A Side-Channel Attack on a Hardware Implementation of CRYSTALS-Kyber

Yanning Ji  
*KTH Royal Institute of Technology*  
Stockholm, Sweden  
yanning@kth.se

Ruize Wang  
*KTH Royal Institute of Technology*  
Stockholm, Sweden  
ruize@kth.se

Kalle Ngo  
*KTH Royal Institute of Technology*  
Stockholm, Sweden  
kngo@kth.se

Elena Dubrova  
*KTH Royal Institute of Technology*  
Stockholm, Sweden  
dubrova@kth.se

Linus Backlund  
*KTH Royal Institute of Technology*  
Stockholm, Sweden  
lbackl@kth.se

Abstract—CRYSTALS-Kyber has been recently selected by the NIST as a new public-key encryption and key-establishment algorithm to be standardized. This makes it important to assess how well CRYSTALS-Kyber implementations withstand side-channel attacks. Software implementations of CRYSTALS-Kyber have been already analyzed and the discovered vulnerabilities were patched in the subsequently released versions. In this paper, we present a profiling side-channel attack on a hardware implementation of CRYSTALS-Kyber with the security parameter \( k = 3 \), Kyber768. Since hardware implementations carry out computation in parallel, they are typically more difficult to break than their software counterparts. We demonstrate a successful message (session key) recovery by deep learning-based power analysis. Our results indicate that currently available hardware implementations of CRYSTALS-Kyber need better protection against side-channel attacks.

Index Terms—Post-quantum cryptography, CRYSTALS-Kyber, LWE-based KEM, side-channel attack, FPGA, power analysis, deep learning

I. INTRODUCTION

In July 2022, the National Institute of Standards and Technology (NIST) has selected CRYSTALS-Kyber as a new public-key encryption and key-establishment algorithm to be standardized [1]. CRYSTALS-Kyber is a lattice-based cryptographic algorithm that is expected to be capable of protecting confidential information after the emergence of large-scale quantum computers. Even if it might take many years until large-scale quantum computers become a reality, the need for long-term security necessitates an early start of the transition. The industry is preparing to plan and budget for a shift to quantum-resistant cryptographic algorithms. The National Security Agency (NSA) has recently included CRYSTALS-Kyber in the suite of cryptographic algorithms recommended for national security systems [2].

The standardization of CRYSTALS-Kyber makes it important to assess the resistance of its implementations to side-channel attacks (SCAs). Several software implementations of CRYSTALS-Kyber have been already analyzed [3]–[6] and the discovered vulnerabilities were patched in the subsequently released versions. However, to the best of our knowledge, no successful attack on a hardware implementation of CRYSTALS-Kyber has been reported until now. Hardware implementations are typically significantly more difficult to break than software variants because they carry out the computation in parallel. There are opinions that the parallelism, in a combination with a smaller minimum feature size of application-specific integrated circuits (ASICs) of field-programmable gate array (FPGAs) process technologies may provide a sufficient resistance against side-channel analysis without any SCA-specific countermeasures.

Our Contributions: In this paper, we present a profiling side-channel attack on a hardware implementation of CRYSTALS-Kyber with the security parameter \( k = 3 \), Kyber768. Using deep learning-based power analysis, we successfully recover messages from a Xilinx Artix-7 FPGA implementation of CRYSTALS-Kyber from [7]. In CRYSTALS-Kyber, a message recovery from a properly generated ciphertext implies the session key recovery since the session key is derived from the message using hash functions. Furthermore, by recovering messages contained in seven chosen ciphertexts [8], one can extract the secret key of CRYSTALS-Kyber. Unlike the non-profiling attack on Kyber512 presented in [9], our attack does not require any knowledge of register reference values. The success of our attack is due to the new sliced multi-bit error injection method which is an extension of the original multi-bit error injection method introduced in [10] for side-channel analysis of software implementations of lattice-based PKE/KEMs. Software implementations do not require slicing because they execute instructions sequentially. However, slicing is essential for hardware implementations. Our experimental results show that, without slicing, only 10% of messages can be recovered with enumeration up to \( 2^{64} \). With slicing, we can recover all messages with the same enumeration.

The success of our attack is due to the new sliced multi-bit error injection method which is an extension of the original multi-bit error injection method introduced in [10] for side-channel analysis of software implementations of lattice-based PKE/KEMs. Software implementations do not require slicing because they execute instructions sequentially. However, slicing is essential for hardware implementations. Our experimental results show that, without slicing, only 10% of messages can be recovered with enumeration up to \( 2^{64} \). With slicing, we can recover all messages with the same enumeration.

The rest of this paper is organized as follows. Section II reviews previous work related to side-channel analysis of CRYSTALS-Kyber implementations. Section III gives a background on CRYSTALS-Kyber algorithm. Section IV describes the equipment used in the experiments. Sections V and VI...
present the profiling and attack stages, respectively. Section VII summarizes experimental results. Section VIII concludes the paper and discusses future work.

II. PREVIOUS WORK

Side-channel attacks on implementations of post-quantum cryptographic algorithms have drawn a lot of attention since the beginning of the NIST post-quantum cryptography (PQC) standardization process in 2016. The lattice-based public key encryption (PKE) algorithms and key encapsulation mechanisms (KEMs) were the most popular targets. This is due to the fact that many of the NIST PQC candidates are based on lattice problems: an NTRU-based scheme NTRU [11], a Learning With Errors (LWE)-based scheme CRYSTALS-Kyber [12], and a Learning With Rounding (LWR)-based scheme Saber [13]. The majority of the presented side-channel attacks are on software implementations.

In [14], profiling deep-learning-based message recovery attacks using a single power trace representing the execution of the encapsulation procedure are demonstrated for unprotected software implementations of CRYSTALS-Kyber, Saber, and FrodoKEM. In [15], near field EM secret key recovery attacks on unprotected and protected implementations of the same algorithms are presented. It is shown how their masked implementations can be broken in two steps, by attacking each share individually. In [16] an attack on a first-order masked implementation of CRYSTALS-Kyber using the one-step method introduced in [17] is reported. It exploits the message encoding vulnerability discovered in [14].

In [18], secret key recovery attacks on software implementations of lattice-based KEMs, including CRYSTALS-Kyber, Saber, and NTRU, based on power/near field EM analysis are demonstrated. These attacks make use of the side-channel leakage during the re-encryption step of the decapsulation procedure. In [19], correlation power analysis-based attacks of all lattice-based candidates of the NIST are presented, targeting the polynomial multiplication in unprotected software implementations.

In [8], a near field EM emanations-based message recovery attack on an unprotected software implementation of CRYSTALS-Kyber making use of a vulnerability in the Fujisaki-Okamoto (FO) transform is described. In [20] an EM-based chosen ciphertext side-channel attack on an unprotected software implementation of CRYSTALS-Kyber is shown. In [21], side-channel attacks on two implementations of masked polynomial comparison, demonstrated on CRYSTALS-Kyber, are presented.

All attacks mentioned above targeted software implementations of CRYSTALS-Kyber and other lattice-based algorithms. Since these algorithms perform many repeated computations, it is of advantage to use hardware for their implementations. In [22] a hardware implementation of CRYSTALS-Kyber in FPGA is described which takes 10 times less clock cycles than the ARM Cortex-M4 implementation of CRYSTALS-Kyber from [23].

In [7] an even smaller and faster hardware implementation of CRYSTALS-Kyber is presented. With suitably designed pipelines and well-optimized architecture, this implementation is capable of executing the decapsulation procedure of any version of CRYSTALS-Kyber in 14,000 clock cycles while consuming 7,500 LUTs only. Thus, it can fit into the smallest Xilinx Artix-7 series devices.

However, the evaluation of resistance of hardware implementations of PQC KEMs to side-channel analysis is still in the beginning. In [22] a power-based key recovery attack on the McEliece cryptosystem, a round 4 candidate of the NIST PQC, implemented in Xilinx Virtex-7 FPGA is presented. In [25], a deep learning-based power analysis of Saber implemented in Xilinx Artix-7 FPGA is presented. Due to the parallel processing of hardware, the MLP models are able to predict the Hamming weight of messages with the accuracy of 88% at most. The message bits cannot be distinguished accurately. In [26] a preliminary power analysis of a hardware implementation of CRYSTALS-Kyber from [22] is presented. Leakage points were found using a t-test for the Kyber512 implementation in Xilinx Virtex-7 FPGA.

In [9], a non-profiling correlation EM analysis of an FPGA implementation of Kyber512 (security parameter $k = 2$) is presented targeting polynomial multiplication during the PKE decryption step of the decapsulation algorithm. The attack utilizes 166,620 traces to recover the secret key. However, it requires knowledge of the register reference values.

To the best of our knowledge, no successful side-channel attack on a hardware implementation of Kyber768 (security parameter $k = 3$) has been demonstrated so far. Typically, the higher the security parameter, the more difficult the implementation is to break.

III. CRYSTALS-KYBER ALGORITHM

CRYSTALS-Kyber [12] is an IND-CCA2-secure cryptographic algorithm which means that it is indistinguishable under an adaptive chosen ciphertext attack. The security of CRYSTALS-Kyber relies on the hardness of the Module Learning with Errors (M-LWE) problem.

CRYSTALS-Kyber contains a CPA-secure PKE scheme, KYBER.CPAPKE, and a CCA-secure KEM scheme, KYBER.CCAKEM, based on a post-quantum version of the Fujisaki-Okamoto transform [27]. These algorithms are described in Fig. 1 and Fig. 2 respectively.

Let $\mathbb{Z}_q$ denote the ring of integers modulo a prime $q$, and $R_q$ denote the quotient ring $\mathbb{Z}_q[X]/(X^n + 1)$. CRYSTALS-Kyber works with vectors of ring elements in $R_q^k$, where $k$ is the lattice dimension used to scale the security levels.

The term $x \leftarrow \chi(S;r)$ denotes sampling $x$ from a distribution $\chi$ over a set $S$ using seed $r$. The uniform distribution is denoted by $\mathcal{U}$. The centered binomial distribution with parameter $\mu$ is denoted by $\beta_\mu$.

The Decode function is designed to decode an array of 32l bytes into a 256 length polynomial with each coefficient in $\{0,1,\cdots,2^l-1\}$. The Encode function is the inverse of Decode, which encodes each polynomial coefficient
Table I. Parameters of CRYSTALS-Kyber.

<table>
<thead>
<tr>
<th>n</th>
<th>k</th>
<th>q</th>
<th>η_1</th>
<th>η_2</th>
<th>(d_u, d_v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KYBER512</td>
<td>256</td>
<td>2</td>
<td>3329</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>KYBER768</td>
<td>256</td>
<td>3</td>
<td>3329</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>KYBER1024</td>
<td>256</td>
<td>4</td>
<td>3329</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

individually and concatenates the output byte arrays. The Compress_e(x, d) and Decompress_e(x, d) functions are defined as follows:

\[
\text{Compress}_e(x, d) = \left\lfloor \frac{2^d / q}{x} \right\rfloor \mod + 2^d,
\]

\[
\text{Decompress}_e(x, d) = \left\lfloor \frac{q}{2^d} \right\rfloor x.
\]

The functions \( \mathcal{G} \) and \( \mathcal{H} \) represent the SHA3-512 and SHA3-256 hash functions, respectively. The KDF represents key derivation function. It is realized by SHAKE-256.

There are three different parameter sets for CRYSTALS-Kyber, called KYBER512, KYBER768 and KYBER1024, see Table I. In this paper, we focus on KYBER768 with the security level \( k = 3 \).

IV. Experimental setup

This section describes the equipment we use for trace acquisition and the FPGA implementation of CRYSTALS-Kyber from [12].

A. Equipment

Our equipment is shown in Fig. 3. It consists of the ChipWhisperer-Lite board and the CW305 Artix-7 FPGA target board.

The ChipWhisperer toolkits include hardware and opensource software for security analysis based on a low-cost hardware platform [28]. The ChipWhisperer-Lite board is integrated with high-speed power measurement and programmer for the target device. The board is capable of capturing power traces of the target device synchronously, and controls the communication between the target device and the computer. A shunt resistor is placed between the power supply and the target device for power measurement. The ChipWhisperer-Lite board has a maximum sampling rate of 105 MS/sec and a buffer size of 24,400 samples.

The target board CW305 contains an Artix-7 XC7A100T FPGA.

B. Target FPGA implementation

Our target is the FPGA implementation of CRYSTALS-Kyber presented in [12]. It does not contain any countermeasures against side-channel attacks.

The implementation is described in Verilog. Its architecture consists of a client module, which is responsible for encryption, and a server module which is responsible for key...
For each byte $i \in \{0,1,\ldots,31\}$, we partition $\mathcal{T}$ into two sets, $\mathcal{T}_0$ and $\mathcal{T}_1$ as:

$$\mathcal{T}_0 = \{ T_j \in \mathcal{T} \mid m_j[i] < 128 \}$$
$$\mathcal{T}_1 = \{ T_j \in \mathcal{T} \mid m_j[i] > 128 \},$$

where $m_j[i]$ is the $i$th byte of $m_j$, for all $j \in \{1,2,\ldots,|\mathcal{T}|\}$. The t-test determines if there is a noticeable difference in the means of $\mathcal{T}_0$ and $\mathcal{T}_1$ by computing:

$$t = \frac{\bar{\mu}_0 - \bar{\mu}_1}{\sqrt{\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}}},$$

where $\mu_0$, $\mu_1$, and $n_i$ are the mean, standard deviation and cardinality of $\mathcal{T}_k$ for $k \in \{0,1\}$.

Fig. 4(c) shows the t-test results. One can clearly see 32 groups corresponding to the processing of 32 message bytes. Fig. 4(d) gives a zoomed-in view of 8 bytes. We can see two groups of t-test peaks for every byte, separated by a space. Each byte, with the exception of the ones in the beginning and at the end, overlaps with two adjacent bytes. This overlap could mean that several computations involving different message bytes are carried out in parallel. We will discuss this issue in more details in Section VI.

C. Training set expansion by cut-and-join

From Fig. 4(d) we can see that the shape of t-test peaks for different bytes is similar. Hence, we can increase the size of the profiling set by a factor of 32 (the number of bytes) using the cut-and-join technique from [17] without having to capture 32 times as many traces. For each trace $T_j \in \mathcal{T}$ and each byte $i \in \{0,1,\ldots,31\}$, we cut an interval of $T_j$ including all t-test peaks for the byte $i$. The union of the resulting intervals gives us the expanded profiling set. In our experiments, we use 64-point intervals.

Fig. 5 shows the t-test results after cut-and-joining profiling traces. One can see that the peaks of different bytes are perfectly aligned. In the figure we show only selected bytes in order not to make it too crowded, but the perfect alignment holds for other bytes as well.

D. Trace pre-processing

As in many other side-channel attacks, we pre-process traces by standardization (or variance scaling [30]) to smooth the shifting along y-axis of power measurements.

Given a set of traces $\mathcal{T}$ with elements $\mathcal{T} = (\tau_1, \ldots, \tau_{|\mathcal{T}|})$, each trace $\tau \in \mathcal{T}$ is standardized to $\tau' = (\tau'_1, \ldots, \tau'_{|\mathcal{T}|})$ such that, for all $i \in \{1,\ldots,|\mathcal{T}|\}$:

$$\tau'_i = \frac{\tau_i - \mu_i}{\sigma_i},$$

where $\mu_i$ and $\sigma_i$ are the mean and standard deviation of the traces of $\mathcal{T}$ at the $i$th data point.
TABLE II: MLP architecture used for message recovery.

<table>
<thead>
<tr>
<th>Layer type</th>
<th>Output shape</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Normalization 1</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>Dense 1</td>
<td>64</td>
<td>4160</td>
</tr>
<tr>
<td>Batch Normalization 2</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>ReLU</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>Dense 2</td>
<td>256</td>
<td>16640</td>
</tr>
<tr>
<td>Softmax</td>
<td>256</td>
<td>0</td>
</tr>
</tbody>
</table>

Total parameters: 21312
Trainable parameters: 21056

E. Network architecture and training parameters

We use multilayer perceptron (MLP) neural networks with the architecture shown in Table II.

The neural networks are trained with a batch size of 1024 for a maximum of 100 epochs using early stopping with patience 20. We use Nadam optimizer with a learning rate of 0.01 and a numerical stability constant epsilon = 1e-08. Categorical cross-entropy is used as a loss function to evaluate the network classification error. 70% of the training set is used for training and 30% is left for validation. Only the model with the highest validation accuracy is saved.

VI. ATTACK STAGE

In this section, we first describe the original multi-bit error injection method proposed in [10] for the attacks on software implementations of lattice-based PKE/KEMs and then explain how we adopted it to recover messages from a hardware implementation. We also show how session keys can be used to derive from the recovered messages.

A. Multi-bit error injection

Let \( m = (m[0], m[1], \ldots, m[31]) \) be a message to be recovered, where \( m[i] \) is the \( i \)th message byte for \( i \in \{0, 1, \ldots, 31\} \), and \( c = (u, v) \) be the corresponding ciphertext generated by KYBER.CPAPKE.Enc() for \( m \). It is known that, by subtracting the center of the integer ring \( \mathbb{Z}_q \) from the \( j \)th coefficient of \( v \), the \( j \)th bit of \( m \) can be flipped without changing other bits [15]. Using this property, the authors of [10] extend the attack set for message recovery from a single trace to 256 traces by injecting all possible multi-bit errors to the message bytes as follows.

For \( e \in \{0, 1, 2, \ldots, 255\} \), the original ciphertext \( c \) is modified to \( c_e \) such that CPAPKE.Dec() decrypts \( c_e \) to \( m_e = (m[0] \oplus e, m[1] \oplus e, \ldots, m[31] \oplus e) \).

This means that, all bits of \( m[i] \) in which the 8-bit binary expansion of \( e \) that has the value "1" are flipped. Then, the prediction for each \( m[i] \oplus e \) for all possible 256 values of \( e \) is made and the result is XORed with \( e \) to get \( m[i] \).

In this way, the multi-bit error injection method converts a non-differential side-channel attack to a differential one. This makes good predictions possible, with not-so-perfect deep learning models, biased towards some labels, since at least one of the traces in the extended attack set has the ground truth label to which the models are biased towards.

B. Sliced multi-bit error injection

In [10], the multi-bit error injection attack method is applied to a software implementation of Saber in which the message bytes are processed sequentially. However, hardware implementations carry out computations in parallel. From Fig. 4, we can see that the t-test peaks of the adjacent bytes are overlapping. If the same error \( e \) is injected into all bytes, as in [10], it may be cancelled out, reducing the attack efficiency.

We propose to overcome this problem by using the following technique which we call slicing. Instead of injecting an error \( e \) into all message bytes, we inject \( e \) into every fourth byte. As a result, the attack set is expanded into a \((256 \times 4)\)-trace set captured for the ciphertexts \( c_e \) decrypting to messages \( m_e \) whose bytes \( m_e[i] \) are defined by:

\[
m_e[i] = \begin{cases} m[i] \oplus e & \text{if } i \mod 4 = x, \\ m[i] & \text{otherwise,} \end{cases}
\]

where \( x \in \{0, 1, 2, 3\} \).

Clearly, the number of slices may be different from four. For implementations with a higher overlapping between the byte’s computations, a larger number may be more suitable.

Fig. 4 illustrates the difference between error injection in every byte and in every fourth byte. Each rectangular block represents the computation of some byte value. If the same \( e \) is injected into every byte, it may be cancelled out due to the overlapping computation of adjacent bytes. However, if \( e \) is injected into every fourth byte, the bytes with injected error do not affect each other.

C. Attack trace acquisition

To recover the message \( m \) from a properly generated ciphertext \( c \), we collect the attack dataset as follows. For each \( x \in \{0, 1, 2, 3\} \), we construct 256 ciphertexts \( c_e \) decrypting to messages \( m_e \) defined by (1) and capture a set of 256 traces \( T_A \) during the decapsulation of these ciphertexts.

Eight message bytes \( m[i] \) such that \( i \mod 4 = x \) are recovered from \( T_A \) for each \( x \) by giving the segments of traces \( T_e \) representing the processing of \( m_e[i] = m[i] \oplus e \) as input to the MLP model \( N \) trained at the profiling stage. For each \( T_e \), the model \( N \) outputs a score vector \( S_{i,e} = N(T_e) \) in which the value of the \( l \)th element, \( S_{i,e}[l] \), is the probability that \( m_e[i] = l \), for \( l \in \{0, \ldots, 255\} \).

The most likely label for \( m[i] \) among 256 candidates is decided as:

\[
l = \arg \max_{i \in \{0,1,\ldots,255\}} \prod_{e=0}^{255} S_{i,e}[l \oplus e].
\]

If \( l = m[i] \), the classification is successful. The condition \( l = m[i] \) can be verified by checking if the rank of \( m[i] \) is zero.

\(^1\) A rank of a byte \( b \) is the number of other bytes which have the probability higher than \( b \) in the score vector.
D. Session key recovery

A successful recovery of the message $m$ from a properly generated ciphertext $c$ trivially implies the session key recovery, since the session key can be derived as $K = \text{KDF}(\hat{K}, H(c))$ where $(\hat{K}, r) = G(m, H(pk))$ (see lines 3 and 5 of KYBER.CCAKEM.Encaps()).

VII. EXPERIMENTAL RESULTS

This section summarises our experimental results. Following the profiling strategy in Section V we captured 200K training traces during the execution of KYBER.CCAKEM.Decaps() procedure with input ciphertexts encrypting random messages. The training set was then expanded to 6.4M using the cut-and-join technique. An MLP model with the architecture listed in Table II was trained. In Section VII-B we describe how model analysis helped us to optimize the model.

A. Message/session key recovery attack

In this section, we present the results of message recovery attacks. We compare the proposed sliced multi-bit error injection method to the original multi-bit error injection method and to a direct repetition attack. All attacks are performed on the same 10 messages selected at random. Table III shows the time it takes to capture the test set for one message in each case.

1) Sliced multi-bit error injection attack: For each message, we captured $256 \times 4$ traces for the 4-sliced multi-bit error injection attack. Each measurement was repeated $N = 5$ times.

Table IV summarizes the results of message recovery. It shows the average and the maximum ranks of all message bytes for each message. We can see that all maximum ranks are smaller than 4. This means that we can recover the messages by enumerating the bytes with ranks 0, 1, 2 and 3, i.e. using up to $4^32 = 2^64$ enumerations in total, which is feasible.

Fig. 7 shows how the enumeration complexity varies for each message as we increase the number of traces. The y-axis
Fig. 6: An illustration of how the injected errors might be cancelled out. The blocks represent computations executed in parallel.

### TABLE III: Time for capturing the training and test sets.

<table>
<thead>
<tr>
<th>Attack method</th>
<th># Traces</th>
<th>Time (min)</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>repetition</td>
<td>256 x 5</td>
<td>5:40</td>
<td>0%</td>
</tr>
<tr>
<td>Test set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi-bit error injection w/o slicing</td>
<td>256 x 5</td>
<td>5:40</td>
<td>10%</td>
</tr>
<tr>
<td>multi-bit error injection w/o slicing</td>
<td>256 x 20</td>
<td>13:21</td>
<td>10%</td>
</tr>
<tr>
<td>4-sliced multi-bit error injection</td>
<td>256 x 4 x 5</td>
<td>22:42</td>
<td>100%</td>
</tr>
<tr>
<td>Training set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 hours to capture 200K traces</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7: Enumeration complexity as a function of the number of traces in the 4-sliced multi-bit error injection attack.

represents the number of enumerations required for message recovery in $\log_2$ scale. The red dashed line marks the threshold $64 = \log_2(2^{64})$ which we consider feasible. We can see that, as the number of traces increases, the enumeration complexity first sharply decreases, and then flattens. Further increase of the number of repetitions $N$ typically does not affect the ranks significantly in our experiments.

2) Multi-bit error injection attack: To compare the presented 4-sliced method to the original multi-bit error injection method, we also captured 256 traces in which the error $e$ is injected in all message bytes. We tested two cases with a different number of repetitions: $N = 5$ and $N = 20$. The case of $N = 20$ is included because the total number of traces used by the original multi-bit error injection method with $N = 20$ is equal to the total the total number of traces used by the 4-sliced multi-bit error injection method with $N = 5$.

Table [V] summarizes the results of message recovery. We can see that, for both $N = 5$ and $N = 20$ cases, only one message can be recovered with enumeration up to $2^{64}$, message 2.

Fig. 8 shows how the enumeration complexity changes as the number of traces grows for the case of $N = 5$. We can see that the curves flatten quite quickly. Therefore, further increase of $N$ from from 5 to 20 does not affect the ranks significantly.

This experiment shows that slicing is essential for an attack on the hardware implementation of CRYSTALS-Kyber from [7] in which the message bytes are processed in parallel. Without slicing, only 10% of messages can be recovered with enumeration up to $2^{64}$. With slicing, we can recover all messages with the same enumeration.

3) Repetition attack: To further highlight the effectiveness of the presented sliced multi-bit error injection method, we also compared it to a direct repetition attack which uses multiple traces captured for the same ciphertext $c$.

Table [VI] shows the results of message recovery from $256 \times 5$ traces. Clearly, these ranks are not feasibly enumerable. We did not attempted to increase to the number of test traces further because it was obvious that the repetition attack does not work on the target implementation.
TABLE IV: Results of the 4-sliced multi-bit error injection attack using $256 \times 4 \times 5$ traces.

<table>
<thead>
<tr>
<th>Error injected into every 4th byte</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. byte rank</td>
<td>0.09 0.19 0.09 0.09 0.03 0.19 0.06 0.22 0.28 0.16</td>
</tr>
<tr>
<td>Max. byte rank</td>
<td>2 2 1 1 1 2 1 2 3 3</td>
</tr>
</tbody>
</table>

All messages can be recovered with enumeration $\leq 2^{64}$.

TABLE V: Results of the multi-bit error injection attack without slicing.

<table>
<thead>
<tr>
<th>Error injected into all bytes</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. byte rank</td>
<td>1.63 2.34 0.34 0.84 0.72 1.59 1.93 1.88 2.28 1.09</td>
</tr>
<tr>
<td>Max. byte rank</td>
<td>16 15 24 2 13 2 5 23 25 24 26 8</td>
</tr>
</tbody>
</table>

$256 \times 5$ traces

| Avg. byte rank | 1.56 1.94 0.31 0.69 0.78 1.22 1.88 1.88 2.16 1.06 |
| Max. byte rank | 16 19 2 11 6 15 20 23 22 9 |

1 out of 10 messages can be recovered with enumeration $\leq 2^{64}$.

1 out of 10 messages can be recovered with enumeration $\leq 2^{64}$.

TABLE VI: Results of the repetition attack using $256 \times 5$ traces.

<table>
<thead>
<tr>
<th>Repetition attack</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. byte rank</td>
<td>116 123 127 120 121 99 130 105 138 116</td>
</tr>
<tr>
<td>Max. byte rank</td>
<td>251 237 247 248 253 240 255 248 252 255</td>
</tr>
</tbody>
</table>

No messages can be recovered with enumeration $\leq 2^{64}$.

Fig. 8: Enumeration complexity as a function of the number of traces in the multi-bit error injection attack without slicing.

B. Feature and model analysis

Explaining how deep learning models make decisions is important because it may help identify and patch vulnerabilities in the implementation under attack. It may also help in optimizing the model.

1) Feature analysis: To evaluate the importance of different input features for the model, we use two techniques: (1) weight analysis, and (2) stuck-at-0 fault injection. Both techniques have been shown useful in previous deep learning-based side-channel attacks of lattice-based PKE/KEMs [31].

Fig. 9 shows a plot representing the weights of the input Batch Normalization layer of the MLP used in the experiments. The Batch Normalization first standardizes the input values $X$ of the layer using their respective mean, $\mu$, and standard deviation, $\sigma$, $X_{\text{norm}} = (X - \mu) / \sigma$, and then applies the scaling, $\gamma$ (gamma) and offset, $\beta$ (beta), parameters to the result, $X' = (\gamma \ast X_{\text{norm}}) + \beta$. The parameters $\gamma$ and $\beta$ are learned by the model during the training process, i.e. the backpropagation algorithm is adjusted to operate on the transformed inputs, and error is used to update the new scaling and offset parameters learned by the model. Thus, a higher value of $\gamma$ indicates the higher importance of the corresponding input point in the decision taken by the model.

We can see that there are two clusters of peaks - one around the point 50 and another - around the point 25. We can relate these clusters to the peaks of the t-test shown in Fig. 5.

The relation becomes even more apparent after the stuck-at-0 fault injection analysis. Fig. 10 shows how the prediction accuracy of the model is affected by setting all but 9 consecutive points of a test trace to 0 before making inference (implying that the model takes its decision based on these 9 points only). If the rank remains close to 0 after the stuck-at-0 faults are injected into all points except for $\{p - 4, \ldots, p + 4\}$ (mod 64), the point $p$ is important.

In Fig. 10 we can clearly see two “valleys” around the points 50 and 25. They show that the most input features for the model’s decision are contained in these intervals. This, in turn, implies that the computations performed by the implementation of CRYSTALS-Kyber during the corre-
Fig. 9: The $\gamma$ parameters of the input Batch Normalization layer. Greater $\gamma$ indicates a more important data point.

Fig. 10: The effect of stuck-at-0 faults on the rank. For each data point $p \in \{0, \ldots, 63\}$, all points except $\{p-4, \ldots, p+4\} \pmod{64}$ are set to 0 and inference is performed on the modified data. If the rank remains close to 0, $p$ is important.

2) Model analysis: In early experiments, we used MLPs with the architecture similar to the one in Table II but with 6 dense layers instead of two. We were expecting models for breaking a hardware implementation of CRYSTALS-Kyber to be more complex than the ones used in the attacks on its software implementations. For example, the MLPs used in [10] for recovering message bytes from an ARM Cortex-M4 CPU implementation of CRYSTALS-Kyber has 3 dense layers.

After multiple training attempts, our best model was able to recover 70% of messages with the enumeration less than $2^{64}$ (see the first line of Table VII). We used the 4-slice multi-bit error injection method for the attack, with 5 repetitions, as in the experiments for Table IV.

To check if the model is optimal, we first dropped dense layers one-by-one and re-trained. The results are summarised in Table VII. Surprisingly, the model with two dense layers had approximately the same accuracy as the models with more layers. It could still recover 70% of messages with the enumeration less than $2^{64}$ and had the maximum rank of 6 (see the last line of Table VII).

We continued verifying optimality of the model with two dense layers by incrementally halving the number of neurons in the layer. For each case, we trained three models using the same training set and hyperparameters. Table VIII shows the results for the best of the three cases. We can see that the model with 64 neurons outperforms the others. We selected this model as optimal for the rest of experiments.

We also tried joining multiple models into an ensemble, but this did not improve the prediction accuracy further. Probably the models make dependent errors. We can see from the columns of Table VIII that some messages are easy and some are difficult for SCA. We do not yet have a confirmed explanation of why this is the case. Traces for all messages are captured under the same conditions, so it is unlikely that measurements are to blame. Since there is an overlapping in message byte processing (see Fig. 3 bottom), different combinations of the bytes processed in parallel sum up to different 8-bit words. It is possible that the difficulty of extracting a given byte from this word by SCA depends on other bytes in the word. A larger set of experiments is required to verify this hypothesis.

### TABLE VII: The impact of the number of dense layers on the maximum rank of MLPs. Each dense layer has 256 neurons.

<table>
<thead>
<tr>
<th># Layers</th>
<th>Message</th>
<th>Avg</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4 2 1 1 1 1 2 1 2 6 5</td>
<td>2.5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>2 2 1 1 1 1 2 1 2 9 5</td>
<td>2.6</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>2 2 1 1 1 3 1 4 6 2</td>
<td>2.3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2 2 1 1 2 1 2 1 3 6 6</td>
<td>2.6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2 2 1 1 2 1 1 2 4 6 5</td>
<td>2.6</td>
<td>6</td>
</tr>
</tbody>
</table>

### TABLE VIII: The impact of the layer size on the maximum rank of MLPs with two dense layers.

<table>
<thead>
<tr>
<th># Neurons</th>
<th>Message</th>
<th>Avg</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2 2 1 2 1 2 1 4 6 5</td>
<td>2.6</td>
<td>6</td>
</tr>
<tr>
<td>128</td>
<td>2 2 1 2 1 2 1 3 6 6</td>
<td>2.6</td>
<td>6</td>
</tr>
<tr>
<td>64</td>
<td>2 2 1 1 1 2 1 2 3 3</td>
<td>1.8</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>2 2 1 1 2 1 2 2 6 6</td>
<td>2.5</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>2 2 1 2 1 2 1 2 8 7</td>
<td>2.8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>5 2 1 2 2 2 1 3 8 5</td>
<td>3.1</td>
<td>8</td>
</tr>
</tbody>
</table>
VIII. CONCLUSION

We demonstrated a message recovery attack on a hardware implementation of CRYSTALS-Kyber by deep learning-based power analysis. The success of the attack is due to the sliced multi-bit error injection method which we introduced in this paper. It is an extension of the original multi-bit error injection method presented in [10] for side-channel analysis of software implementations of lattice-based PKE/KEMs. Software implementations do not require slicing because they execute instructions sequentially. However, our experimental results show that it is essential for hardware implementations. Without slicing, only 10% of messages can be recovered with enumeration up to $2^{64}$. With slicing, we can recover all messages with the same enumeration.

Future work includes re-designing the CRYSTALS-Kyber implementation to make it resistant to side-channel attacks.

IX. ACKNOWLEDGMENTS

This work was supported in part by the Swedish Civil Contingencies Agency (Grant No. 2020-11632), the Swedish Research Council (Grant No. 2018-04482) and the Sweden’s Innovation Agency Vinnova (Grant No. 2021-02426).

REFERENCES