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

Linus Backlund(✉), Kalle Ngo(✉), Joel Gärtner(✉), and 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 attacks. 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 attacks more difficult. Recently, a side-channel attack on a masked and shuffled implementation of Saber requiring 61,680 power traces to extract the long-term secret key was reported. In this paper, we present an attack that can recover the long-term 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 decryption 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 of length l, where k is the module rank, 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.

Keywords: Public-key cryptography · Post-quantum cryptography · CRYSTALS-Kyber · Saber · Side-channel attack · Power analysis

1 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 [25], as a Public-Key Encryption (PKE) and Key Encapsulation Mechanism (KEM)

J. Zhou et al. (Eds.): ACNS 2023 Workshops, LNCS 13907, pp. 159–177, 2023.
https://doi.org/10.1007/978-3-031-41181-6_9

to be standardized [19]. The other two lattice-based finalists, an NTRU-based scheme NTRU [8] and a Learning With Rounding (LWR)-based scheme Saber [11], 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 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 [15].

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 KEM 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 [17], power consumption [16], or electromagnetic (EM) emanations [2]. No physical device free of side-channels has yet been constructed. Instead countermeasures such as masking [7], shuffling [29], random delays insertion [9], etc. are used to combat side-channel leakage.

Masking [7] attempts to eliminate the leakage by partitioning a secret variable into two or more shares and executing all operations separately on them. 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. Boolean masking uses the XOR to combine the shares into the original secret. Arithmetic masking uses the arithmetic addition.

Shuffling [29] 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 sufficient protection against side-channel attacks. However, in [21] an attack defeating the combined protection was shown, which uses the bit flipping ciphertext malleability of LWE/LWR PKEs discovered in [24]. 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 [24] presents two ways to accomplish this utilizing the fact that ciphertexts of ring-based 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, the recovery of a message encapsulated in a properly generated ciphertext 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 can be 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 [24]. In [20], 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 [21] are:

- *A message recovery method using 0 and 255 FY indexes and rotations.* Rather than extracting the message Hamming Weight (HW) and bit flipping, as in [21], 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 [24,35]. 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 [19]. Given that the National Security Agency (NSA) has included CRYSTALS-Kyber in the suite of cryptographic algorithms recommended for national security systems [1], 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.

2 Previous Work

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

2.1 Implementations

Several implementations of Saber [4,18] and CRYSTALS-Kyber [6,12,13] protected by masking have been presented.

Masking brings a significant execution 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. [4], has an overhead factor of 2.5 on an ARM Cortex-M4 compared to an unmasked one. This implementation employs 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. [4] 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 packed byte array in 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 [3,14,28], were proposed to strengthen side-channel resistance of the NIST PQC candidates.

To the best of our knowledge, the implementation of Saber presented in [21] is the only available masked and shuffled implementation of a module-LWE/LWR PKE/KEM scheme.

2.2 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 [24,26,30].

In [24], 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 [20]. 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 [20] how to recover the secret key from 24 power traces captured during the execution of the decryption procedure for CCTs. The ciphertexts are constructed using an ECC-based method which can correct some errors in the recovered messages. In [23] the attack was extended to the implementation of Saber from [18].

An attack applying the method of [20] to a first-order masked implementation of CRYSTALS-Kyber was presented in [30], targeting the message encoding vulnerability found in [27]. In [22], it was demonstrated that the method of [20] 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 XOR them to get the message. In [5], side-channel attacks on two implementations of masked polynomial comparison were demonstrated on CRYSTALS-Kyber.

The closest related work to the presented one is [21] 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 [21], $N = 10$ is used. Similarly to the method of [20], a neural network trained at the profiling stage is used to 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 a ciphertext created using the bit flipping method of [24]. This ciphertext encrypts a message m', in which one bit of m is flipped, and recovering $HW(m')$ allows determining the flipped bit of m by comparing $HW(m)$ and $HW(m')$.

3 Saber and CRYSTALS-Kyber Algorithms

In this section, we briefly describe CRYSTALS-Kyber and Saber algorithms. For more details, the reader is referred to [25] and [11].

Figure 1 shows pseudocode of the CPA-PKE algorithms, upon which the respective KEMs are built. The pseudocode is applicable to both CRYSTALS-Kyber and Saber [31]. CPA-PKE contains three algorithms: key genera-

CPA-PKE.KeyGen()
1: $seed_{\mathbf{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k\times k}; seed_{\mathbf{A}})$
3: $\mathbf{s} \leftarrow \chi_1(R_q^{k\times 1})$
4: $\mathbf{e} \leftarrow \chi_2(R_q^{k\times 1})$
5: $\mathbf{b} = \lfloor \mathbf{As} + \mathbf{e} \rceil_{p_1}$
6: $pk = (seed_{\mathbf{A}}, \mathbf{b}), sk = \mathbf{s}$
7: **return** (pk, sk)

CPA-PKE.Dec($\mathbf{s}, c = (\mathbf{u}, v)$)
1: $x = \lfloor v \cdot q/p_3 \rceil - \mathbf{s}\, \lfloor \mathbf{u} \cdot q/p_2 \rceil$
2: $m' = \mathsf{decode}(x)$
3: **return** m'

CPA-PKE.Enc($pk = (seed_{\mathbf{A}}, \mathbf{b}), m, r$)
1: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k\times k}; seed_{\mathbf{A}})$
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: $e'' \leftarrow \chi_3(R_q^{1\times 1}; r)$
5: $\mathbf{u} = \lfloor \mathbf{s}'\mathbf{A} + \mathbf{e}' \rceil_{p_2}$
6: $\mathbf{v}' = \mathbf{s}'\, \lfloor \mathbf{b} \cdot q/p_1 \rceil + e''$
7: $v = \lfloor \mathbf{v}' + \mathsf{encode}(m) \rceil_{p_3}$
8: **return** $c = (\mathbf{u}, v)$

Fig. 1. Pseudocode of CPA-PKE algorithms.

tion, CPA-PKE.KeyGen; encryption, CPA-PKE.Enc; and decryption, CPA-PKE.Dec.

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 χ_i over a set S is denoted by $v \leftarrow \chi_i(S)$ while $v \leftarrow \chi_i(S; r)$ denotes deterministic sampling from χ_i using seed r. The uniform distribution is denoted by $\mathcal{U}$ and $\lfloor v \rceil_p$ stands for $\lfloor v \rceil_p = \lfloor v \cdot (p/q) \rceil$, where $\lfloor x \rceil$ means rounding of x to the closest integer with ties being rounded up.

The encode function in CPA-PKE encodes a message to a polynomial by letting each coefficient of the polynomial be equal to the corresponding bit of the message times $\lfloor p_3/2 \rceil$. 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 $\lfloor q/2 \rceil$ than to 0, the message bit is 1. Otherwise, the message bit is 0.

The PKEs of both CRYSTALS-Kyber and Saber can be seen as different parametrizations of CPA-PKE, with the security levels of the schemes mainly differ by the rank k of the module that they use. 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 χ_i that the schemes use and in rounding parameters p_1, p_2, p_3. In CRYSTALS-Kyber, all distributions χ_i are centered binomial distributions with $p_1 = 1$, $p_2 > 1$ and $p_3 > 1$. In Saber, only χ_1 is a centered binomial distribution while samples from the other distributions χ_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.

4 Attack Scenario

We assume a scenario in which the attacker has physical access to the target device to acquire side-channel information and has the ability to query the device

with chosen ciphertexts. In addition, we assume that the keys (pk, sk) are static and that the attacker has a fully controllable profiling device similar to the device under attack.

```
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
```

```
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
```

Fig. 2. Modified C code of the masked_poly_tomsg() procedure of masked CRYSTALS-Kyber with shuffling added.

5 Experimental Setup

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

5.1 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. To verify that the negative effect of intra-device/board variations, which would be expected in a real attack scenario, is not preventing the presented attack, we selected D_P and D_A as a pair of boards acquired from different chip vendors with different ages and wear-out.

Similar equipment is used in the attack on Saber from [21] except that in [21] ChipWhisperer-Lite is used instead of ChipWhisperer-Pro. ChipWhisperer-Pro has a larger buffer size of 98K samples compared to that of ChipWhisperer-Lite, which is 24K samples.

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

[1] ChipWhisperer-Pro has an option of streaming, however, streaming can be used only at a maximum of 10 MHz sampling frequency.

5.2 Target Implementations

To the best of our knowledge, there are no publicly available implementations of CRYSTALS-Kyber protected by both masking and shuffling. The experiments presented in this paper are performed using the C implementation which we built on top of the first-order masked implementations of CRYSTALS-Kyber from [13].

The attack point of the presented side-channel attack is the procedure masked_poly_tomsg(). It is called by CPA-PKE.Dec() during the decode operation at line 2 in Fig. 1. The modified code of masked_poly_tomsg(), with shuffling added, is shown in Fig. 2. The lines marked in blue and red indicate vulnerabilities exploited in the presented attack. The blue lines of FY_Gen() are where the index is being loaded and stored in its randomized position during generation. The blue line in masked_poly_tomsg() shows where the random index is loaded for use in the inner loop. 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 [21] 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).

Table 1. The MLP layer widths.

Width	Saber		CRYSTALS-Kyber	
	N_Y	N_M	N_Y	N_M
Input	215	35	820	225
Layer 1	512	256	1024	512
Layer 2	256	128	512	128
Layer 3	128	64	256	64
Output	256	2	256	2

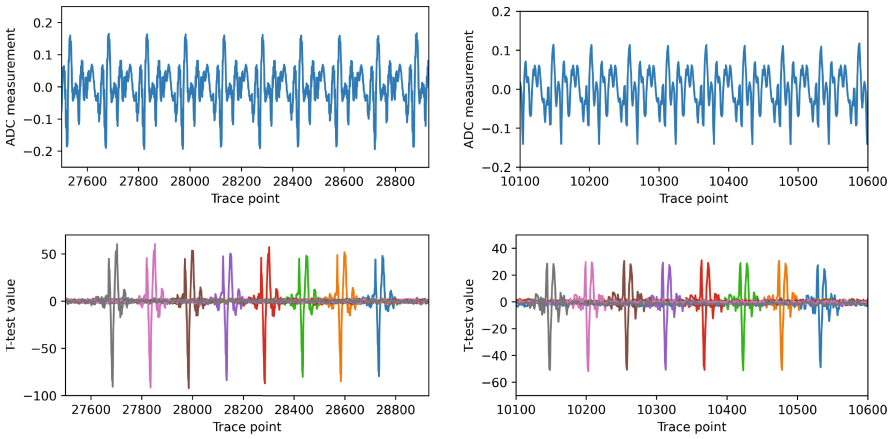
6 Profiling Stage

Our profiling strategy is similar to [21] 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 [21], but with different layer widths (see Table 1). For CRYSTALS-Kyber two separate models N_m are trained for the first and the last bits of a byte (because their leakages differ).

We train on 15K traces for Saber and 50K traces for CRYSTALS-Kyber, captured from the profiling device D_P. As in [21], we cut-and-join traces across

bits to increase the training set without having to capture more traces. Unlike in [21], we do not use traces captured from the device under attack D_A for profiling. In addition, we use standardization of the traces. The implementation of the index generation function FY_Gen() which we use is not constant-time due to entropy sourcing in the random number generator. Thus, the traces have 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 the neural networks is a concatenation of two intervals covering the FY index generation and the FY index usage in the inner loop. Each of those intervals covers the target index and both of its neighbors. For message bit recovery, the input trace to neural networks is an interval covering the processing of both shares for the targeted message bit.



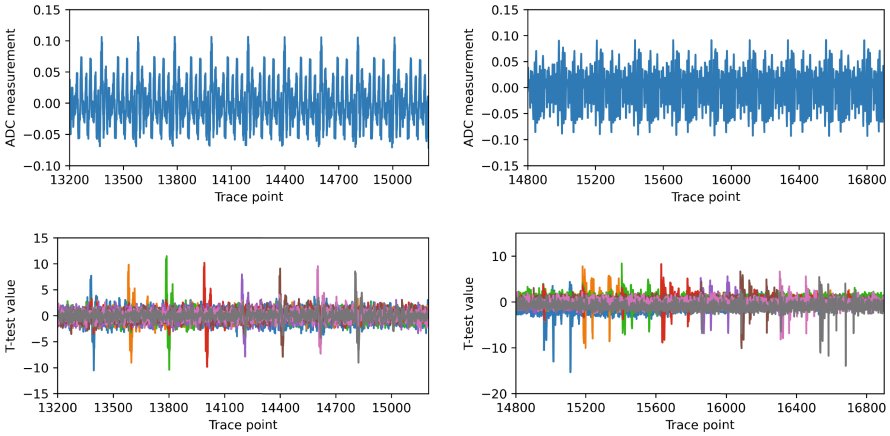
(a) Saber - Index generation (b) CRYSTALS-Kyber - Index generation

Fig. 3. An average trace representing the generation of the FY indexes 64–72 (top) and its t-test (bottom) for Saber (a) and CRYSTALS-Kyber (b), respectively. Both t-tests are performed on 5K traces with known FY indexes.

To find where the FY indexes are generated, we apply Welch's t-test [33] to a set of 5K traces $\boldsymbol{T}$ with known FY indexes captured from D_P during the execution of CPA-PKE.Dec(). For each $i \in \{0, 1, \ldots, 255\}$, we partition $\boldsymbol{T}$ into two subsets such as:

$$\boldsymbol{T}_0 = \{T_j \in \boldsymbol{T} \mid HW(FY_j[i]) < 4\},$$
$$\boldsymbol{T}_1 = \{T_j \in \boldsymbol{T} \mid HW(FY_j[i]) > 4\},$$

where $HW(FY_j[i])$ is the HW of the FY index of ith processed bit of message m_j in trace T_j, for all $j \in \{0, 1, \ldots, |\boldsymbol{T}|\}$. The Welch's t-test determines if there is a noticeable difference in the means of $\boldsymbol{T}_0$ and $\boldsymbol{T}_1$.



(a) Saber - Index usage (b) CRYSTALS-Kyber - Index usage

Fig. 4. An average trace representing the usage of FY indexes during message processing (top) and its t-test (bottom) for Saber (a) and CRYSTALS-Kyber (b), respectively. Both t-tests are performed on 5K traces with known FY indexes.

Figure 3 show the t-test results for the index generation part of Saber (a) and CRYSTALS-Kyber (b), 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 more samples taken per clock cycle, the probability to catch a 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. 4 for Saber (a) and CRYSTALS-Kyber (b), 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 maximize the prediction accuracy of the neural networks.

The message bits are recovered from the same trace segments where FY indexes are used. 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 packed 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. The order in which the bits are set is randomized for each execution due to shuffling. This makes incremental recovery with regard to previously set bits unfeasible.

For CRYSTALS-Kyber, we train N_{FY} and N_m models on standardized 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 [32] and retrain to improve the accuracy. We remove all points whose assignment to 0 decreases 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 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, considering only the votes by models N_m with prediction confidence higher than 0.9.

Due to the 3-stage pipelining of ARM Cortex M4, the first and last processed indexes and message bits look different. Therefore, additional model ensembles are trained specifically for each of them.

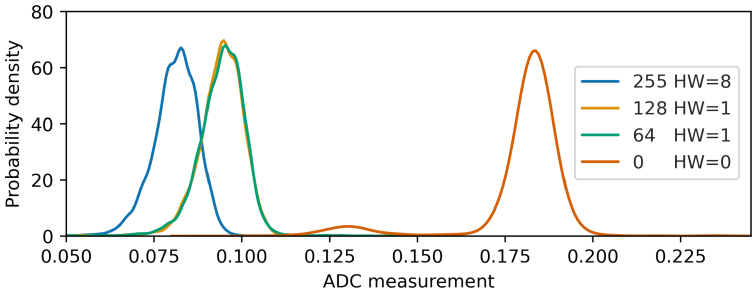


Fig. 5. Distribution of power consumption during the generation of FY indexes 0, 64, 128 and 255 at the trace point with the maximum absolute t-test value in Saber. Index 0 to the left, 64 and 128 in the middle, and 255 to the right.

7 Attack Stage

7.1 Message Recovery Using 0 and 255 FY Indexes and Rotation

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. 5. Bytes that have the same HW are almost completely overlapping. Only the "corner" indexes 0 and 255, whose HW is unique, 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 decryption of c, recover FY indexes 0 and 255 using neural networks for index recovery, N_{FY}, and extract the corresponding two bits of m using neural networks for message recovery, N_m. Then, we modify c to c' which encrypts m rotated by two bit positions, capture a power trace during the decryption of c', and extract the next two message bits with FY indexes 0 and 255. This way, by rotating m by two bit positions 128 times, all message bits are extracted.

7.2 Cyclic Rotation of CCTs

It is known [24,35] 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 $\mathsf{decode}(-x)$ and $\mathsf{decode}(x)$ can be different, where x is defined in line 1 of CPA-PKE.Dec() in Fig. 1. The function $\mathsf{decode}(x)$ 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 is decoded to 0 because its value is closer to 0, while $-x = q - x$ is decoded to 1 as if its value 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 $\mathbf{s}$ 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 $\mathbf{s}[i]$, this is sufficient for the full secret key recovery. Care should be taken to keep track of whether $\mathbf{s}[i]$ has wrapped around modulo n or not. In the former case, the message bit is determined by $-\mathbf{s}[i]$ and not $\mathbf{s}[i]$.

7.3 Incremental CCT Construction Method

In the ECC-based CCT construction method of [20], the secret key coefficients are mapped into the codewords of an $[8, 4, 4]$ extended Hamming code[2]. 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

[2] 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.

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

Table 2. 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
0	0	6	5	(240,10)	1	1	0	0	1	1	0	0	1
1	4			(377,10)	0	0	1	0	1	1	0	1	0
2	6	0	3	(613,4)	1	0	1	0	1	1	0	1	0
3	1	2	0	(373,15)	1	0	1	1	0	1	0	0	1
4	9	5	2	(913,15)	1	1	1	0	0	0	0	1	1
5	10	7	6	(12,3)	1	0	0	0	0	0	0	0	0
6	2	3	1	(793,10)	1	0	0	1	1	0	0	1	1
7				(755,4)	0	1	0	0	1	1	0	0	1
8	8			(917,10)	0	1	1	1	1	0	0	0	0
9	5			(806,10)	0	0	0	1	1	0	0	1	1
10	7	4		(456,15)	1	1	0	1	0	1	0	1	0
11	3	1	4	(68,4)	1	1	1	1	1	0	0	0	0

The presented method uses $3l$ ciphertexts to recover the secret key coefficients $\mathbf{s}[256r+i]$, for all $i \in \{0, 1, \ldots, 255\}$ and $r \in \{0, 1, 2\}$. The coefficient $\mathbf{s}[256r+i]$ is derived from the codeword $(m_{rl}[i], \ldots, m_{(r+1)l-1}[i])$ of an $[l, w, d]$ linear code using the mapping defined by Tables 2 and 3 for Saber and CRYSTALS-Kyber, respectively.

For all $j \in \{0, 1, \ldots, l-1\}$, 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 2 and 3.

Table 3. 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
0	0	1	8	5	0		(1,153,8)	0	1	1	1	0
1	9	7	0	6	4	1	(0,77,5)	1	1	1	0	0
2	4	0	5	4	1	0	(2,335,15)	1	1	0	1	1
3	6	10	1	1	5		(3,432,3)	0	1	0	0	1
4	1	2	2	3	2	3	(3,606,3)	1	0	0	1	0
5	7	5	9				(1,864,8)	0	1	1	1	0
6	2	3	4	2			(0,915,5)	0	0	1	1	1
7	3	4					(0,898,5)	0	0	1	1	1
8	10	8	3	0	3	2	(3,432,8)	1	0	1	0	1
9	11						(0,105,5)	1	1	1	0	0
10	8	6	7				(3,632,3)	1	0	0	1	0
11							(3,386,3)	0	1	0	0	1
12	12						(3,606,8)	1	0	1	0	1
13	5	9	6				(2,321,15)	1	1	0	1	1

The non-empty entries in the first multi-column of Tables 2 and 3 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 3, for $d = 2$, the codeword length is 4, so there are 4 non-empty entries, $j \in \{1, 0, 3, 2\}$, in the column.

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 $c_{4r}, c_{4r+1}, c_{4r+2}, c_{4r+3}$, 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 that 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 key coefficient is 2.

As in [20], we select the constants k_1 and k_0 so that each secret key coefficient is uniquely mapped into some codeword composed from the l decrypted message bits. For CRYSTALS-Kyber, in which the leakage of different bits within a byte is non-uniform and affected by previously set bits, we also use one more constant, k_2, which allows us to minimize the HW of messages. This helps 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 [13], the decoding does not perfectly match the decoding in the unpro-

tected reference implementation [25]. To handle this, we select the constants (k_2, k_1, k_0) for CRYSTALS-Kyber so that $\mathsf{decode}(x \pm \varepsilon) = \mathsf{decode}(x)$ for small ε, results in equivalent ciphertexts in both implementations.

8 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 4. 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)		Success	Attack time (the worst case)		
		Undetected	Detected	rate	Capture	Message rec.	Enum.
30	8	0	0	100%	19.5 h	20.1 h	0 s
	6	0	0.2	100%	15.3 h	15.8 h	0.01 s
	4	0	2.1	100%	9.8 h	10.0 h	0.08 s
	2	17.3	0.9	0%	5.6 h	5.8 h	0.02 s
20	8	0	0	100%	13.0 h	13.4 h	0 s
	6	0	0.4	100%	10.2 h	10.5 h	0.01 s
	4	0.1	5.4	90%	6.5 h	6.7 h	15 s
	2	29.1	3.0	0%	3.7 h	3.9 h	0.4 s
10	8	0	1.8	100%	6.5 h	6.7 h	0.05 s
	6	0.1	9.1	90%	5.1 h	5.3 h	96 min
	4	0.4	27.8	0%	3.3 h	3.4 h	–
	2	73.1	15.8	0%	1.9 h	2.0 h	–

The results are summarized in Tables 4 and 5. The most important part is 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, excluding the detected incorrect coefficient.

Next in importance is column 4; if the number of detected incorrect key coefficients is small, the coefficients can be recovered by enumerating 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

Table 5. 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)		Success	Attack time (the worst case)		
		Undetected	Detected	rate	Capture	Message rec.	Enum.
3	6	0	0	100%	4.0 h	26.0 min	0 s
	4	0	0.3	100%	2.7 h	17.3 min	0.01 s
2	6	0	1.7	100%	3.3 h	17.3 min	10.5 s
	4	0.4	5.7	80%	2.2 h	11.6 min	12 min
1	6	0	4.9	100%	3.0 h	8.7 min	2 min
	4	0.7	13.2	0%	2.0 h	5.8 min	–

allowing typical lattice-based attacks against LWE [10] to succeed in recovering the remaining coefficients.

Column 5 states the success rate, i.e. the percentage of experiments in which the secret key was successfully extracted.

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

Table 6. Number of traces required for successful key recovery in all 10 attacks.

Algorithm	Code distance			
	8	6	4	2
Saber		4608	9216	
CRYSTALS-Kyber	48384	38016	59136	–

Finally, in Table 6 we compare the number of traces required for secret key recovery using different ECCs. For both Saber and CRYSTALS-Kyber, the ECC with code distance 6 seems 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 [13].
- The traces were sampled at a third of the rate used to acquire Saber traces.

9 Countermeasures

The following techniques could make the presented attack more difficult:

1. Protecting the shuffling index generation procedure using masking or other countermeasures.
2. Bitslicing the implementations.

The following techniques would make the presented attack impossible:

1. Prevent decryption of sparse or low-entropy ciphertexts by introducing a check for minimal entropy as suggested by [34].
2. Prevent decryption of chosen ciphertexts by using e.g. Encrypt-then-Sign method suggested by [3].
3. Re-generating the keys (pk, sk) for each new shared key establishment.

10 Conclusion

We demonstrated a secret key recovery attack on first-order masked and shuffled implementations of Saber and CRYSTALS-Kyber by deep learning power SCA.

Note that the ChipWhisperer platform is essentially noise free. Therefore, the conditions in the experiments can be considered the best-case. Yet, the presented attack shows a 13-fold improvement over previous work (also performed on the ChipWhisperer platform), requires no profiling traces from the target device, and is more robust to inter-device/board variations.

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 possible) and rotating the message, may be useful in side-channel attacks on other software and hardware implementations of ring-based LWE/LWR PKE/KEMs.

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

Our code is available at https://github.com/lbacklund/SCA-Masked-Shuffled-Saber-Kyber.

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. Announcing the commercial national security algorithm suite 2.0. National Security Agency, U.S Department of Defense (2022). https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF
2. Agrawal, Dakshi, Archambeault, Bruce, Rao, Josyula R.., Rohatgi, Pankaj: The EM side—channel(s). In: Kaliski, Burton S.., Koç, çetin K.., Paar, Christof (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_4

3. Azouaoui, M., et al.: Post-quantum authenticated encryption against chosen-ciphertext side-channel attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. 372–396 (2022). https://doi.org/10.46586/tches.v2022.i4.372-396
4. Beirendonck, M.V., et al.: A side-channel-resistant implementation of saber. J. Emerg. Technol. Comput. Syst. **17**(2) (2021). https://doi.org/10.1145/3429983
5. Bhasin, S., et al.: Attacking and defending masked polynomial comparison for lattice-based cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. 334–359 (2021). https://doi.org/10.46586/tches.v2021.i3.334-359
6. Bos, J.W., et al.: Masking kyber: first-and higher-order implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst **2021**(4), 173–214 (2021). https://doi.org/10.46586/tches.v2021.i4.173-214
7. Chari, Suresh, Jutla, Charanjit S.., Rao, Josyula R.., Rohatgi, Pankaj: Towards sound approaches to counteract power-analysis attacks. In: Wiener, Michael (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
8. Chen, C., et al.: NTRU algorithm specifications and supporting documentation (2020). https://csrc.nist.gov/projects/postquantum-cryptography/round-3-submissions
9. Coron, Jean-Sébastien., Kizhvatov, Ilya: An efficient method for random delay generation in embedded software. In: Clavier, Christophe, Gaj, Kris (eds.) CHES 2009. LNCS, vol. 5747, pp. 156–170. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_12
10. Dachman-Soled, Dana, Ducas, Léo., Gong, Huijing, Rossi, Mélissa.: LWE with side information: attacks and concrete security estimation. In: Micciancio, Daniele, Ristenpart, Thomas (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 329–358. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_12
11. D'Anvers, J., et al.: Saber algorithm specifications and supporting documentation (2020). https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf
12. D'Anvers, J.P., et al.: Revisiting higher-order masked comparison for lattice-based cryptography: algorithms and bit-sliced implementations. Cryptology ePrint Archive, 2022/110 (2022). https://eprint.iacr.org/2022/110
13. Heinz, D., et al.: First-order masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Report 2022/058 (2022). https://eprint.iacr.org/2022/058
14. Hoffmann, C., 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
15. Hofheinz, Dennis, Hövelmanns, Kathrin, Kiltz, Eike: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Yael, Reyzin, Leonid (eds.) TCC 2017. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12
16. Kocher, Paul, Jaffe, Joshua, Jun, Benjamin: Differential power analysis. In: Wiener, Michael (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
17. Kocher, Paul C..: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, Neal (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
18. Kundu, S., et al.: Higher-order masked Saber. Cryptology ePrint Archive, Report 2022/389 (2022). https://eprint.iacr.org/2022/389

19. Moody, D.: 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
20. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked IND-CCA secure saber KEM implementation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 676–707 (2021). https://doi.org/10.46586/tches.v2021.i4.676-707
21. Ngo, K., Dubrova, E., Johansson, T.: 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. ACM (2021)
22. Ngo, K., Wang, R., Dubrova, E., Paulsrud, N.: Side-channel attacks on lattice-based KEMs are not prevented by higher-order masking. Cryptology ePrint Archive, Report 2022/919 (2022). https://eprint.iacr.org/2022/919
23. Paulsrud, N.: A side channel attack on a higher-order masked software implementation of saber. Master's thesis, KTH (2022)
24. Ravi, P., 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
25. Schwabe, P., et al.: CRYSTALS-Kyber algorithm specifications and supporting documentation (2020). https://csrc.nist.gov/projects/postquantum-cryptography/round-3-submissions
26. Shen, M., et al.: Find the bad apples: an efficient method for perfect key recovery under imperfect SCA oracles - a case study of Kyber. Cryptology ePrint Archive, Report 2022/563 (2022). https://eprint.iacr.org/2022/563
27. Sim, B.Y., 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
28. Tsai, T.T., et al.: Leakage-resilient certificate-based authenticated key exchange protocol. IEEE Open J. Comput. Soc. **3**, 137–148 (2022). https://doi.org/10.1109/OJCS.2022.3198073
29. Veyrat-Charvillon, Nicolas, Medwed, Marcel, Kerckhof, Stéphanie., Standaert, François-Xavier.: Shuffling against side-channel attacks: a comprehensive study with cautionary note. In: Wang, Xiaoyun, Sako, Kazue (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 740–757. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_44
30. Wang, J., 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
31. Wang, R., Ngo, K., Dubrova, E.: A message recovery attack on LWE/LWR-based PKE/KEMs using amplitude-modulated EM emanations. In: International Conference on Information Security and Cryptology (2022). https://eprint.iacr.org/2022/852
32. Wang, R., Ngo, K., Dubrova, E.: Side-channel analysis of Saber KEM using amplitude-modulated EM emanations. In: Proceedings of the 25th Euromicro Conference on Digital System Design (2022). https://eprint.iacr.org/2022/807
33. Welch, B.L.: The generalization of 'Student's' problem when several different population variances are involved. Biometrika **34**(1/2), 28–35 (1947)
34. Xu, Z., et al.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: the case study of Kyber. Cryptology ePrint Archive, Paper 2020/912 (2020). https://doi.org/10.1109/TC.2021.3122997
35. Yajing, C., et al.: Template attack of LWE/LWR-based schemes with cyclic message rotation. Entropy **24**(10), 15 (2022). https://doi.org/10.3390/e24101489