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 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 [24] a power-based key recovery attack on the McEliece cryptosystem, a round 4 candidate of the NIST PQC, implemented in Xilinx Artix-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 Z_q denote the ring of integers modulo a prime q, and R_q denote the quotient ring $Z_q[X]/(X^n + 1)$. 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 $x \leftarrow (S; r)$ denotes sampling x from a distribution over a set S using seed r. The uniform distribution is denoted by U. The centered binomial distribution with parameter η is denoted by η.

The Decode_l function is designed to decode an array of $32l$ bytes into a 256 length polynomial with each coefficient in $\{0, 1, \ldots, 2^l - 1\}$. The Encode_l function is the inverse of Decode_l, which encodes each polynomial coefficient

KYBER.CPAPKE.KeyGen()

1: $(\ ;\ ) \ U \ (f0;1g^{256})$
2: $A \ U \ (R_q^{k\ k};\ )$
3: $s;e \quad {}_1(R_q^{k\ 1};\ )$
4: $t = \text{Encode}_{12}(As + e)$
5: $s = \text{Encode}_{12}(s)$
6: return $(pk = (t;\ );sk = s)$

KYBER.CPAPKE.Dec$(s;c)$

1: $u = \text{Decompress}_q(\text{Decode}_{d_u}(c_1);d_u)$
2: $v = \text{Decompress}_q(\text{Decode}_{d_v}(c_2);d_v)$
3: $s = \text{Decode}_{12}(s)$
4: $m = \text{Encode}_1(\text{Compress}_q(v \quad s^T u;1))$
5: return m

KYBER.CPAPKE.Enc$(pk = (t;\ );m;r)$

1: $t = \text{Decode}_{12}(t)$
2: $A \ U \ (R_q^{k\ k};\ )$
3: $r \quad {}_1(R_q^{k\ 1};r)$
4: $e_1 \quad {}_2(R_q^{k\ 1};r);\ e_2 \quad {}_2(R_q^{1\ 1};r)$
5: $u = A^T r + e_1$
6: $v = t^T r + e_2 + \text{Decompress}_q(m;1)$
7: $c_1 = \text{Encode}_{d_u}(\text{Compress}_q(u;d_u)$
8: $c_2 = \text{Encode}_{d_v}(\text{Compress}_q(v;d_v)$
9: return $c = (c_1;c_2)$

Fig. 1: Description of KYBER.CPAPKE algorithm from [12].

KYBER.CCAKEM.KeyGen()

1: $z \ U \ (f0;1g^{256})$
2: $(pk;s) =$ KYBER.CPAPKE.KeyGen()
3: $sk = (s;pk;H(pk);z)$
4: return $(pk;sk)$

KYBER.CCAKEM.Encaps(pk)

1: $m \ U \ (f0;1g^{256})$
2: $m = H(m)$
3: $(\hat{K};r) = G(m;H(pk))$
4: $c =$ KYBER.CPAPKE.Enc$(pk;m;r)$
5: $K = \text{KDF}(\hat{K};H(c))$
6: return $(c;K)$

KYBER.CCAKEM.Decaps$(sk = (s;pk;H(pk);z),c)$

1: $m' =$ KYBER.CPAPKE.Dec$(s;c)$
2: $(\hat{K}';r') = G(m';H(pk))$
3: $c' =$ KYBER.CPAPKE.Enc$(pk;m';r')$
4: if $c = c'$ then
5: return $K = \text{KDF}(\hat{K}';H(c))$
6: else
7: return $K = \text{KDF}(z;H(c))$
8: end if

Fig. 2: Description of KYBER.CCAKEM algorithm from [12].

TABLE I: Parameters of CRYSTALS-Kyber.

	n	k	q	$_1$	$_2$	$(d_u;d_v)$
KYBER512	256	2	3329	3	2	(10, 4)
KYBER768	256	3	3329	2	2	(10, 4)
KYBER1024	256	4	3329	2	2	(11, 5)

individually and concatenates the output byte arrays. The $\text{Compress}_q(x;d)$ and $\text{Decompress}_q(x;d)$ functions are de ned as follows:

$$\text{Compress}_q(x;d) = d(2^d/q) \ x \bmod^+ 2^d;$$
$$\text{Decompress}_q(x;d) = d(q/2^d) \ x:$$

The functions G and 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 [7].

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 open-source 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 [7]. 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 encapsulation, and a server module which is responsible for key

Fig. 3: Equipment for trace acquisition.

generation and decapsulation. The implementation achieves good performance with limited resources. It can fit into the smallest Xilinx Artix-7 device. The implementation covers all three versions of CRYSTALS-Kyber.

We adapted the implementation to a CW305 target board and complemented it with a controller in Verilog which sends in keys and ciphertexts to the server and reads out session keys from the server. We synthesized the Kyber768 implementation to target a Xilinx XC7A100TFTG256 FPGA device using the Vivado synthesis toolchain.

V. Profiling stage

This section describes our profiling strategy.

A. Training trace acquisition

Using the equipment described in the previous section, we captured traces representing the execution of the decapsulation procedure KYBER.CCAKEM.Decaps(). For the profiling stage, the traces are captured for ciphertexts generated by KYBER.CPAPKE.Enc() for random messages. Since Kyber is a public-key cryptographic algorithm, we can create a properly generated ciphertext for any message using the corresponding public key. Thus, the device under attack can be used for creating a labeled dataset for profiling [17].

Fig. 4(a) shows a trace representing the execution of the complete KYBER.CCAKEM.Decaps() procedure.

B. Location of points of interest

Since, for each trace T_j of the profiling set T, the value of the message m_j processed by KYBER.CCAKEM.Decaps() during the acquisition of T_j is known, we can use Welch's t-test [29] to analyze the leakage and check if it is possible to locate intervals corresponding to the individual message bytes.

For each byte $i \in \{0, 1, \ldots, 31\}$, we partition T into two sets, T_0 and T_1 as:

$$T_0 = \{T_j \in T \mid m_j[i] < 128\}$$
$$T_1 = \{T_j \in T \mid m_j[i] > 128\},$$

where $m_j[i]$ is ith byte of m_j, for all $j \in \{1, 2, \ldots, |T|\}$. The t-test determines if there is a noticeable differences in the means of T_0 and T_1 by computing:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}}},$$

where μ_i, σ_i, and n_i are the mean, standard deviation and cardinality of 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 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 $T_j \in T$ 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 T with elements $T = (t_1, \ldots, t_{|T|})$, each trace $T \in T$ is standardized to $T' = (t'_1, \ldots, t'_{|T|})$ such that, for all $i \in \{1, \ldots, |T|\}$:

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

where μ_i and σ_i are the mean and standard deviation of the traces of T at the ith data point.

TABLE II: MLP architecture used for message recovery.

Layer type	Output shape	# Parameters
Batch Normalization 1	64	256
Dense 1	64	4160
Batch Normalization 2	64	256
ReLU 1	64	0
Dense 2	256	16640
Softmax	256	0
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 derived 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 ith 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 jth coefficient of v, the jth 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 implementations 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 (256×4)-trace set captured for the ciphertexts c_e decrypting to messages m_e whose bytes $m_e[i]$ are defined by:

$$m_e[i] = \begin{cases} m[i] \oplus e & \text{if } i \bmod 4 = x; \\ m[i] & \text{otherwise,} \end{cases} \quad (1)$$

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. 6 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 $i \pmod 4 = x$ 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 $\mathcal{N}$ trained at the profiling stage. For each T_e, the model $\mathcal{N}$ outputs a score vector $S_{i,e} = \mathcal{N}(T_e)$ in which the value of the lth element, $S_{i,e}[l]$, is the probability that $m_e[i] = l$, for $l, e \in \{0, \ldots, 255\}$.

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

$$\tilde{l} = \arg\max_{l \in \{0,1,\ldots,255\}} \left(\prod_{e=0}^{255} S_{i,e}[l \oplus e] \right).$$

If $\tilde{l} = m[i]$, the classification is successful. The condition $\tilde{l} = m[i]$ can be verified by checking if the rank of $m[i]$ is zero[1].

[1] A rank of a byte b is the number of other bytes which have the probability higher than b in the score vector.