A Single-Trace Message Recovery Attack on a Masked and Shuffled Implementation of CRYSTALS-Kyber

Sönke Jendral, Kalle Ngo, Ruize Wang and Elena Dubrova

KTH Royal Institute of Technology, Stockholm, Sweden
{jendral,kngo,ruize,dubrova}@kth.se

Abstract. Last year CRYSTALS-Kyber was chosen by NIST as a new, post-quantum secure key encapsulation mechanism to be standardized. This makes it important to assess the resistance of CRYSTALS-Kyber implementations to physical attacks. Pure side-channel attacks on post-quantum cryptographic algorithms have already been well-explored. In this paper, we present an attack on a masked and shuffled software implementation of CRYSTALS-Kyber that combines fault injection with side-channel analysis. First, a voltage fault injection is performed to bypass the shuffling. We found settings that consistently glitch the desired instructions without crashing the device. After the successful fault injection, a deep learning-assisted profiled power analysis based on the Hamming weight leakage model is used to recover the message (shared key). We propose a partial key enumeration method that allows us to significantly increase the success rate of message recovery (from 0.122 without enumeration to 0.887 with 32 enumerated bits).

Keywords: Fault injection · Side-channel attack · CRYSTALS-Kyber · ML-KEM · Post-quantum cryptography

1 Introduction

CRYSTALS-Kyber is a key encapsulation mechanism (KEM) that is IND-CCA2-secure in the classical and quantum random oracle models [3]. IND-CCA2 is the strongest of the definitions of security, meaning that the algorithm is indistinguishable under an adaptive chosen-ciphertext attack (CCA).

Last year CRYSTALS-Kyber was chosen by the National Institute of Standards and Technology (NIST) as a new public key encryption (PKE) and key establishment algorithm to be standardized, ML-KEM [39]. The National Security Agency (NSA) has added CRYSTALS-Kyber to the suite of cryptographic algorithms that are recommended for national security systems [1]. Thus, it is important to assess the resistance of CRYSTALS-Kyber's implementations to physical attacks such as side-channel analysis and fault injection.

Susceptibility of post-quantum cryptographic (PQC) algorithms' implementations to physical attacks is one of the focuses of the ongoing fourth round of the NIST PQC standardization process [27]. In today's digitalized and networked world, devices executing cryptographic algorithms can often be physically accessed by an attacker, who might be the user. Examples of such devices include smart home appliances, building access control and alarm systems, medical devices and wearables, car multimedia interfaces, and more. These devices often lack security features of IT systems due to resource constraints. This opens the door to physical attacks. These attacks can be carried out in the context of government/industry espionage, or motivated by personal interests, e.g. monetary advantages.

Non-invasive physical attacks such as side-channel analysis and fault injection are effective and low-cost. They do not require expensive equipment and expert-level skills like invasive attacks. Many different types of side-channel and fault attacks on block ciphers, stream ciphers, and public key ciphers have been demonstrated in the past [6,5,32]. Moreover, with advances in machine learning, side-channel attacks got a powerful ally [29]. Since machine learning techniques are good at finding correlations in raw data, they enable an attacker to bypass many traditional countermeasures, e.g., masking [24], shuffling [35], and randomized clock [22], and break protected implementations. They can significantly improve the effectiveness of traditional differential power analysis (e.g. four instead of 400 power traces are required to extract the secret key from a USIM card [13]) and enable new types of attacks (e.g. on a true random number generator in a commercial integrated circuit [23]). Given huge investments in machine learning, the capability of machine learning-assisted attacks is likely to keep growing in the future. Therefore, it is important to continue exploring their limits.

Contributions: In this paper, we present a combined fault and side-channel attack on a masked and shuffled implementation of CRYSTALS-Kyber KEM. The presented attack requires only a single power trace to recover the encapsulated message (shared key). This is a significant improvement over the previous pure side-channel attack on a masked and shuffled software implementation of CRYSTALS-Kyber KEM which requires about 3K power traces [4][1]. To the best of our knowledge, the presented attack is the first attack on a protected implementation of a PQC KEM which makes use of both fault injection and side-channel analysis. Only pure side-channel or fault attacks on PQC KEM implementations have been demonstrated in the past. A combination of both is clearly more powerful, as shown by previous work on other cryptographic algorithms [2].

We demonstrate a practical message recovery attack on the first-order masked bitsliced software implementation of Kyber-768 by Bronchain et al. [14] with shuffling added. First, a voltage fault injection using the crowbar technique of [28] is performed to bypass the shuffling. Finding settings that consistently glitch the desired instructions without crashing the device is the most challenging part

[1] The number of traces required for message recovery is not reported in [4] since the attack targets secret key recovery. We made a rough estimation by dividing the number of traces used for secret key recovery with the error-correcting code with code distance six (38,016) by the number of chosen ciphertexts for this code (11).

of this step. After the successful fault injection, a deep learning-assisted profiled power analysis based on the Hamming weight leakage model is applied to recover the message. Following [41], the presented attack exploits leakage of the masked message encoding procedure which is called during the re-encryption step of the decapsulation algorithm. The message recovery is performed using the single-step approach of Ngo et al. [24] which extracts the masked secret directly, without extracting each share explicitly. A new contribution is a partial key enumeration method that allows us to increase the probability of full message recovery from 0.122 (no enumeration) to 0.887 for (32 enumerated bits).

In CRYSTALS-Kyber, a successful message recovery trivially implies the shared key recovery, since the shared key is derived from the message using hash functions. Furthermore, by recovering messages contained in a set of chosen ciphertexts constructed using known methods, e.g. [4,38], one can extract the long-term secret key.

The rest of the paper is organised as follows. Section 2 describes previous work. Section 3 provides the background information on the CRYSTALS-Kyber algorithm, masking and shuffling countermeasures, and voltage fault injection. Section 4 describes the adversary model, key establishment protocol and attack scenario. Section 5 presents the experimental setup. Section 6 and 7 describe the fault injection and the power analysis parts of the attack, respectively. Section 8 summarises the experimental results. Section 9 discusses possible countermeasures. Section 10 concludes the paper and discusses open problems.

2 Previous Work

This section describes previous side-channel and fault attacks on cryptographic algorithms related to the presented work.

Two pure side-channel attacks on first-order masked and shuffled software implementations of PQC KEMs based on module lattices are presented in [25] and [4]. Both attacks use deep learning-assisted power analysis. The attack in [25] requires 61,680 traces to extract the secret key of Saber KEM. It is based on full message Hamming weight extraction and iterative bit flipping. The attack in [4] can extract the secret key of Saber from 4,608 traces. It employs shuffling index recovery and cyclic rotations.

In [4] a secret key recovery attack on a first-order masked and shuffled software implementation of CRYSTALS-Kyber is also demonstrated. The implementation is built on top of the first-order masked implementation by Heinz et al. [21]. The attack exploits a leakage from the message decoding procedure which is performed at the decryption step of decapsulation. The attack uses 38,016 traces to extract the secret key.

Many pure side-channel attacks on masked implementations of CRYSTALS-Kyber are reported, including [33,20,10,40,38,16,19,41]. The attack presented in [41] uses the same masked implementation of CRYSTALS-Kyber [14] as the one used in this work (except that we also add shuffling to it). By combining three leakage points of the message encoding procedure carried out at the re-encryption step of decapsulation with a novel chosen ciphertext construction method, the secret key can be recovered from two traces of a first-order masked implementation with the probability of 0.98.

A number of pure fault attacks on PQC KEMs are demonstrated, see [34] for a thorough overview of the attacks related to CRYSTALS-Kyber and the Dilithium signature scheme, which is also selected for standardization by NIST. Several countermeasures against side-channel and fault attacks are proposed in [34], for instance the addition of shuffling to the number theoretic transform (NTT) and the KECCAK hash function (SHA-3) to prevent single-trace side-channel attacks.

We are not aware of any combined fault and side-channel attack on a PQC KEM, but other cryptographic algorithms have been targeted by such attacks, including [42,31,30,36]. In [42] a fault-assisted side-channel attack on a masked implementation of Advanced Encryption Standard (AES) is presented. Segments of a trace in which the random masks are generated are identified from a single power trace and then masking is bypassed by fault injection. After a successful fault injection, differential power analysis is applied to recover all subkeys. The attack requires 230 traces on average. In [31,30] combined fault injection and side-channel attacks on implementations of block ciphers AES and PRESENT with countermeasures against side-channel and fault injection attacks are presented. In [36] a combined fault injection and side-channel attack is used to recover the secret recovery from a first-order masked implementation of the GIMLI permutation (the core primitive of a submission to the NIST lightweight cryptography project [9]) with fewer than 10,000 traces.

3 Background

This section describes the notation used in the remainder of this work, the CRYSTALS-Kyber algorithm specification, masking and shuffling countermeasures against side-channel analysis, and the voltage fault injection attack method.

3.1 Notation

Let $\mathbb{Z}_q$ be the ring of integers modulo a prime q. Let R_q be the quotient ring $\mathbb{Z}_q[X]/(X^n + 1)$. Regular font letters denote elements in R_q, bold lower-case letters are used for vectors with coefficients in R_q, and bold upper-case letters for matrices. The transpose of a vector $\boldsymbol{v}$ (or matrix $\boldsymbol{A}$) is denoted by $\boldsymbol{v}^T$ (or $\boldsymbol{A}^T$). The ith entry of a vector $\boldsymbol{v}$ is denoted by $\boldsymbol{v}[i]$.

The term $x \leftarrow \mathcal{D}(S; r)$ stands for sampling x from a probability distribution $\mathcal{D}$ over a set S using a seed r. The uniform distribution is denoted by $\mathcal{U}$. The centered binomial distribution with a parameter μ is denoted by B_μ.

The term $\lceil x \rfloor$ stands for rounding of x to the closest integer with ties being rounded up.

```
KYBER.CPAPKE.KeyGen()
1: $(\rho, \sigma) \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{k\times k}; \rho)$
3: $\boldsymbol{s}, \boldsymbol{e} \leftarrow B_{\eta_1}(R_q^{k\times 1}; \sigma)$
4: $\boldsymbol{t} = \mathsf{Encode}_{12}(\boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$
5: $\boldsymbol{s} = \mathsf{Encode}_{12}(\boldsymbol{s})$
6: return $(pk = (\boldsymbol{t}, \rho), sk = \boldsymbol{s})$

KYBER.CPAPKE.Dec($\boldsymbol{s}, c$)
1: $\boldsymbol{u} = \mathsf{Decompress}_q(\mathsf{Decode}_{d_u}(c_1), d_u)$
2: $v = \mathsf{Decompress}_q(\mathsf{Decode}_{d_v}(c_2), d_v)$
3: $\boldsymbol{s} = \mathsf{Decode}_{12}(\boldsymbol{s})$
4: $m = \mathsf{Encode}_1(\mathsf{Compress}_q(v - \boldsymbol{s}\cdot\boldsymbol{u}, 1))$
5: return $m$

KYBER.CPAPKE.Enc($pk = (\boldsymbol{t}, \rho), m, r$)
1: $\boldsymbol{t} = \mathsf{Decode}_{12}(\boldsymbol{t})$
2: $\boldsymbol{A} \leftarrow \mathcal{U}(R_q^{k\times k}; \rho)$
3: $\boldsymbol{r} \leftarrow B_{\eta_1}(R_q^{k\times 1}; r)$
4: $\boldsymbol{e}_1 \leftarrow B_{\eta_2}(R_q^{k\times 1}; r)$
5: $e_2 \leftarrow B_{\eta_2}(R_q^{1\times 1}; r)$
6: $\boldsymbol{u} = \boldsymbol{A}^T\boldsymbol{r} + \boldsymbol{e}_1$
7: $v = \boldsymbol{t}^T\boldsymbol{r} + e_2 +$
   $\mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$
8: $c_1 = \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\boldsymbol{u}, d_u))$
9: $c_2 = \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(v, d_v))$
10: return $c = (c_1, c_2)$
```

Fig. 1: KYBER.CPAPKE algorithms from [3] (simplified).

```
KYBER.CCAKEM.KeyGen()
1: $z \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $(pk, \boldsymbol{s}) =$
   KYBER.CPAPKE.KeyGen()
3: $sk = (\boldsymbol{s}, pk, \mathcal{H}(pk), z)$
4: return $(pk, sk)$

KYBER.CCAKEM.Encaps($pk$)
1: $m \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $m = \mathcal{H}(m)$
3: $(\hat{K}, r) = \mathcal{G}(m, \mathcal{H}(pk))$
4: $c =$ KYBER.CPAPKE.Enc($pk, m, r$)
5: $K = \mathsf{KDF}(\hat{K}, \mathcal{H}(c))$
6: return $(c, K)$

KYBER.CCAKEM.Decaps($sk$,c)
1: $m' =$ KYBER.CPAPKE.Dec($\boldsymbol{s}, c$)
2: $(\hat{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
3: $c' =$ KYBER.CPAPKE.Enc($pk, m', r'$)
4: if $c = c'$ then
5:     return $K = \mathsf{KDF}(\hat{K}', \mathcal{H}(c))$
6: else
7:     return $K = \mathsf{KDF}(z, \mathcal{H}(c))$
8: end if
```

Fig. 2: KYBER.CCAKEM algorithms from [3] (simplified).

3.2 CRYSTALS-Kyber algorithm

The security of CRYSTALS-Kyber relies on the difficulty of the module learning with errors (M-LWE) problem, which results from adding unknown noise to linear equations.

CRYSTALS-Kyber consists of a chosen plaintext attack (CPA)-secure public key encryption (PKE) scheme, KYBER.CPAPKE, and a CCA-secure KEM, KYBER.CCAKEM, constructed on the top of KYBER.CPAPKE using a tweaked version of the Fujisaki-Okamoto (FO) transform [18]. These schemes are described in Figs. 1 and 2 respectively.

Inputs and outputs to all application programming interface (API) functions of CRYSTALS-Kyber are byte arrays. CRYSTALS-Kyber works with vectors of ring elements in R_q^k, where k is the rank of the module defining the security

Table 1: Parameters of different versions of CRYSTALS-Kyber.

Version	n	k	q	η_1	η_2	(d_u, d_v)
Kyber-512	256	2	3329	3	2	(10, 4)
Kyber-768	256	3	3329	2	2	(10, 4)
Kyber-1024	256	4	3329	2	2	(11, 5)

level. There are three versions of CRYSTALS-Kyber: Kyber-512, Kyber-768 and Kyber-1024, for $k = 2, 3$ and 4, respectively, see Table 1 for details. In this paper, we consider Kyber-768. Other versions can be approached similarly.

CRYSTALS-Kyber employs the NTT to perform multiplications in R_q efficiently. The NTT details are omitted from Fig. 1 and Fig. 2 to simplify the pseudocode.

The Decode_l function decodes an array of $32l$ bytes into a polynomial with n coefficients in the range $\{0, 1, \cdots, 2^l - 1\}$. The Encode_l function is the inverse of Decode_l. It first encodes each polynomial coefficient separately and then concatenates the resulting output byte arrays.

The $\mathsf{Compress}_q(x, d)$ and $\mathsf{Decompress}_q(x, d)$ functions, for $x \in \mathbb{Z}_q$ and $d < \lceil \log_2(q) \rceil$, are given by:

$$\mathsf{Compress}_q(x, d) = \lceil (2^d/q) \cdot x \rfloor \bmod^+ 2^d,$$
$$\mathsf{Decompress}_q(x, d) = \lceil (q/2^d) \cdot x \rfloor.$$

If $\mathsf{Compress}_g$ or $\mathsf{Decompress}_q$ is applied to $x \in \mathbb{R}_q$ or $x \in \mathbb{R}_q^k$, the function is applied to each coefficient individually. These functions enable the removal of some low-order bits in the ciphertext without significantly affecting the correctness probability of decryption.

The functions $\mathcal{G}$ and $\mathcal{H}$ represent the $\mathsf{SHA3\text{-}512}$ and $\mathsf{SHA3\text{-}256}$ hash functions, respectively. The KDF is a key derivation function realized by $\mathsf{SHAKE\text{-}256}$.

3.3 Masking countermeasure

Masking is a well-known countermeasure against side-channel attacks [15]. A k-order masking partitions any sensitive variable x into $k+1$ shares, $x_0, x_1, \ldots, x_k$, such that $x = x_0 \circ x_1 \circ \ldots \circ x_k$, and executes all operations separately on the shares. The operator "$\circ$" depends on the type of masking. In arithmetic masking, it is the arithmetic addition, "$\circ = +$". In Boolean masking, it is the Boolean XOR, "$\circ = \oplus$". The shares are randomized at each execution. The randomization is typically done by assigning random masks $x_0, x_1, \ldots, x_{k-1}$ to k shares and computing the last share as $x - (x_0 + x_1 + \ldots + x_{k-1})$ for arithmetic masking or $x \oplus x_0 \oplus x_1 \oplus \ldots \oplus x_{k-1}$ for Boolean masking.

In theory, executing all operations on the shares $x_0, x_1, \ldots, x_k$ rather than on x, should prevent side-channel attacks targeting x. However, the attack on a first-order masked implementation of Saber KEM presented in [24] has shown that neural networks are capable of identifying trace segments representing the processing of two Boolean shares x_0 and x_1 and then XOR x_0 and x_1 to obtain the ground truth label x. In this way, the sensitive variable x is recovered directly, without explicitly extracting random masks. The attacks on higher-order masked implementation of Saber and CRYSTALS-Kyber KEMs presented in [26,16] have further demonstrated that the complexity of learning the $(k+1)$-argument XOR grows linearly in the number of shares $k+1$.

3.4 Shuffling countermeasure

Shuffling is another well-known countermeasure against side-channel attacks [15]. It is applicable to operations on sensitive variables which are not dependent on the processing order. First, a random permutation is generated, typically by applying the Fisher-Yates algorithm [17, p.26–27]. Then, the sensitive variables are processed based on the permutation.

Since the processing order is randomized, in theory, shuffling should prevent side-channel attacks. Even if the attacker can recover the sensitive variables, their processing order remains unknown. However, deep learning-assisted side-channel attacks presented in [25,4] have shown that there are different ways of overcoming shuffling in practice.

3.5 Voltage fault injection

Voltage fault injection is a low-cost, minimally invasive method for inducing faults in the execution of an algorithm run on a physical device without requiring extensive knowledge of the specific implementation under attack.

Previous voltage fault injection approaches such as [8,7] focus on inducing a fault by uniformly underfeeding a processor, which increases the time for logic gates to reach a stable state, causing faults. The advantage of such an approach is that it does not require precise timing to induce glitch, however, the resulting faults are spatially localised (thus only affecting specific instructions) and do not offer sufficiently precise control over the affected instructions to conduct an attack that relies on the mostly unchanged execution of an algorithm.

More recent voltage fault injection approaches [28,11] focus instead on using a precise timing control and even control over the waveform of the injected voltage fault. This offers a greater degree of control over the affected instructions and is not limited by the type of targeted instructions.

In this paper, we use a crowbar circuit-based technique for voltage fault injection introduced by [28]. By using a MOSFET to short across the power rails of the processor, it is possible to induce oscillations in the target circuit, which can be used to inject faults with high precision.

4 Assumptions

In this section we describe assumptions on the adversary model, key establishment protocol, and attack scenario.

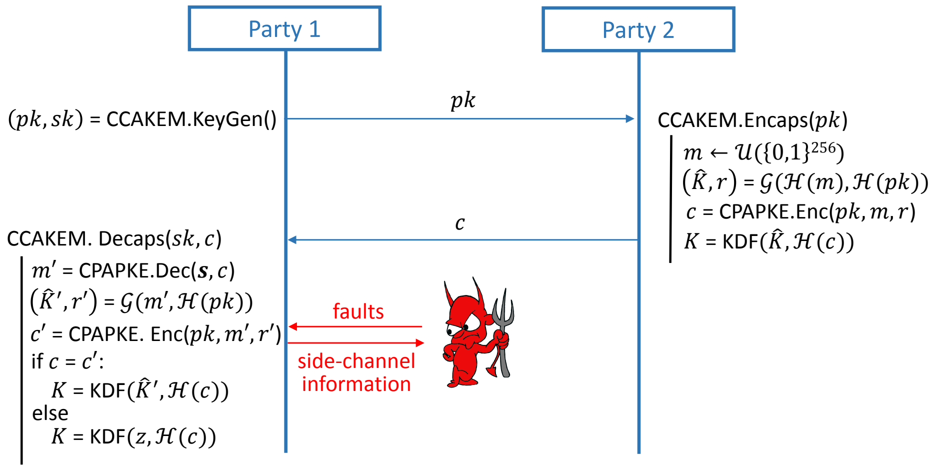
Party 1
Party 2
(pk, sk) = CCAKEM.KeyGen()
pk
CCAKEM.Encaps(pk)
$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$(\hat{K}, r) = \mathcal{G}(\mathcal{H}(m), \mathcal{H}(pk))$
c = CPAPKE.Enc(pk, m, r)
K = KDF($\hat{K}, \mathcal{H}(c)$)
c
CCAKEM. Decaps(sk, c)
m' = CPAPKE.Dec($\boldsymbol{s}, c$)
$(\hat{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
c' = CPAPKE. Enc(pk, m', r')
if $c = c'$:
K = KDF($\hat{K}', \mathcal{H}(c)$)
else
K = KDF($z, \mathcal{H}(c)$)
faults
side-channel information

Fig. 3: An LWE/LWR KEM-based shared key establishment protocol.

4.1 Adversary model

We make the following assumptions regarding the capabilities and the goals of the attacker.

The attacker is a clever outsider who has expertise in fault injection and side-channel analysis and all the necessary equipment and tools. The attacker also has physical access to the device to inject faults and measure the total power consumption.

The goal of the attacker is to recover the shared key K from side-channel information acquired from the device which runs the CRYSTALS-Kyber algorithm.

4.2 Shared key establishment protocol

We assume that the LWE/LWR KEM-based protocol shown in Fig. 3 is used to establish a shared key between two parties communicating over a public channel.

To initiate a new shared key establishment session, the Party 1 applies the key generation algorithm CCAKEM.KeyGen() to generate a fresh key pair (pk, sk) and sends the public key pk to the Party 2. Then, the Party 2 uses the encapsulation algorithm CCAKEM.Encaps() to generate a ciphertext c encapsulating a shared key K and returns c to the Party 1. Finally, the Party 1 uses the decapsulation algorithm CCAKEM.Decaps() to compute K from c.

4.3 Attack scenario

At the profiling stage, the attacker collects a set of power traces $\boldsymbol{T} = \{T_1, T_2, \ldots, T_{|\boldsymbol{T}|}\}$, $T_i \in \mathbb{R}^d$ captured during the execution of the decapsulation algorithm for ciphertexts c_i encrypting known messages m_i, where d is the number of datapoints in a trace and $i \in \{1, 2, \ldots, |\boldsymbol{T}|\}$. The traces are used to train a neural network $\mathcal{N} : \mathbb{R}^d \mapsto \{0,1\}^2$ which predicts message bits.

Note that, if the device under attack is used as a profiling device, then either the shuffling countermeasure should be disabled by fault injection, or ciphertexts encrypting all-0 and all-1 messages should be used for profiling. Otherwise, the order of message bits is unknown (and hence labels for traces). Alternatively, one can use multiple profiling devices, as in the attack on CRYSTALS-Kyber by Wang et al. [41], to minimize the negative effect of inter-device variability on neural network's classification accuracy.

At the attack stage, the attacker inserts a fault to disable shuffling and simultaneously measures the total power consumption of the device under attack during the decapsulation of the ciphertext c received from the Party 2. The model $\mathcal{N}$ trained at the profiling stage is then used to extract the message m encrypted in c from the resulting power trace T.

Once m is recovered, the pre-key $\bar{K}$ is derived as $(\bar{K}, r) = \mathcal{G}(m, \mathcal{H}(pk))$ (line 3 of KYBER.CCAKEM.Encaps() in Fig.2) and the shared key K is computed as $K = \mathsf{KDF}(\bar{K}, \mathcal{H}(c))$ (line 5 of KYBER.CCAKEM.Encaps() in Fig.2).

5 Experimental Setup

This section describes equipment we use for fault injection and power analysis as well as the target software implementation of CRYSTALS-Kyber.

5.1 Equipment

In the experiments, we use a ChipWhisperer-Husky, a CW313 adapter board and a CW308T-STM32F4 target device (see Fig. 4).

The target device contains an ARM Cortex-M4-based STM32F415RGT6. It is run at a frequency of 16 MHz with traces being captured with a sampling rate of 192 MS/s, which is the highest possible rate given the target frequency, meaning that we capture 12 data points per clock cycle.

We use ARM CoreSight ETM/DWT watchpoints to trigger fault injection and power trace capture. In this way, no modifications of the source code are required. In a real attack, reference waveforms for the power consumption, or external communication of the processor could be used to trigger fault injection.

5.2 Target implementation

In our experiments, we use a modified version of the masked CRYSTALS-Kyber implementation by Bronchain et al. [14] in which shuffling is added to the message encoding procedure `masked_poly_frommsg()` as shown in Listing 1. The message encoding is carried out during the execution of the encryption algorithm KYBER.CPAPKE.Enc(). The presented attack targets the re-encryption step of the FO transform of decapsulation (line 3 of KYBER.CCAKEM.Decaps()

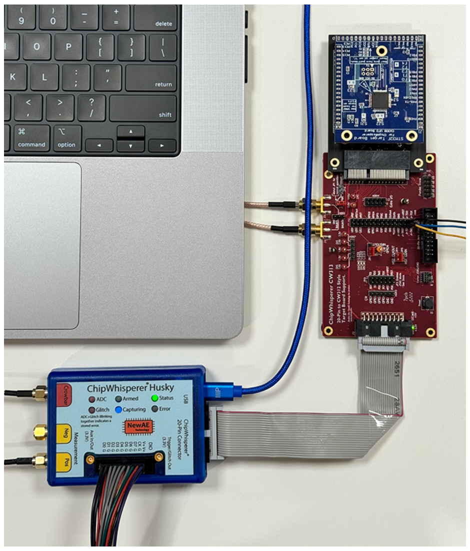

Fig. 4: ChipWhisperer-Husky, CW313 adapter board and CW308T-STM32F4 board used in the experiments.

in Fig. 2)). In principle, it can be similarly applied to the encryption step of encapsulation (line 4 of KYBER.CCAKEM.Encaps() in Fig. 2)[2].

The message encoding operation converts an array of 32 bytes representing a message m into a polynomial f with coefficients $f[j] = \lceil q/2 \rfloor \cdot m[j]$, for all $j \in \{0, 1, \cdots, 255\}$, where $m[j]$ is the jth bit of m, see $\mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$ at line 7 of KYBER.CPAPKE.Enc() in Fig. 1). The implementation of Bronchain et al. [14] employs the masking strategy of [37] in which a masked Boolean to arithmetic conversion algorithm is applied to transform the Boolean shares $\{m_0, m_1, \cdots, m_k\}$ into the arithmetic shares $\{f_0, f_1, \cdots, f_k\}$ such that, for all $j \in \{0, 1, \cdots, 255\}$, $\sum_{i=0}^{k} f_i[j] \bmod q = m[j]$. This is done by first extracting each Boolean share bit from the corresponding byte (line 10 in Listing 1) and then calling `secb2a_1bit()` to convert the extracted bit into the arithmetic domain (line 11).

In the C code of Listing 1, the additional code realizing shuffling is highlighted in orange. It is explained in more detail in Section 6.1.

The implementation is compiled using `arm-none-eabi-gcc` with the highest optimization level `-O3` (recommended default).

[2] Note that, in the CRYSTALS-Kyber implementation of [14] (as well as in many other implementations of CRYSTALS-Kyber), the encapsulation is not protected. Therefore, an unmasked message encoding is carried out during the execution of KYBER.CPAPKE.Enc() in KYBER.CCAKEM.Encaps().

```
void masked_poly_frommsg(StrAPoly y, uint8_t m[32 * NSHARES]) {
  uint8_t* shuffle = fy_gen_shuffle();
  uint32_t t1[NSHARES];
  int16_t t2[NSHARES];
  for (int x = 0; x < 256; x++) {
    int shuffled_index = shuffle[x];
    int i = shuffled_index / 8;
    int j = shuffled_index % 8;
    for (int k = 0; k < NSHARES; ++k)
      t1[k] = (m[i + k * 32] >> j) & 1;
    secb2a_1bit(NSHARES, t2, 1, t1, 1);
    for (int k = 0; k < NSHARES; ++k)
      y[k][i * 8 + j] = (t2[k] * ((KYBER_Q + 1) / 2)) % KYBER_Q;
  }
}
```

Listing 1: The modified C code of `masked_poly_frommsg()` procedure of CRYSTALS-Kyber implementation from [14] with shuffling added.

6 Fault injection method

This section describes the fault injection part of the presented attack. We begin by outlining the Fisher-Yates algorithm and the fault model's underlying assumptions. After that, we explain how voltage fault injection is carried out.

6.1 Fisher-Yates algorithm

As we mentioned earlier, shuffling is usually implemented using the Fisher-Yates algorithm [17, p. 26–27]. The algorithm has two phases. First, an array of n elements is created (lines 2-4 of Listing 2). Then, the elements of the array are shuffled from the last to the first as follows. At each iteration, an element from the remaining unshuffled elements is selected at random and swapped with the current element (lines 6-11). This is repeated until all elements have been shuffled.

6.2 Fault model

We assume a fault model that includes both single/multiple instruction skips and instruction corruption from a single glitch. While we empirically observed instruction corruptions during our experiments, these corruptions did not end up being useful in our attack and are only mentioned for the sake of completeness.

Unlike attacks based on electromagnetic fault injection, we do not consider modification of already stored data (i.e. bit flips) as a part of the fault model, though in some specific cases the manipulation of data can be approximated through instruction skipping and corruption.

```c
void fy_gen_shuffle(uint8_t[256] shuffle) {
  for (uint32_t i = 0; i < 256; i++) {
    shuffle[i] = i;
  }

  for (uint32_t i = 256 - 1; i > 0; i--) {
    uint32_t index = rng_get_random_blocking() % (i + 1);
    uint8_t temp = shuffle[index];
    shuffle[index] = shuffle[i];
    shuffle[i] = temp;
  }
}
```

Listing 2: The C code of the Fisher-Yates shuffle generation algorithm. Index generation is highlighted in blue and the instructions targeted by the fault injection are highlighted in red.

```asm
init_loop:
    strb.w     r3, [r2, #1]!
    adds       r3, #1
    cmp.w      r3, #256
    bne.n      init_loop
    add        r3, sp, #32
    addw       r6, sp, #287
    rsb        r4, r3, #1
shuffle_loop:
    bl         rng_get_random_blocking
    adds       r3, r4, r6
    udiv       r2, r0, r3
    mls        r0, r2, r3, r0
    add        r3, sp, #32
    add        r1, sp, #32
    ldrb       r3, [r3, r0]
    ldrb       r2, [r6, #0]
    strb       r2, [r1, r0]
    strb.w     r3, [r6], #-1
    cmp        r6, r1
    mov        r3, r1
    bne        shuffle_loop
```

Listing 3: Assembly code of the Fisher-Yates shuffle generation algorithm. Index generation is highlighted in blue and instructions targeted by the fault injection are highlighted in red.

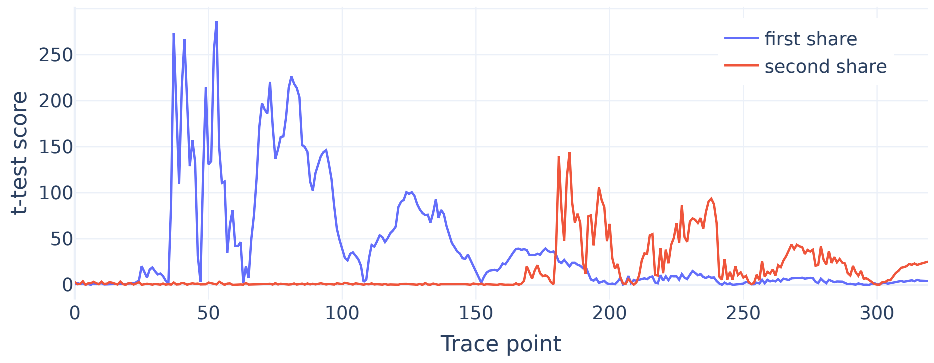

Fig. 5: T-test results for a single bit of each Boolean share during their processing by `secb2a_1bit()`.

6.3 Main idea

The presented fault attack exploits the fact that elements generated at the first phase of the Fisher-Yates algorithm are in a known order before being shuffled at the second phase. Thus, by skipping the second phase, we can force `fy_gen_shuffle()` to return a known sequence of elements, bypassing the shuffling countermeasure.

To skip the second phase, we perform a voltage fault injection targeting the conditional backwards branch (`bne`) which forms the loop at the second phase (line 22 of the assembly code in Listing 3). By skipping this instruction, we exit the loop after the first iteration. Therefore, all but two elements (the last one and the element it gets swapped with) retain their original order.

We further improve upon this result by widening the injected glitch to skip the `strb` and `strb.w` operations as well (lines 18 and 19 in Listing 3). By canceling the execution of these operations, we omit swapping of these two elements, thus retaining the original, sorted order of array elements. Running the `masked_poly_frommsg()` procedure with this order has the same effect as running an implementation without shuffling.

7 Power analysis method

In this section, we describe the power analysis part of the presented attack. We use a deep learning-assisted profiled attack method based on the Hamming weight leakage model similar to the one employed in the side-channel attacks on CRYSTALS-Kyber presented in [4] and [41]. A new contribution is a partial key enumeration method which allows us to reach a high full message recovery probability with a subpar average message bit recovery probability.

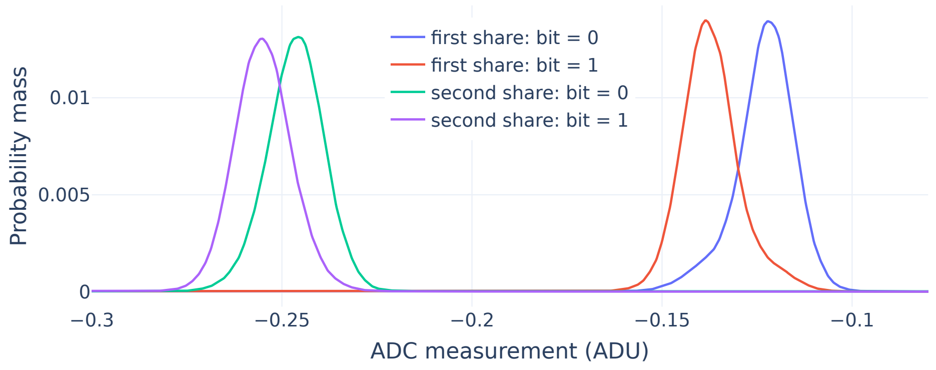

Fig. 6: Distributions of power consumption for a single bit of each Boolean share during their processing by `secb2a_1bit()`.

7.1 Leakage analysis

For the extraction of message bits, we exploit the leakage of the masked Boolean to arithmetic conversion procedure `secb2a_1bit()` which is called during the message encoding. As mentioned in Section 5.2, in the message encoding procedure `masked_poly_frommsg()`, each Boolean share bit is extracted from the corresponding byte one-by-one (see line 10 in Listing 1) and then `secb2a_1bit()` is called to convert the extracted bit into the arithmetic domain (line 11). For an extensive analysis of the leakage of `masked_poly_frommsg()` procedure, we refer the reader to [41] in which a pure side-channel attack on the masked implementation of CRYSTALS-Kyber by Bronchain el at. [14] is described. The leakage we exploit in this work is called *direct-copy* leakage in [41].

Figure 5 shows t-test results for a single bit of each Boolean share during their processing by `secb2a_1bit()` procedure (for 1K traces). The traces for the test are acquired from a device running an implementation with known masks. They are used for leakage analysis only. One can see that the first share leaks stronger than the the second share. This is because they are treated differently by `secb2a_1bit()`, see Listing 4.

Figure 6 shows the distributions of power consumption for a single bit of each Boolean share during their processing by `secb2a_1bit()`. The overlap between the distributions for the first share (on the right) is smaller than the overlap for the second share (on the left), indicating a stronger leakage for the first share. This is in line with the t-test results in Figure 5.

7.2 Trace preprocessing and neural network training

For profiling, we use a 2.56M dataset obtained by capturing $t = 10$K traces $\{T_1, T_2, \ldots, T_t\}$ for known messages $\{m_1, m_2, \ldots, m_t\}$. Then, we apply the cut-and-join technique of [25] to divide each T_i into $n = 256$ segments, $T_i[j]$, covering the processing of the jth bits of both Boolean shares, $m_{i,0}[j]$ and $m_{i,1}[j]$, of the

```c
void secb2a_1bit(size_t nshares, int16_t *a, uint32_t *x) {
  b2a_qbit(nshares, a, x);
  refresh_add(nshares, a);
}

void b2a_qbit(size_t nshares, int16_t *a, uint32_t *x) {
  a[0] = x[0]; /* First share */
  for (size_t i = 1; i < nshares; i++) {
    secb2a_qbit_n(i + 1, a, a, x[i]); /* Second share */
  }
}
```

Listing 4: The C code of `secb2a_1bit()` procedure

message $m_i = m_{i,0} \oplus m_{i,1}$. The segments $T_i[j]$ are labeled by the corresponding message bit value $l(T_i[j]) = m_i[j]$, for all $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, n\}$ (which is equivalent to the Hamming weight of $m_i[j]$).

We perform a segmented capture, therefore traces do not require any additional synchronization. If the segmented capture is not possible, synchronization via cross-correlation with templates can be used instead, as in [41].

We use multilayer perceptron (MLP) neural networks with the architecture shown in Table 2. The networks are of type $\mathcal{N} : \mathbb{R}^d \mapsto \{0,1\}^2$, where d is the number of datapoints in a trace segment $T_i[j]$. A network $\mathcal{N}$ maps each $T_i[j]$ into a score vector $S_{i,j} = \mathcal{N}(T_i[j])$ such that its kth element $s_{i,j,k}$ represents the probability that $m_i[j]$ is equal to $k \in \{0,1\}$ in $T_i[j]$:

$$s_{i,j,k} = \Pr[m_i[j] = k].$$

During training, we use the Nadam optimiser with a learning rate of 0.001 and a numerical stability constant $\epsilon = 10^{-8}$. We train for a maximum of 250 epochs using early stopping with a patience of 15 and a batch size of 4096. We use 70% of the data for training and 30% for validation.

7.3 Partial key enumeration method

Messages of CRYSTALS-Kyber are relatively long, $n = 256$ bits. This puts a tough requirement on the lower bound of the average message bit recovery probability, e.g. $p_{bit} = 0.9973$ is needed to get a full message recovery probability of $p_m = (p_{bit})^{256} = 0.50$. In this section we propose a simple approach for partial key enumeration which significantly lowers the requirement on p_{bit}. For instance, it allows us to reach $p_m = 0.887$ with the average $p_{bit} = 0.9894$ and 32 enumerated bits. Without the enumeration, we would only get $p_m = 0.122$.

In the traditional key enumeration method, a key $K = K_1||K_2||\ldots||K_r$ is recovered by enumerating the l most likely candidates for each subkey K_i, where $|K_i| = |K|/r$ is the size of K_i, for $i \in \{1, 2, \ldots, r\}$. The likelihood of candidates is

Table 2: Neural network architecture used for message recovery.

Layer type	Output shape
Batch Normalization 1	320
Dense 1	320
ReLU	320
Batch Normalization 2	320
Dense 2	256
ReLU	256
Batch Normalization 3	256
Dense 3	128
ReLU	128
Batch Normalization 4	128
Dense 4	2
Softmax	2

determined using various methods, e.g., in correlation power analysis, the Person correlation coefficient is used [12]. If the subkey size is a byte, then, e.g., in an attack on AES-128, one can enumerate the two most likely candidates for each of the 16 subkeys, resulting in the total enumeration complexity of 2^{16}. In our attack on CRYSTALS-Kyber, the "key" is a 256-bit message and the subkey size is one (bit). Hence, the traditional enumeration is not feasible.

Instead, we sort the elements of score vectors $S_j = \mathcal{N}(T[j])$ inferred by the neural network $\mathcal{N}$ for each trace segment $T[j]$, $j \in \{1, 2, \ldots, n\}$, where T is the trace captured from the device under attack during the decapsulation of the ciptertext c, and identify k message bits predicted by $\mathcal{N}$ with the least confidence. These bits are then enumerated over all 2^k possible assignments. The Algorithm 1 gives the details.

This simple method is effective because the distribution curve of maximum predicted class probabilities approaches the value of 1 sharply, see Figure 7. Therefore, enumerating a small subset of bits, e.g. 32, significantly improves average message bit recovery probability of the remaining bits.

8 Experimental Results

This section describes the results of our message recovery attack combining voltage fault injection and deep learning-assisted power analysis.

8.1 Glitch settings and evaluation

By using a grid search over the set of parameters provided by the ChipWhisperer-Husky, we found settings that consistently glitch the desired instructions without crashing the device. Namely, we employed the `'enable_only'` mode to insert a

Algorithm 1 Partial key enumeration

```
Input: trace $T$, ciphertext $c$, neural network $\mathcal{N}$, number of enumerated bits $k$
Output: shared key $K$
 1: for each message bit $j$ from 1 to $n$ do
 2:     $(s_{j,0}, s_{j,1}) = \mathcal{N}(T[j])$
 3:     $Pr[j] = \max(s_{j,0}, s_{j,1})$ /* array of maximum predicted class probabilities */
 4:     if $s_{j,0} > s_{j,1}$ then
 5:         $m[j] = 0$
 6:     else
 7:         $m[j] = 1$
 8:     end if
 9: end for
10: Select $k$ smallest elements of $Pr$ and store their indices as $I = \{i_1, \ldots, i_k\}$
11: for each $x \in \{0,1\}^k$ do
12:     $(m[i_1], \ldots, m[i_k]) = x$
13:     $(\hat{K}', r) = \mathcal{G}(m, \mathcal{H}(pk))$
14:     $c' =$ KYBER.CPAPKE.Enc$(pk, m, r)$
15:     if $c = c'$ then
16:         return $K = \mathsf{KDF}(\hat{K}, \mathcal{H}(c))$
17:     end if
18: end for
19: return Failed to recover $K$
```

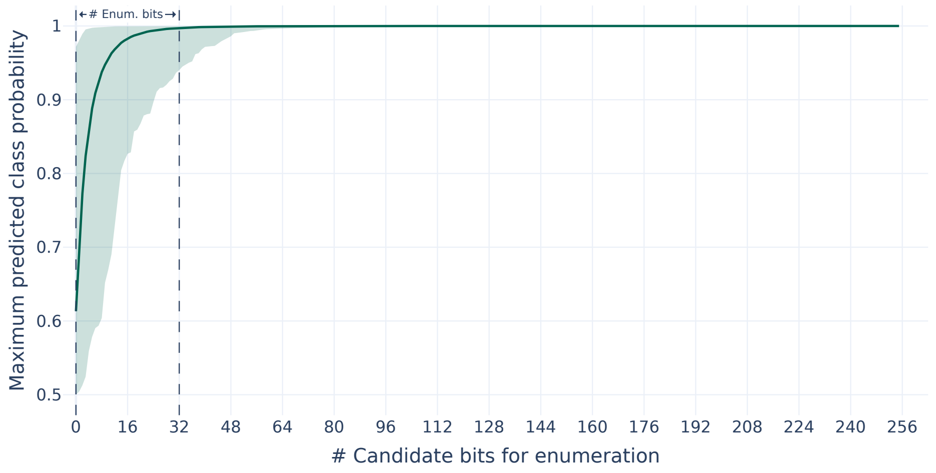

Fig. 7: Selection of a subset of bits for enumeration based on the empirical maximum predicted class probability. The mean probability value over 1K tests is shown by the green line.

glitch lasting for four clock cycles using both the high-power and low-power crowbar MOSFETs at an offset of 700 units[3].

[3] These units are dimensionless and depend on the internal frequencies of the ChipWhisperer-Husky, though the offset always describes the distance between the rising edge of the clock cycle and the beginning of the glitch.

Note that the insertion of a glitch over four clock cycles does not imply that only four instructions are affected by this glitch. Furthermore note that not all instructions targeted by the glitch are single-cycle instructions. Indeed, we found that various combinations of offsets and glitch lengths yielded similar results with respect to affecting the generated Fisher-Yates shuffle. However, larger glitch lengths increased the probability of the device crashing.

Table 3 shows the probability of inserting a successful glitch with the settings described above. While all our glitch attempts succeeded in affecting the generated shuffle, we found that in 15.8% of cases the processing of bits is shifted by one iteration as a consequence of the glitch. In these cases, the first iteration would read memory out of bounds and thus produce no exploitable leakage. The second iteration would correspond to the processing of the first message bit, and so on. Consequently, in these cases it is only possible to recover 255 of the 256 message bits because the total number of iterations stays the same. We account for these cases during the message recovery by considering both the message as it is predicted, and its shifted variant in which we enumerate the last bit.

Table 3: Glitch success probability for 1000 attempts.

Successful		Unsuccessful
full	shifted	
0.842	0.158	0.0

8.2 Message recovery

We trained neural networks as described in Section 7.2 to perform message recovery after the successful fault injection. In order to reduce the trace capture time, we run only the part of the decapsulation algorithm that contains the masked message encoding procedure `masked_poly_frommsg()`. This has no practical effect on the attack, as the other parts of the decapsulation algorithm are not used in the attack. The neural networks were trained and tested on a PC with an AMD Ryzen 7 Pro 5850U running at 1.9 GHz and 32GB RAM.

Table 4 and Figure 8 show the resulting empirical probability (mean over 1K different messages selected at random) to recover a full message from a single trace using the partial key enumeration method described in Section 7.3. Obviously, the probability increases as the number of enumerated bits grows. One can see that, without the partial enumeration, the probability for full message recovery would be 0.122 only. The partial enumeration allows us to increase the success probability to 0.887 for 32 enumerated bits, and to 0.969 for 64 enumerated bits.

As mentioned in Section 8.1, due to the shifting that can occur as a side effect of the glitch, we need to consider both the message as it is predicted

Table 4: Empirical single-trace full message recovery probability (mean over 1K tests) for a given number of enumerated bits.

# Enumerated bits												
0	1	2	3	4	5	6	7	8	9	10	11	12
0.122	0.217	0.297	0.354	0.410	0.460	0.502	0.530	0.560	0.586	0.609	0.631	0.649
13	14	15	16	17	18	19	20	21	22	23	24	25
0.667	0.679	0.691	0.706	0.727	0.747	0.762	0.769	0.791	0.802	0.810	0.823	0.831
26	27	28	29	30	31	**32**	33	34	35	36	37	38
0.838	0.847	0.855	0.864	0.878	0.881	**0.887**	0.891	0.895	0.900	0.904	0.909	0.910
39	40	41	42	43	44	45	46	47	48	49	50	51
0.914	0.919	0.922	0.925	0.928	0.928	0.928	0.930	0.933	0.935	0.939	0.943	0.946
52	53	54	55	56	57	58	59	60	61	62	63	64
0.948	0.951	0.954	0.955	0.956	0.957	0.958	0.958	0.960	0.961	0.964	0.965	0.969

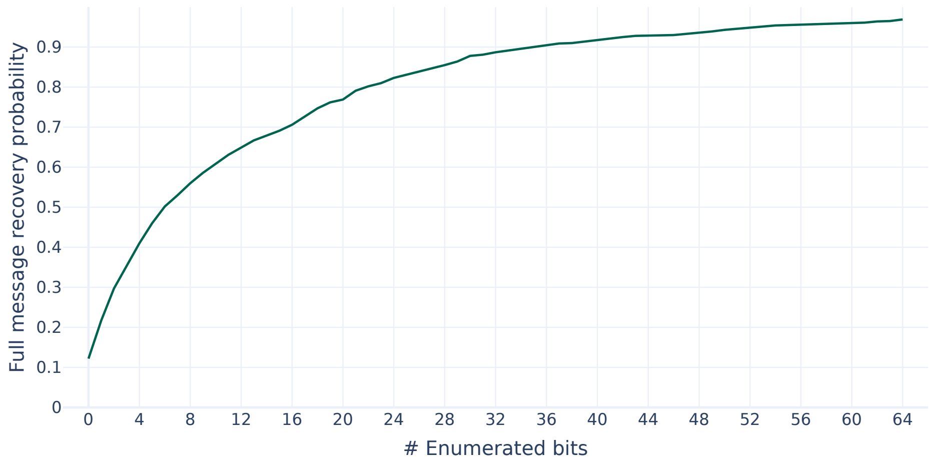

Fig. 8: Empirical single-trace full message recovery probability (mean over 1K tests) as a function of the number of enumerated bits.

and its shifted counterpart, in which we guess the last bit. This increases the complexity of enumeration from 2^k to 2^{k+1}, where k is the number of enumerated bits.

9 Countermeasures

The presented fault injection method would not be applicable if the algorithm generating the shuffled indices did not contain any control flow operations that can be targeted by a glitch, i.e. if the loops were unrolled. Note however that, given the high stability of the glitch demonstrated in this work, it is likely possible to generate a partially predictable sequence through injection of multiple faults into an unrolled implementation. Clearly, the complexity of such an attack would be higher than that of the presented one.

It is also possible to apply a technique similar to the *dynamic loop counter* introduced in [34] to ensure that no iterations of the loops used in the shuffle generation are skipped. In this technique, a counter is initialised to a random value and incremented at every iteration of the loops. When the shuffled indices are later used in a different loop, the value of the counter is verified to contain a specific multiple of the random value, indicating that all iterations of the loops were executed. However, similarly to the unrolled implementation case, it may be possible to generate a partially predictable sequence through injection of multiple faults. Additionally, the dynamic loop counter can only ensure that the shuffling loop was executed. Thus a sophisticated fault injection attack may shift the area of memory affected by the shuffling into unused memory and execute the shuffling as normal, thereby bypassing the countermeasure.

Finally, avoiding the creation of a sorted array of indices in the first place seems to be the most robust countermeasure against the presented fault injection method. Instead of first generating a sorted array and then shuffling it, one can generate a random permutation directly by iteratively querying a random number generator and discarding repeated indices until all indices have been generated. Provided the uninitialised sequence of indices is sufficiently unpredictable, such an approach would not be exploitable by the presented attack at the cost of significantly increased runtime.

10 Conclusions

We presented a practical message recovery attack on a masked and shuffled software implementation of Kyber-768 that combines voltage fault injection with deep learning-assisted power analysis. We found settings for voltage fault injection that consistently glitch the desired instructions without crashing the device. These setting are not specific to CRYSTALS-Kyber, they are tailored to the Fisher-Yates algorithm. Thus, they may be applicable to other cryptographic algorithms which use Fisher-Yates-based shuffling for their protection.

We also proposed a method for partial key enumeration which allows us to significantly increase the success rate of message recovery. Again, the method is not specific to CRYSTALS-Kyber. It may help increase the success rate of side-channel attacks on other cryptographic algorithms.

Our work demonstrates that it is possible to recover a shared key from a single trace of a masked and shuffled implementation of CRYSTALS-Kyber.

This is clearly alarming since the presented attack is equally applicable to the message encoding procedure carried out during the encryption step of the encapsulation algorithm of CRYSTALS-Kyber. A common opinion at present is that single-trace shared key recovery attacks on the encapsulation algorithm of CRYSTALS-Kyber are not possible. For this reason, the existing implementations of CRYSTALS-Kyber protect the decapsulation algorithm only. Our results show that the encapsulation algorithm also needs protection. They also show that the conventional countermeasures such as masking and shuffling may not be able to stop advanced attacks.

Future work includes developing stronger countermeasures against side-channel and fault attacks on implementations of PQC algorithms.

11 Acknowledgments

The authors are grateful to Linus Backlund for his help with implementing the shuffling countermeasure, setting up equipment for experiments, and diagnosing and resolving issues.

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

References

1. 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
2. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and active combined attacks: Combining fault attacks and side channel analysis. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007). pp. 92–102 (Sept 2007). https://doi.org/10.1109/FDTC.2007.12
3. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber algorithm specifications and supporting documentation (2021)
4. Backlund, L., Ngo, K., Gärtner, J., Dubrova, E.: Secret key recovery attack on masked and shuffled implementations of CRYSTALS-Kyber and Saber. In: Zhou, Jianying et al. (ed.) Applied Cryptography and Network Security Workshops. pp. 159–177. Springer Nature Switzerland, Cham (2023)
5. Baksi, A., Bhasin, S., Jakub, B., Jap, D., Saha, D.: A survey on fault attacks on symmetric key cryptosystems. ACM Computing Surveys **55** (04 2022). https://doi.org/10.1145/3530054
6. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE **100**(11), 3056–3076 (Nov 2012)
7. Barenghi, A., Bertoni, G., Breveglieri, L., Pellicioli, M., Pelosi, G.: Low voltage fault attacks to AES and RSA on general purpose processors. IACR Cryptol. ePrint Arch. p. 130 (2010), http://eprint.iacr.org/2010/130

8. Barenghi, A., Bertoni, G., Parrinello, E., Pelosi, G.: Low voltage fault attacks on the RSA cryptosystem. In: Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J. (eds.) Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009. pp. 23–31. IEEE Computer Society (2009). https://doi.org/10.1109/FDTC.2009.30, https://doi.org/10.1109/FDTC.2009.30
9. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., et al.: Gimli. Submission to the NIST Lightweight Cryptography project (2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/gimli-spec.pdf
10. Bhasin, S., D'Anvers, J.P., Heinz, D., Pöppelmann, T., Beirendonck, M.V.: Attacking and defending masked polynomial comparison for lattice-based cryptography. Cryptology ePrint Archive, Paper 2021/104 (2021)
11. Bozzato, C., Focardi, R., Palmarini, F.: Shaping the glitch: Optimizing voltage fault injection attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 199–224 (2019). https://doi.org/10.13154/tches.v2019.i2.199-224, https://doi.org/10.13154/tches.v2019.i2.199-224
12. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004. pp. 16–29 (2004)
13. Brisfors, M., Forsmark, S., Dubrova, E.: How deep learning helps compromising USIM. In: Proc. of the 19th Smart Card Research and Advanced Application Conference (CARDIS'2020) (Nov 2020)
14. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. IACR Cryptol. ePrint Arch. p. 158 (2022), https://eprint.iacr.org/2022/158
15. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in Cryptology - CRYPTO '99. vol. 1666, pp. 398–412. Springer (1999)
16. Dubrova, E., Ngo, K., Gartner, J.: Breaking a fifth-order masked implementation of CRYSTALS-Kyber by copy-paste. In: Proc. of the 10th ACM Asia Public-Key Cryptography Workshop (APKC 2023) (2023)
17. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural and medical research (3rd ed.). London: Oliver and Boyd (1948)
18. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual international cryptology conference. pp. 537–554. Springer (1999)
19. Guo, Q., Nabokov, D., Nilsson, A., Johansson, T.: SCA-LDPC: A code-based framework for key-recovery side-channel attacks on post-quantum encryption schemes. Cryptology ePrint Archive (2023)
20. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 88–113 (2021)
21. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D.: First-order masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Paper 2022/058 (2022)
22. Moraitis, M., Ji, Y., Brisfors, M., Dubrova, E., Lindskog, N., Englund, H.: Securing CRYSTALS-Kyber in FPGA using duplication and clock randomization. IEEE Design & Test (2023). https://doi.org/10.1109/MDAT.2023.3298805

23. Ngo, K., Dubrova, E.: Side-channel analysis of the random number generator in STM32 MCUs. In: Proc. of the Great Lakes Symposium on VLSI (GLSVLSI '22) (2022)
24. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked IND-CCA secure Saber KEM implementation. IACR Trans. on Cryptographic Hardware and Embedded Systems pp. 676–707 (2021)
25. Ngo, K., Dubrova, E., Johansson, T.: Breaking masked and shuffled CCA secure Saber KEM by power analysis. In: Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security. pp. 51–61 (2021)
26. Ngo, K., Wang, R., Dubrova, E., Paulsrud, N.: Higher-order boolean masking does not prevent side-channel attacks on LWE/LWR-based PKE/KEMs. In: IEEE International Symposium on Multiple-Valued Logic (2023)
27. NIST: PQC standardization process: Announcing four candidates to be standardized, plus fourth round candidates. NIST Computer Security Resource Center (July 2022), https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4
28. O'Flynn, C.: Fault injection using crowbars on embedded systems. IACR Cryptol. ePrint Arch. p. 810 (2016), http://eprint.iacr.org/2016/810
29. Panoff, M., Yu, H.S.A., Shan, H., Jin, Y.: A review and comparison of AI-enhanced side channel analysis. ACM Journal on Emerging Technologies in Computing Systems (JETC) **18**, 1 – 20 (2022), https://api.semanticscholar.org/CorpusID:247617543
30. Papadimitriou, A., Nomikos, K., Psarakis, M., Aerabi, E., Hély, D.: You can detect but you cannot hide: Fault assisted side channel analysis on protected software-based block ciphers. In: Dilillo, L., Psarakis, M., Siddiqua, T. (eds.) IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2020, Frascati, Italy, October 19-21, 2020. pp. 1–6. IEEE (2020). https://doi.org/10.1109/DFT50435.2020.9250870, https://doi.org/10.1109/DFT50435.2020.9250870
31. Patranabis, S., Breier, J., Mukhopadhyay, D., Bhasin, S.: One plus one is more than two: A practical combination of power and fault analysis attacks on PRESENT and present-like block ciphers. IACR Cryptol. ePrint Arch. p. 1073 (2017), http://eprint.iacr.org/2017/1073
32. Randolph, M., Diehl, W.: Power side-channel attack analysis: A review of 20 years of study for the layman. Cryptography **4**(2) (2020). https://doi.org/10.3390/cryptography4020015, https://www.mdpi.com/2410-387X/4/2/15
33. Ravi, P., Bhasin, S., Roy, S.S., Chattopadhyay, A.: On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks. IEEE Transactions on Information Forensics and Security (2021)
34. Ravi, P., Chattopadhyay, A., Baksi, A.: Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. IACR Cryptol. ePrint Arch. p. 737 (2022), https://eprint.iacr.org/2022/737
35. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR Trans. on Cryptographic Hardware and Embedded Systems. pp. 307–335 (2020)
36. Saha, S., Ravi, P., Jap, D., Bhasin, S.: Non-profiled side-channel assisted fault attack: A case study on DOMREP. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023. pp. 1–6. IEEE (2023). https://doi.org/10.23919/DATE56975.2023.10137176, https://doi.org/10.23919/DATE56975.2023.10137176

37. Schneider, T., Paglialonga, C., Oder, T., Güneysu, T.: Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In: Public-Key Cryptography–PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II 22. pp. 534–564. Springer (2019)
38. Shen, M., Cheng, C., Zhang, X., Guo, Q., Jiang, T.: Find the bad apples: An efficient method for perfect key recovery under imperfect SCA oracles – a case study of Kyber. Cryptology ePrint Archive, Paper 2022/563 (2022), https://eprint.iacr.org/2022/563
39. of Standards, N.I., Technology: Module-lattice-based key-encapsulation mechanism standard FIPS 203 (Draft) (2023). https://doi.org/10.6028/NIST.FIPS.203.ipd
40. Wang, J., Cao, W., Chen, H., Li, H.: Practical side-channel attack on message encoding in masked Kyber. In: 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 882–889. IEEE (2022)
41. Wang, R., Brisfors, M., Dubrova, E.: A side-channel attack on a bitsliced higher-order masked CRYSTALS-Kyber implementation. IACR Cryptol. ePrint Arch. p. 1042 (2023), https://eprint.iacr.org/2023/1042
42. Yao, Y., Yang, M., Patrick, C., Yuce, B., Schaumont, P.: Fault-assisted side-channel analysis of masked implementations. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018. pp. 57–64. IEEE Computer Society (2018). https://doi.org/10.1109/HST.2018.8383891, https://doi.org/10.1109/HST.2018.8383891