW308T-STM32F4 target boards, one for profiling,  $D_P$ , and another for the attack,  $D_A$ , see Fig. 3. To verify that the negative effect of boards diversity, which would be expected in a real attack scenario, is not preventing the presented attack, we selected as  $D_P$  and  $D_A$  a pair of boards acquired from different chip vendors, and having a different age and wear-out.

A similar equipment is used in the attack on Saber from [12] except that in [12] ChipWhisperer-Lite is used instead of ChipWhisperer-Pro. ChipWhisperer-Pro has a large buffer size of 98K samples. The buffer size of ChipWhisperer-Lite is 24K samples.

The target board CW308T-STM32F4 contains an ARM Cortex-M4 CPU with an STM32F415-RGT6 chip. It is run at 24 MHz. The traces are sampled at 24MHz for CRYSTALS-Kyber and 72MHz for Saber. The choice of sampling rate is limited by the size of ChipWhisperer-Pro buffer<sup>1</sup>. For CRYSTALS-Kyber, all points of interest do not fit into the buffer if a higher sampling rate is used.

#### B. Target implementations

To the best of our knowledge, there are no publicly available implementations of CRYSTALS-Kyber protected by both masking

<sup>1</sup>ChipWhisperer-Pro has an option of streaming, however, streaming can be used only at a maximum of 10 MHz sampling frequency.

```

void FY_Gen(uint8* permutation, int max)
1: for (i = 0; i < max; i++) do
2:   permutation[i] = i;
3: end for
4: for (i = max - 1; i > 0; i=i-1) do
5:   int index = rand() % (i+1);
6:   uint8 temp = permutation[index];
7:   permutation[index] = permutation[i];
8:   permutation[i] = temp;
9: end for

void masked_poly_tomsg(uint8 msg[2][32],
uint16 poly[2][256])
uint16 c[2];
uint8 permutation[256];
1: FY_Gen(permutation, 256);
2: for (x = 0; x < 256; x++) do
3:   x_rand = permutation[x];
4:   i = x_rand / 8;
5:   j = x_rand % 8;
6:   ... Processing ...
7:   msg[0][i] += ((c[0] >> 15) & 1)<<j;
8:   msg[1][i] += ((c[1] >> 15) & 1)<<j;
9: end for

```

Fig. 4: Modified C code of `masked_poly_tomsg()` procedure of masked CRYSTALS-Kyber with shuffling added.

and shuffling. The experiments presented in this paper are performed using the C implementation which we built on the top of the first-order masked implementations of CRYSTALS-Kyber from [21].

The attack point of the presented side-channel attack is the procedure `masked_poly_tomsg()`. It is called by `CPA-PKE.Dec()` during the decapsulation (step 1 of `CCA-KEM.Decaps()` in Fig. 2). The modified code of `masked_poly_tomsg()`, with shuffling added, is shown in Fig. 4. The lines marked in blue and red indicate vulnerabilities exploited in the presented attack. The first line in blue (line 1) is a call to `FY_Gen()` which creates a random permutation of 256 indexes. In the second line in blue (line 3), the FY indexes are used to define the processing order of message bits in the **for**-loop. The side-channel leakage from both parts is used in the presented attack for FY index recovery. In red color are the lines representing the processing of indexed message bits. The side-channel leakage from this part is used for recovering the message.

For Saber, we used the same C implementation as in the attack of [12] upgraded to the latest release. The attack point is the procedure `poly_A2A()`.

The C implementations of Saber and CRYSTALS-Kyber were compiled using `arm-none-eabi-gcc` with the optimization level `-O3` (recommended default).

## V. PROFILING STAGE

Our profiling strategy is similar to [12] except that we train two types of neural networks: one for FY index recovery,  $N_{FY}$ , and another for message bit recovery,  $N_m$ . We use the same three dense layer Multilayer Perceptron (MLP) architecture as in [12], but with different layer widths (see Tables I and II). For Saber, we use dense layers with the widths 512, 256 and 128 for FY index recovery and with the widths 256, 128 and 64 for message recovery. For CRYSTALS-Kyber, we use dense layers with the widths 1024, 512

TABLE I: The MLP architecture.

Layer type	(Input, output) shape
Batch Normalization 1	(A, A)
Dense 1	(A, B)
Batch Normalization 2	(B, B)
ReLU	(B, B)
Dense 2	(B, C)
Batch Normalization 3	(C, C)
ReLU	(C, C)
Dense 3	(C, D)
Batch Normalization 4	(D, D)
ReLU	(D, D)
Dense 4	(D, E)
Softmax	(E, E)

TABLE II: The MLP layer widths.

Algorithm	Model	A	B	C	D	E
Saber	$N_{FY}$	215	512	256	128	256
	$N_m$	35	256	128	64	2
CRYSTALS-Kyber	$N_{FY}$	820	1024	512	256	256
	$N_m$	225	512	128	64	2

and 256 for FY index recovery and with the widths 512, 128 and 64 for message recovery. For CRYSTALS-Kyber two separate models  $N_m$  are trained for the first and the last bits of a byte (because the leakage type is different).

We train on 15K traces for Saber and 50K traces for CRYSTALS-Kyber, captured from the profiling device  $D_P$ . As in [12], we cut-and-join traces to increase the training set without having to capture more traces. Unlike in [12], we do not use traces captured from the device under attack  $D_A$  for profiling.

In addition, we use standardization. Given a set of traces  $\mathbf{T}$  with elements  $T = (t_1, \dots, t_{|T|})$ , each  $T \in \mathbf{T}$  is standardized to  $T' = (t'_1, \dots, t'_{|T|})$  such that:

$$t'_i = \frac{t_i - \mu_i}{\sigma_i},$$

where and  $\mu_i$  and  $\sigma_i$  are the mean and the standard deviation of the elements of  $\mathbf{T}$  at the  $i$ th data point,  $i \in \{1, \dots, |T|\}$ .

The implementation of the index generation function `FY_Gen()` which we use is not constant-time. Thus, the traces need to be synchronized before training. We perform this by cutting the segments corresponding to each FY index at points identified through correlation.

For FY index recovery, the input to neural networks is a concatenation of two intervals covering the FY index generation part and the FY index usage in the inner loop. Each of those intervals covers the target index and its both neighbours. For message bit recovery, the input trace to neural networks is an interval covering the processing of both shares for the targeted message bit.

To find where the FY indexes are generated, we apply Welch's t-test [32] to a set of 5K traces  $\mathbf{T}$  with known FY indexes captured from  $D_P$  during the execution of `CCA-KEM.Decaps()`. For each  $i \in \{0, 1, \dots, 255\}$ , we partition  $\mathbf{T}$  into two subsets such as:

$$\begin{aligned} \mathbf{T}_0 &= \{T_j \in \mathbf{T} \mid HW(FY_j[i]) < 4\}, \\ \mathbf{T}_1 &= \{T_j \in \mathbf{T} \mid HW(FY_j[i]) > 4\}, \end{aligned}$$

where  $HW(FY_j[i])$  is the HW of the FY index of  $i$ th processed bit of message  $m_j$  in trace  $T_j$ , for all  $j \in \{1, 2, \dots, |T|\}$ .

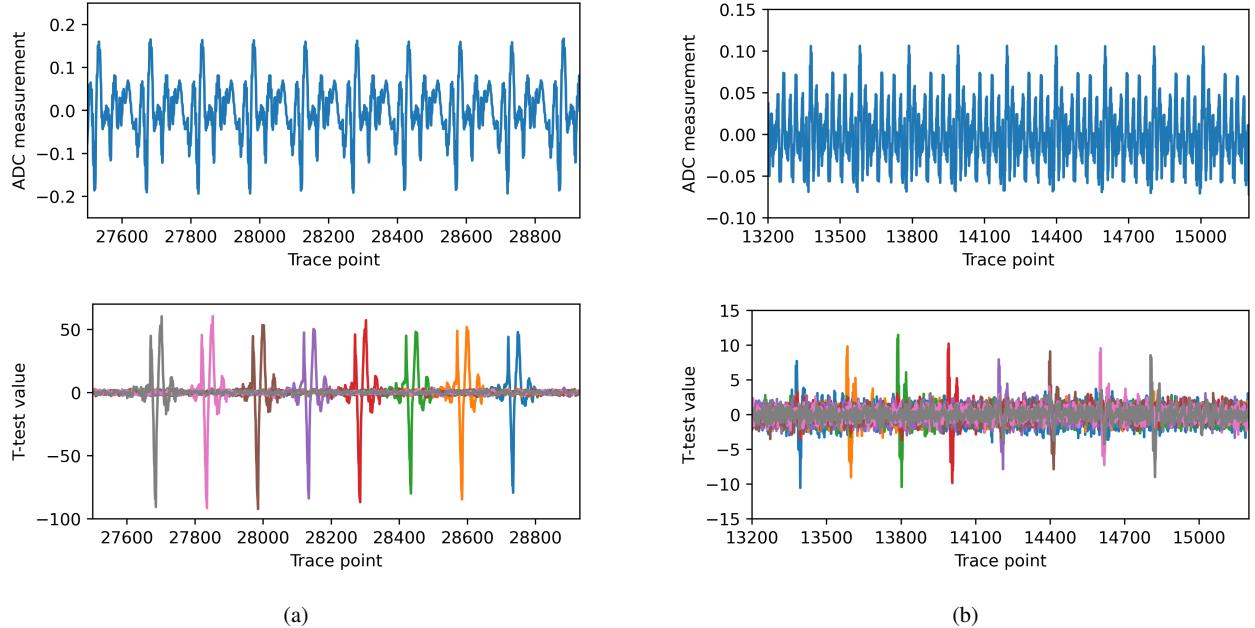


Fig. 5: (a) An average trace representing the generation of the FY indexes 64-72 in Saber (top) and its t-test (bottom); (b) An average trace representing the usage of the indexes during message processing (top) and its t-test (bottom). Both t-tests are performed on 5K traces.

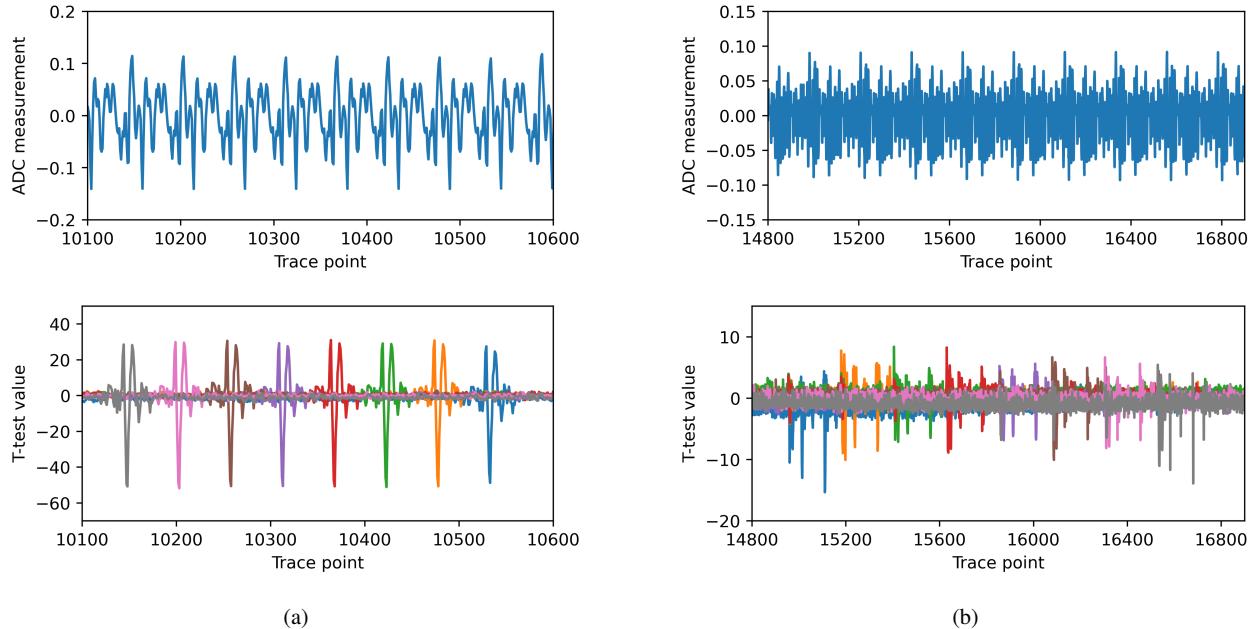


Fig. 6: (a) An average trace representing the generation of the FY indexes 64-72 in CRYSTALS-Kyber (top) and its t-test (bottom); (b) An average trace representing the usage of the indexes during message processing (top) and its t-test (bottom). Both t-tests are performed on 5K traces.

The Welch's t-test determines if there is a noticeable difference in the means of  $\mathbf{T}_0$  and  $\mathbf{T}_1$ , by computing:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{|\mathbf{T}_0|} + \frac{\sigma_1^2}{|\mathbf{T}_1|}}},$$

where  $\mu_i$  and  $\sigma_i$  are the mean and the standard deviation of the set

$\mathbf{T}_i$ , for  $i \in \{0, 1\}$ .

Fig. 5(a) and 6(a) show the t-test results for the index generation part of Saber and CRYSTALS-Kyber, respectively. We can see that in both cases the leakage is of similar type. This is because Saber and CRYSTALS-Kyber use the same implementation of `FY_Gen()`. The higher t-test values for Saber are probably due to oversampling. With three samples taken per clock cycle, the probability to catch a

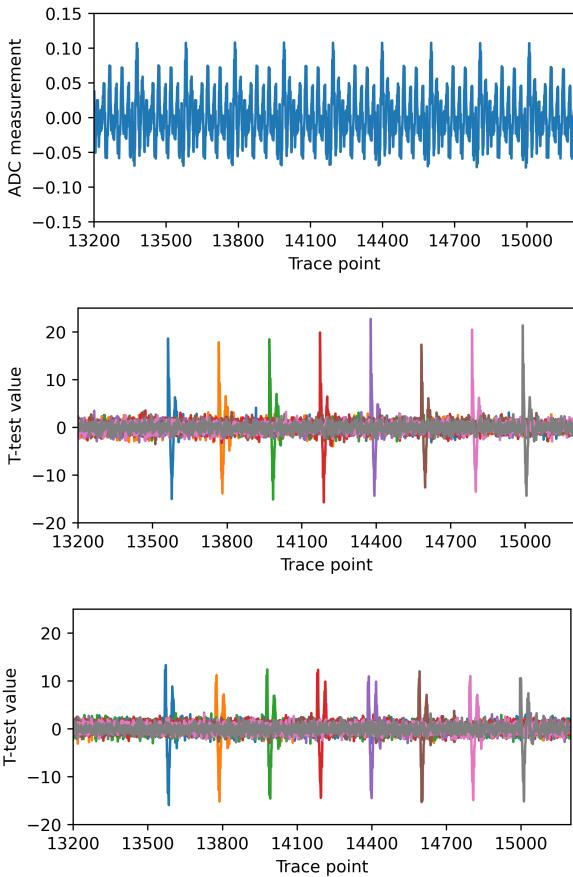


Fig. 7: An average trace representing the processing of masked message bits 64-72 in Saber (top), t-test for share 1 (middle) and for share 2 (bottom). Both t-tests are performed on 1K traces.

point with the strongest leakage is higher.

In a similar way, one can locate a segment of traces corresponding to the usage of FY indexes in the inner loop, see Fig. 5(b) and 6(b) for Saber and CRYSTALS-Kyber, respectively. We can see that the leakage here is considerably weaker than the one in the index generation part. Still, making use of both segments helps us maximize the predication accuracy of neural networks.

The message bits are recovered from the same trace segments where FY indexes are used. Fig. 7 and 8 show the t-test results for both shares for Saber and CRYSTALS-Kyber, respectively. The t-test is performed on 1K traces with known messages, masks and FY indexes captured from  $D_P$ . For each  $i \in \{0, 1, \dots, 255\}$ , we partition the set into two subsets according to the value of  $i$ th bit of the share.

From the t-tests in Fig. 7 and 8 we can see that Saber and CRYSTALS-Kyber implementations have different types of leakage during message processing. In Saber, the leakage is stronger because `poly_A2A()` procedure decodes the message bits one-by-one and stores them in a memory in an unpacked fashion. Thus, the leakage patterns of all message bits are similar. Contrary, in CRYSTALS-Kyber, the message bits are accumulated into a byte array in memory during their processing by `masked_poly_tomsg()` procedure. As a result, within each byte, the bit accumulated first leaks stronger than the bit accumulated last. This makes message recovery more difficult.

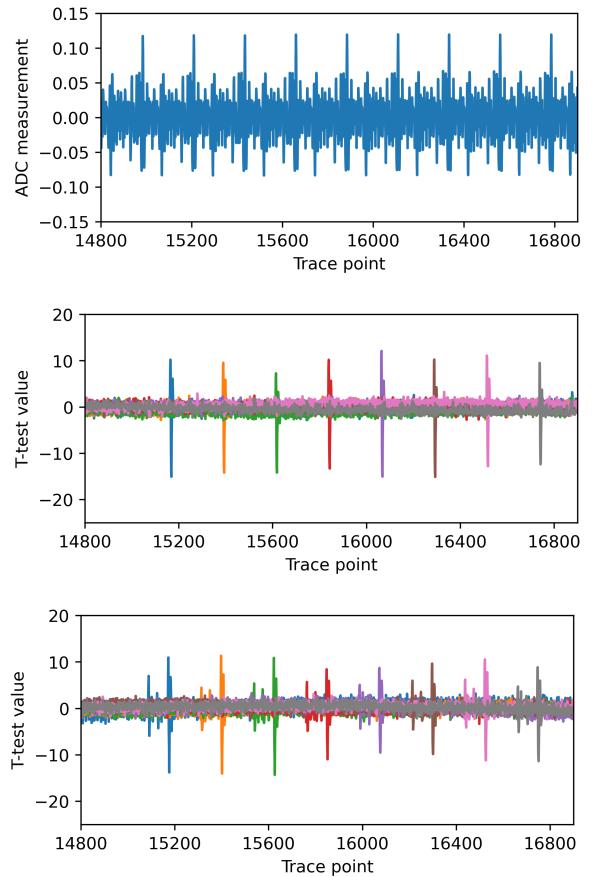


Fig. 8: An average trace representing the processing of masked message bits 64-72 in CRYSTALS-Kyber (top), t-test for share 1 (middle) and for share 2 (bottom). Both t-tests are performed on 1K traces.

The order in which the bits are set is randomized for each execution due to shuffling resulting in the t-test showing the average leakage instead of decreasing for each bit.

For CRYSTALS-Kyber, we train  $N_{FY}$  and  $N_m$  models on trace segments covering FY index generation and usage, and message processing, respectively, without any modifications. For Saber, however, we trim some redundant input data points which we identify using the stuck-at-0 fault method of [33] and retrain to improve the accuracy. We remove all points whose assignment to 0 decrease the prediction accuracy of  $N_{FY}$  and  $N_m$  less than 0.5% and 0.01%, respectively. We also applied trimming to CRYSTALS-Kyber, but the attack results got worse. A higher effect of trimming on Saber is likely to be due to oversampling.

For each case,  $N_{FY}$  and  $N_m$ , we train ten models. At the attack stage, these models are used in an ensemble. For the index prediction, the output of the ensemble is determined by multiplying the probabilities of score vectors of all ten models  $N_{FY}$ . For the message bit prediction, the output of the ensemble is obtained by majority voting on the bits predicted only by models  $N_m$  with a confidence level higher than 0.9.

Additional models are trained specifically for the first and last FY indexes and message bits.

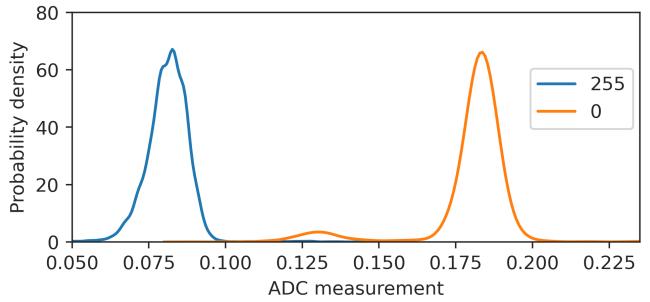
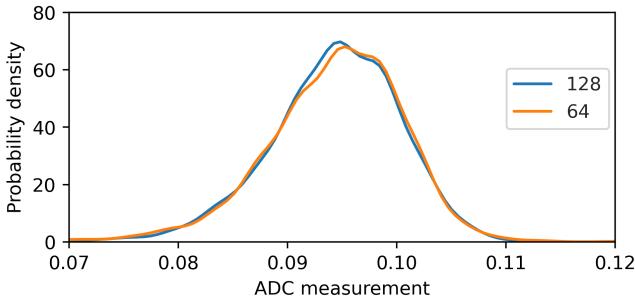


Fig. 9: Distribution of power consumption during the generation of FY indexes 64 and 128 (left) and 0 and 255 (right) at a trace point with the maximum absolute t-test value in Saber.

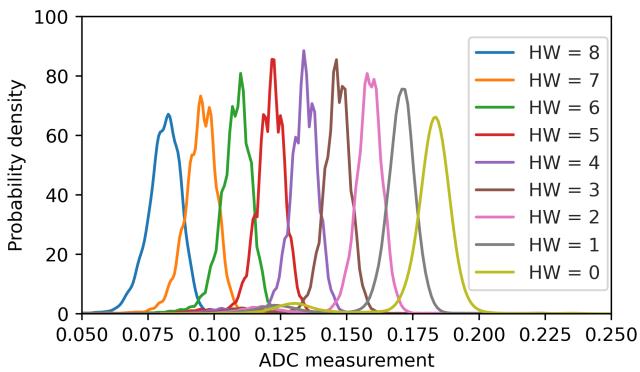


Fig. 10: Distribution of power consumption during the generation of FY indexes with the Hamming weight 0-8 at a trace point with the maximum absolute t-test value in Saber.

## VI. ATTACK STAGE

### A. Message recovery using 0 and 255 FY indexes and cyclic rotations

If it was possible to recover all FY indexes from power traces during their generation or inner loop usage, one could recover a message by first extracting its bits in the unknown order, then recovering the FY indexes of the bits, and finally re-ordering the bits accordingly. However, we found that neural networks cannot classify all possible FY indexes with a high probability from a single power trace. This is due to the high overlap in the distributions of power consumption of some indexes, see Fig. 9. Only the “corner” indexes 0 and 255 can be distinguished with a high probability because their distributions are nearly disjoint. For this reason, we use the following method for message recovery.

To recover the message  $m$  encrypted in a given ciphertext  $c$ , we capture a power trace during the decapsulation of  $c$ , recover FY indexes 0 and 255 using neural networks for index recovery,  $N_{FY}$ , trained at the profiling stage, and extract the corresponding two bits of  $m$  using neural networks for message recovery,  $N_m$ , trained at the profiling stage. Then, we modify  $c$  to  $c'$  which encrypts  $m$  negacyclically rotated by two bit positions, capture a power trace during the decapsulation of  $c'$ , and extract the next two message bits with FY indexes 0 and 255. By negacyclically rotating  $m$  by two bit positions 128 times, all message bits are extracted in this way.

### B. Cyclic rotation of CCTs

It is known [13], [15] that a message  $m$  of a ring-based LWE/LWR can be cyclically rotated by manipulating a ciphertext  $c$  encrypting it. However, an aspect not covered earlier is that the output values computed by  $\text{decode}(-x)$  and  $\text{decode}(x)$  can be different, where  $x$  is defined in line 1 of CPA-PKE.Dec() in Fig. 1. The function  $\text{decode}(x)$  essentially decodes a bit of the message by deciding if  $x$  is closer to 0 or  $q/2$ , but for neither CRYSTALS-Kyber nor Saber this function is completely symmetric, meaning that  $x$  can be closer to 0 while  $-x = q - x$  is decoded as if it is closer to  $q/2$ . This introduces a source of potential errors during negacyclic rotation of the CCTs. We fix this by rotating CCTs in a way adopted to their specific construction.

A ciphertext  $c = (\mathbf{u}, v)$  consists of polynomials in the ring  $\mathbb{Z}_q[X]/(X^{256} + 1)$ . Cyclic rotation of the message  $m$  encrypted in a properly generated  $c$  is performed by multiplying both  $\mathbf{u}$  and  $v$  by indeterminant  $X$ , corresponding to a negacyclic rotation of the polynomial coefficients. For the CCTs, we instead only multiply  $\mathbf{u}$  by  $X$ . In general, this is not equivalent to a cyclic rotation of  $m$ . However, in this way we can control which coefficient of the secret key  $s$  affects a given bit of  $m$  in the CCTs. Since we construct the CCTs so that a specific message bit  $m[i]$  is determined by a specific key coefficient  $s[i]$ , this is sufficient for the full secret key recovery. Care should be taken to keep track of whether  $s[i]$  has wrapped around modulo  $n$  or not. In the former case, the message bit is determined by  $-s[i]$  and not  $s[i]$ .

### C. Incremental CCT construction method

In the ECC-based CCT construction method of [14], the secret key coefficients are mapped into the codewords of an  $[8, 4, 4]$  extended Hamming code<sup>2</sup>. We found that the total number of traces required for secret key recovery can be reduced if more powerful codes are used. We use  $[l, w, d]$  linear codes with code distances up to 6 for Saber and up to 8 for CRYSTALS-Kyber. We designed incremental mapping tables in which CCTs based on an ECC with code distance  $d + 1$  are a superset of the CCTs based on an ECC with distance  $d$ . Our experimental results show that, for a given implementation, there is an optimal code minimizing the number of attack traces. However, the optimal code is not known in advance. The incremental CCT construction method helps reduce the total capture time for CCTs based on different codes. An attacker can capture as many CCT traces

<sup>2</sup>In the notation  $[l, w, d]$ ,  $l$  is the codeword length,  $w$  is the dataword length, and  $d$  is the code distance. A code distance is the minimum Hamming distance between any two codewords of the code.

TABLE III: CCT construction table for Saber.

Order of codeword bits for a code with distance $d$				CCT constants	Mapping of message bits into secret key coefficients								
6	5	4	3	$(k_1, k_0)$	-4	-3	-2	-1	0	1	2	3	4
1	1	7	6	(240,10)	1	1	0	0	1	1	0	0	1
2	5			(377,10)	0	0	1	0	1	1	0	1	0
3	7	1	4	(613,4)	1	0	1	0	1	1	0	1	0
4	2	3	1	(373,15)	1	0	1	1	0	1	0	0	1
5	10	6	3	(913,15)	1	1	1	0	0	0	0	1	1
6	11	8	7	(12,3)	1	0	0	0	0	0	0	0	0
7	3	4	2	(793,10)	1	0	0	1	1	0	0	1	1
8				(755,4)	0	1	0	0	1	1	0	0	1
9	9			(917,10)	0	1	1	1	1	0	0	0	0
10	6			(806,10)	0	0	0	1	1	0	0	1	1
11	8	5		(456,15)	1	1	0	1	0	1	0	1	0
12	4	2	5	(68,4)	1	1	1	1	1	0	0	0	0

TABLE IV: CCT construction table for CRYSTALS-Kyber.

Order of codeword bits for a code with distance $d$						CCT constants	Mapping of message bits into secret key coefficients					
8	7	6	5	4	3	2	$(k_2, k_1, k_0)$	-2	-1	0	1	2
1	1	2	9	6	1		(1,153,8)	0	1	1	1	0
2	10	8	1	7	5	2	(0,77,5)	1	1	1	0	0
3	5	1	6	5	2	1	(2,335,15)	1	1	0	1	1
4	7	11	2	2	6		(3,432,3)	0	1	0	0	1
5	2	3	3	4	3	4	(3,606,3)	1	0	0	1	0
6	8	6	10				(1,864,8)	0	1	1	1	0
7	3	4	5	3			(0,915,5)	0	0	1	1	1
8	4	5					(0,898,5)	0	0	1	1	1
9	11	9	4	1	4	3	(3,432,8)	1	0	1	0	1
10	12						(0,105,5)	1	1	1	0	0
11	9	7	8				(3,632,3)	1	0	0	1	0
12							(3,386,3)	0	1	0	0	1
13	13						(3,606,8)	1	0	1	0	1
14	6	10	7				(2,321,15)	1	1	0	1	1

as the access time to the device under attack allows, and then try different codes.

The presented method uses  $3l$  ciphertexts to recover the secret key coefficients  $s[256r+i]$ , for all  $i \in \{1, 2, \dots, 256\}$  and  $r \in \{0, 1, 2\}$ . The coefficient  $s[256r+i]$  is derived from the codeword  $(m_{rl+1}[i], \dots, m_{(r+1)l}[i])$  of an  $[l, w, d]$  linear code using the mapping defined by Tables III and IV for Saber and CRYSTALS-Kyber, respectively.

For all  $j \in \{1, 2, \dots, l\}$ , the message  $m_{rl+j}$  is recovered from the CCT  $c_{rl+j} = (\mathbf{u}, v)$  which is constructed as

$$\mathbf{u} = \begin{cases} (k_1, 0, 0) \in R_q^{3 \times 1} & \text{for } r = 0 \\ (0, k_1, 0) \in R_q^{3 \times 1} & \text{for } r = 1 \\ (0, 0, k_1) \in R_q^{3 \times 1} & \text{for } r = 2 \end{cases}$$

and

$$v = \begin{cases} k_0 \sum_{i=0}^{255} X^i & \text{for Saber} \\ k_0 + (k_2 \sum_{i=1}^{254} X^i) + k_0 X^{255} & \text{for CRYSTALS-Kyber} \end{cases}$$

for all  $r \in \{0, 1, 2\}$ , where the constants  $(k_2, k_1, k_0)$  are defined in Tables III and IV.

The non-empty entries in the first multi-column of Tables III and IV define indexes  $j$  of the CCTs  $c_{rl+j}$ , i.e. the order of codeword bits. Each column contains  $l$  non-empty entries, where  $l$  is the codeword length. For example, in Table IV, for  $d = 2$ , the codeword length is 4, so there are 4 non-empty entries,  $j \in \{2, 1, 4, 3\}$ , in the column.

TABLE V: Success rate of CRYSTALS-Kyber secret key recovery using an ECC with code distance  $d$  and  $N$  repetitions for 10 attacks.

N	d	# Incorrect key coeff. (mean)		Attack time (the worst case)		
		Undetected	Detected	Capture	Message rec.	Enum.
30	8	0	0	19.5 h	20.1 h	0 sec
	6	0	0.2	15.3 h	15.8 h	0.01 sec
	4	0	2.1	9.8 h	10.0 h	0.08 sec
	2	17.3	0.9	5.6 h	5.8 h	0.02 sec
20	8	0	0	13.0 h	13.4 h	0 sec
	6	0	0.4	10.2 h	10.5 h	0.01 sec
	4	0.1	5.4	6.5 h	6.7 h	15 sec
	2	29.1	3.0	3.7 h	3.9 h	0.4 sec
10	8	0	1.8	6.5 h	6.7 h	0.05 sec
	6	0.1	9.1	5.1 h	5.3 h	96 min
	4	0.4	27.8	3.3 h	3.4 h	-
	2	73.1	15.8	1.9 h	2.0 h	-

TABLE VI: Success rate of Saber secret key recovery using an ECC with code distance  $d$  and  $N$  repetitions for 10 attacks.

N	d	# Incorrect key coeff. (mean)		Attack time (the worst case)		
		Undetected	Detected	Capture	Message rec.	Enum.
3	6	0	0	4.0 h	26.0 min	0 sec
	4	0	0.3	2.7 h	17.3 min	0.01 sec
2	6	0	1.7	3.3 h	17.3 min	10.5 sec
	4	0.4	5.7	2.2 h	11.6 min	12 min
1	6	0	4.9	3.0 h	8.7 min	2 min
	4	0.7	13.2	2.0 h	5.8 min	-

For each non-empty entry  $j$ , the constants  $(k_2, k_1, k_0)$  listed in the second column at the same line as  $j$  define  $\mathbf{u}$  and  $v$  parts of  $c_{rl+j}$ . In the previous example, CCTs  $c_{4r+1}, c_{4r+2}, c_{4r+3}, c_{4r+4}$ , are constructed using the constants  $(2, 335, 15), (0, 77, 5), (3, 432, 8), (3, 606, 3)$ , respectively.

Similarly, the message bits listed in the third multi-column at the same line as non-empty entries  $j$  compose a codeword which determines the secret key coefficient. In the previous example, if the codeword is  $(0, 1, 1, 0)$ , then the secret key coefficient is 0. If the codeword is  $(1, 0, 1, 0)$ , then the secret key coefficient is 2.

Similarly to [14], we select the constants  $k_1$  and  $k_0$  so that each secret key coefficient is uniquely mapped into some codeword composed from the decrypted message bits. For CRYSTALS-Kyber, in which the leakage of different bits within a byte is non-uniform, we also use one more constant,  $k_2$ , which allows us to minimize the HW of messages. This helps us reduce the interference of previously accumulated bits on the success rate of the bit recovery.

We found that, in the masked implementation of CRYSTALS-Kyber from [21], the decoding does not perfectly match the decoding in the unprotected reference implementation [1]. To handle this, we select the constants  $(k_2, k_1, k_0)$  for CRYSTALS-Kyber so that  $\text{decode}(x \pm \varepsilon) = \text{decode}(x)$  for small  $\varepsilon$ , resulting in equivalent ciphertexts in both implementations.

## VII. EXPERIMENTAL RESULTS

For both Saber and CRYSTALS-Kyber, we performed 10 secret key recovery attacks on the implementation programmed into the device  $D_A$  for 10 different secret keys selected at random. For Saber, we used CCTs based on ECCs with code distances 4 and 6. For CRYSTALS-Kyber, we used ECCs with code distances 2, 4, 6 and 8. So, in total, we carried out 60 secret key recovery attacks.

TABLE VII: The maximum number of traces required for successful key recovery in all 10 attacks.

Algorithm	Code distance		
	8	6	4
Saber	N/A	4608	9216
CRYSTALS-Kyber	48384	38016	59136

The results are summarized in Tables V and VI. The most important part is the column 3. If there are undetected incorrect key coefficients, the attack fails. We count incorrect coefficients by comparing the recovered key to the true key.

Next in importance is the column 4. If the number of detected incorrect key coefficients is small, the coefficients can be recovered by enumerating over all their possible values. With  $i$  detected incorrect coefficients, at most  $9^i$  and  $5^i$  enumerations are required to find the true secret key for Saber and CRYSTALS-Kyber, respectively. If the number of detected incorrect coefficients is large, one can instead consider the LWE problem given by these coefficients only. Such an LWE problem has a smaller dimension than the original problem, potentially allowing typical lattice-based attacks against LWE [34] to succeed in recovering the remaining coefficients.

The last three columns show the time required for capturing traces for CCTs, recovering the corresponding messages, and enumerating the detected incorrect key coefficient (in the worst-case) on a PC with a 6 core processor running at 2.2 GHz and 32 GB of RAM (simple single threaded implementation). The sign “-” means that enumeration is not feasible. Note that only the CCT trace capture requires a physical access to the device under attack  $D_A$ . Other computations are done off-line, so their time is not as crucial.

Finally, in Table VII we compare the number of traces required for secret key recovery using different ECCs. For both Saber and CRYSTALS-Kyber, the ECC with the code distance 6 seems to be the best choice. We believe that two main reasons for the higher number of attack traces required for CRYSTALS-Kyber are:

- Non-uniform leakage of masked message bits in CRYSTALS-Kyber implementation [21].
- Three times lower sampling rate which we use to acquire CRYSTALS-Kyber traces.

## VIII. CONCLUSION

We demonstrated secret key recovery attacks on first-order masked and shuffled implementations of Saber and CRYSTALS-Kyber by deep learning-based power analysis.

The new message recovery method might have significance beyond the scope of the presented work. The idea of classifying all-0 and all-1 binary tuples only, instead of all, and rotating the message to compensate, may be useful in side-channel attacks on other software and hardware implementations of LWE/LWR PKE/KEMs.

We are currently working on developing deep learning-resistant countermeasures for LWE/LWR PKE/KEM implementations.

## IX. ACKNOWLEDGMENTS

This work was supported in part by the Swedish Civil Contingencies Agency (Grant No. 2020-11632) and the Swedish Research Council (Grant No. 2018-04482).

## REFERENCES

- [1] P. Schwabe *et al.*, “CRYSTALS-Kyber algorithm specifications and supporting documentation,” 2020, <https://csrc.nist.gov/projects/postquantum-cryptography/round-3-submissions>.
- [2] D. Moody, “Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process,” *Nistir 8309*, pp. 1–27, 2022, <https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf>.
- [3] C. Chen *et al.*, “NTRU algorithm specifications and supporting documentation,” 2020, <https://csrc.nist.gov/projects/postquantum-cryptography/round-3-submissions>.
- [4] J. D’Anvers *et al.*, “Saber algorithm specifications and supporting documentation,” 2020, <https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf>.
- [5] D. Hofheinz, K. Hövelmanns, and E. Kiltz, “A modular analysis of the Fujisaki-Okamoto transformation,” in *Theory of Cryptography*, ser. LNCS. Cham: Springer International Publishing, 2017, pp. 341–371.
- [6] P. C. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems,” in *Advances in Cryptology - CRYPTO ’96*, N. Koblitz, Ed. Springer Berlin Heidelberg, 1996, pp. 104–113.
- [7] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology - CRYPTO’ 99*. Springer, 1999, pp. 388–397.
- [8] D. Agrawal, B. Arachambeault, J. R. Rao, and P. Rohatgi, “The EM side-channel(s),” in *Crypt. Hard. and Embedded Systems*, 2003, pp. 29–45.
- [9] S. Chari *et al.*, “Towards sound approaches to counteract power-analysis attacks,” in *Advances in Cryptology - CRYPTO ’99*, vol. 1666. Springer, 1999, pp. 398–412.
- [10] Veyrat-Charvillon *et al.*, “Shuffling against side-channel attacks: A comprehensive study with cautionary note,” in *Advances in Cryptology – ASIACRYPT 2012*. Springer Berlin Heidelberg, 2012, pp. 740–757.
- [11] J.-S. Coron and I. Kizhvatov, “An efficient method for random delay generation in embedded software,” in *Crypt. Hardware and Embedded Systems*. Springer Berlin Heidelberg, 2009, pp. 156–170.
- [12] K. Ngo, E. Dubrova, and T. Johansson, “Breaking masked and shuffled CCA secure saber KEM by power analysis,” in *Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security*. ACM, 2021, p. 51–61.
- [13] P. Ravi *et al.*, “On exploiting message leakage in (few) NIST PQC candidates for practical message recovery and key recovery attacks,” *Crypt. ePrint Arch.*, 2020/1559, 2020, <https://eprint.iacr.org/2020/1559>.
- [14] 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*, vol. 2021, no. 4, pp. 676–707, aug 2021.
- [15] C. Yajing *et al.*, “Template attack of LWE/LWR-based schemes with cyclic message rotation,” *Entropy*, vol. 24, no. 10, 2022.
- [16] “Announcing the commercial national security algorithm suite 2.0,” *National Security Agency, U.S Department of Defense*, Sep 2022, [https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA\\_CNSA\\_2.0\\_ALGORITHMS\\_.PDF](https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF).
- [17] M. V. Beirendonck *et al.*, “A side-channel resistant implementation of SABER,” *Cryptology ePrint Archive*, Report 2020/733, 2020, <https://eprint.iacr.org/2020/733>.
- [18] S. Kundu *et al.*, “Higher-order masked Saber,” *Cryptology ePrint Archive*, Report 2022/389, 2022, <https://eprint.iacr.org/2022/389>.
- [19] J. W. Bos *et al.*, “Masking Kyber: First-and higher-order implementations,” *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 173–214, 2021.
- [20] J.-P. D’Anvers *et al.*, “Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations,” *Crypt. ePrint Archive*, 2022/110, 2022, <https://eprint.iacr.org/2022/110>.
- [21] D. Heinz *et al.*, “First-order masked Kyber on ARM Cortex-M4,” *Cryptology ePrint Archive*, Report 2022/058, 2022, <https://eprint.iacr.org/2022/058>.
- [22] M. Azouaoui *et al.*, “Post-quantum authenticated encryption against chosen-ciphertext side-channel attacks,” *Cryptology ePrint Archive*, Report 2022/916, 2022, <https://eprint.iacr.org/2022/916>.
- [23] T.-T. Tsai *et al.*, “Leakage-resilient certificate-based authenticated key exchange protocol,” *IEEE Open Journal of the Computer Society*, vol. 3, pp. 137–148, 2022.
- [24] C. Hoffmann *et al.*, “Towards leakage-resistant post-quantum CCA-secure public key encryption,” *Cryptology ePrint Archive*, Report 2022/873, 2022, <https://eprint.iacr.org/2022/873>.
- [25] J. Wang *et al.*, “Practical side-channel attack on masked message encoding in latticed-based KEM,” *Cryptology ePrint Archive*, Report 2022/859, 2022, <https://eprint.iacr.org/2022/859>.
- [26] M. Shen *et al.*, “Find the bad apples: An efficient method for perfect key recovery under imperfect SCA oracles – a case study of Kyber,” *Crypt. ePrint Archive*, Report 2022/563, 2022, <https://eprint.iacr.org/2022/563>.

- [27] N. Paulsrud, “A side channel attack on a higher-order masked software implementation of saber,” Master’s thesis, School of Electrical Engineering and Computer Science, KTH, 2022.
- [28] B.-Y. Sim *et al.*, “Single-trace attacks on the message encoding of lattice-based kems,” Cryptology ePrint Archive, Report 2020/992, 2020, <https://eprint.iacr.org/2020/992>.
- [29] K. Ngo, R. Wang, E. Dubrova, and N. Paulsrud, “Side-channel attacks on lattice-based KEMs are not prevented by higher-order masking,” Crypt. ePrint Archive, Report 2022/919, 2022, <https://eprint.iacr.org/2022/919>.
- [30] S. Bhasin *et al.*, “Attacking and defending masked polynomial comparison for lattice-based cryptography,” Cryptology ePrint Archive, Report 2021/104, 2021, <https://eprint.iacr.org/2021/104>.
- [31] R. Wang, K. Ngo, and E. Dubrova, “A message recovery attack on LWE/LWR-based PKE/KEMs using amplitude-modulated EM emanations,” in *Proc. of 25th Annual Int. Conf. on Information Security and Cryptology*, 2022, <https://eprint.iacr.org/2022/852>.
- [32] B. L. Welch, “The generalization of ‘Student’s’ problem when several different population variances are involved,” *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.
- [33] R. Wang, K. Ngo, and E. Dubrova, “Side-channel analysis of Saber KEM using amplitude-modulated EM emanations,” in *Proc. of 25th Euromicro Conf. on Digital System Design*, 2022, <https://eprint.iacr.org/2022/807>.
- [34] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi, “LWE with side information: Attacks and concrete security estimation,” in *Advances in Cryptology – CRYPTO 2020*, vol. 12171. Springer, 2020, pp. 329–358.