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

Yanning Ji and Elena Dubrova

KTH Royal Institute of Technology, Stockholm, Sweden
yanning,dubrova@kth.se

Abstract. NIST has recently selected CRYSTALS-Kyber as a new public key encryption and key establishment algorithm to be standardized. This makes it important to evaluate the resistance of CRYSTALS-Kyber implementations to side-channel attacks. Software implementations of CRYSTALS-Kyber have already been thoroughly analysed. The discovered vulnerabilities helped improve the subsequently released versions and promoted stronger countermeasures against side-channel attacks. In this paper, we present the first attack on a protected hardware implementation of CRYSTALS-Kyber. We demonstrate a practical message (shared key) recovery attack on the first-order masked FPGA implementation of Kyber-512 by Kamucheka et al. (2022) using power analysis based on the Hamming distance leakage model. The presented attack exploits a vulnerability located in the masked message decoding procedure which is called during the decryption step of the decapsulation. The message recovery is performed using a profiled deep learning-based method which extracts the message directly, without extracting each share explicitly. By repeating the same decapsulation process multiple times, it is possible to increase the success rate of full shared key recovery to 99%.

Keywords: Public key cryptography · post-quantum cryptography· CRYSTALS-Kyber · LWE/LWR-based KEM · side-channel attack · deep learning

1 Introduction

CRYSTALS-Kyber is a key encapsulation mechanism (KEM) that is IND-CCA2-secure (indistinguishable under an adaptive chosen ciphertext attack) in the classical and quantum random oracle models [4]. The security of CRYSTALS-Kyber relies on the hardness of the module learning with errors (M-LWE) problem that comes from introducing unknown noise into linear equations.

In July 2022, the National Institute of Standards and Technology (NIST) has selected CRYSTALS-Kyber as a new public key encryption (PKE) and key establishment algorithm to be standardized [31]. Shortly after, the National Security Agency (NSA) has included CRYSTALS-Kyber in the suite of cryptographic algorithms recommended for national security systems [2]. This makes

it important to evaluate the resistance of CRYSTALS-Kyber to side-channel attacks on its implementations executed on physical devices. Pioneered by Paul Kocher [26, 27], side-channel attacks are considered a main security threat to cryptographic implementations at present. Physical devices tend to leak information through physically measurable, non-primary channels such as timing [27], power consumption [26], or electromagnetic (EM) radiation [3]. There is currently no physical device that is side-channel leakage-free. To combat side-channel attacks, countermeasures such as masking [13,19,35], shuffling [42], randomized clock [28], random delays insertion [14], constant-weight encoding [30], code polymorphism [8], etc. are used to protect cryptographic implementations.

However, in the past few years it has been shown that even protected software implementations of CRYSTALS-Kyber and other post-quantum PKE/KEM algorithms can be broken using advanced side-channel analysis methods based on deep learning. The shared and secret keys have been recovered from the first-order masked implementations [38, 41], higher-order masked implementations [16], and first-order masked and shuffled software implementations of CRYSTALS-Kyber [6]. The discovered vulnerabilities promoted stronger defense mechanisms against side-channel attacks, e.g. [5, 20, 40] and helped streng- then subsequent implementations of CRYSTALS-Kyber [10, 15].

Contributions: In this paper, we present the first attack on a protected hardware implementation of CRYSTALS-Kyber. To the best of our knowledge, only unprotected hardware implementations have been analysed until now [23, 24, 36, 41]. Hardware implementations are typically more difficult to break than software ones because they carry out computations in parallel and their underlying process technology usually uses a smaller size of the transistor's elements. Smaller transistors consume less power, making power analysis more challenging. However, side-channel attacks of hardware implementation may potentially exploit glitches which occur if the processing of shares is not well-synchronized [39].

We demonstrate a practical message recovery attack on the first-order masked FPGA implementation of Kyber-512 by Kamucheka et al. [25] using profiled power analysis based on the Hamming distance leakage model. The presented attack exploits a vulnerability in the implementation which is located in the masked message decoding procedure which is called during the decryption step of the decapsulation algorithm. The message recovery is performed using the profiled deep learning-based method of Ngo et al. [34] which extracts the masked secret directly, without extracting each share explicitly. By repeating the same decapsulation multiple times, it is possible to increase the success rate of full shared key recovery to 99%. As observed in [16], side-channel attacks of masked implementations benefit from the fact that errors in repeated measurements are more independent due to fresh random masks which are re-generated for each execution.

In CRYSTALS-Kyber, a successful message recovery implies the shared key recovery, as 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. [6, 38], one can extract the long-term secret key of CRYSTALS-Kyber.

The rest of this paper is organized as follows. Section 2 reviews previous work on side-channel analysis of CRYSTALS-Kyber implementations. Section 3 gives background necessary for understanding the paper. Section 4 describes assumptions on the adversary model, key establishment protocol, and attack scenario. Sections 5 and 6 present the target implementation and equipment used for trace acquisition, respectively. Sections 7 describes the neural network training strategy. Section 8 summarizes experimental results. Section 9 suggests potential countermeasures. Section 10 concludes the paper.

2 Previous Work

This section describes previous work on unprotected and protected hardware implementations of CRYSTALS-Kyber and the related side-channel attacks on unprotected hardware implementation. To the best of our knowledge, only unprotected hardware implementations of CRYSTALS-Kyber have been analysed until now.

2.1 Hardware implementations of CRYSTALS-Kyber

Huang et al. [21] developed an unprotected FPGA implementation that consumes 88,901 look-up tables (LUTs) and takes 68,815 clock cycles to execute the decapsulation algorithm of Kyber-512. This is 10 times fewer clock cycles than the ARM Cortex-M4 implementation of CRYSTALS-Kyber presented in [11].

Xing et al. [45] created an even smaller and faster unprotected FPGA implementation of CRYSTALS-Kyber, intended for resource-constrained devices. With suitably designed pipelines and highly-optimized architecture, the implementation of [45] executes the decapsulation algorihtm of any version of Kyber within 14,000 clock cycles using only 7,500 LUTs.

Kamucheka et al. [25] presented a first-order masked FPGA implementation of Kyber-512 built on the top of the implementation of Huang et al. [21]. The masking method adds 70% of area and 100% of clock cycle counts on the top of the unprotected implementation. A less secure version of Kyber-512, protected by hiding, with 60% area overhead and 80% clock cycle count overhead is also described in [25].

Jati et al. [22] developed a hardware implementation of CRYSTALS-Kyber protected from side-channel analysis using random delay insertion and address and instruction shuffling. The countermeasures add 5% area overhead at the expense of performance overhead.

2.2 Side-channel attacks on CRYSTALS-Kyber implementations

In [24] a power analysis of a hardware implementation of Kyber-512 from [21] in Xilinx Virtex-7 FPGA is presented. Some leakage points are discovered using a t-test.

In [36], a side-channel attack of Kyber-512 implemented in a Xilinx Artix 7 FPGA based on correlation near-field EM analysis is presented. The attack targets polynomial multiplication during the decryption step of the decapsulation algorithm. To recover the secret key, the attack requires 166,620 traces and knowledge of the register reference values.

In [23], a profiled side-channel attack on a Xilinx Artix-7 FPGA implementation of Kyber-768 from [45] is presented. The attack can recover messages of Kyber-768 from 5,120 traces using deep learning-based power analysis and enumeration of up to 2^{64}.

In [29], side-channel attacks on two implementations of the decryption algorithm of Kyber-768 in Spartan-6 FPGAs (standalone), the one of Xing et al. [45] and an implementation designed by the authors, based on correlation analysis are presented. The attacks can recover the secret key from 27 power traces, or from 60 EM traces.

3 Background

In this section, we describe notation used in the paper and give background on CRYSTALS-Kyber algorithm [4], masking countermeasure against side-channel attacks, and Welch's t-test.

3.1 Notation

Let $\mathbb{Z}_q$ be the ring of integers modulo a prime q and R_q be the quotient ring $\mathbb{Z}_q[X]/(X^n+1)$. We use regular font letters to denote elements of R_q, bold lower-case letters to represent vectors with coefficients in R_q, and bold upper-case letters to represent 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 $x \leftarrow \mathcal{D}(S;r)$ is used for sampling x from a probability distribution $\mathcal{D}$ over a set S using seed r. The uniform distribution is denoted by $\mathcal{U}$. The centered binomial distribution with parameter μ is denoted by B_μ.

3.2 CRYSTALS-Kyber algorithm specification

CRYSTALS-Kyber [4] consists of a CCA-secure KEM scheme,
KYBER.CCAKEM which is buit on the top of a chosen plaintext attack (CPA)-secure PKE scheme, KYBER.CPAPKE using a tweaked version of the Fujisaki-Okamoto (FO) transform [17], see Figs. 1 and 2.

CRYSTALS-Kyber uses vectors of ring elements in R_q^k, where k is the rank of the module defining the security 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 parameters. In this paper, we focus on Kyber-512 because this version is realized in the target implementation of [25].

CRYSTALS-Kyber employs the number-theoretic transform (NTT) to perform multiplications in R_q efficiently. The details of NTT are not included in Fig. 1 and Fig. 2 in order to simplify the pseudocode.

KYBER.CPAPKE.KeyGen()

1: $(\rho, \sigma) \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k \times k}; \rho)$
3: $\mathbf{s}, \mathbf{e} \leftarrow \beta_{\eta_1}(R_q^{k \times 1}; \sigma)$
4: $\mathbf{t} = \mathsf{Encode}_{12}(\mathbf{As} + \mathbf{e})$
5: $\mathbf{s} = \mathsf{Encode}_{12}(\mathbf{s})$
6: **return** $(pk = (\mathbf{t}, \rho), sk = \mathbf{s})$

KYBER.CPAPKE.Dec$(\mathbf{s}, c)$

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

KYBER.CPAPKE.Enc$(pk = (\mathbf{t}, \rho), m, r)$

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

Fig. 1: Description of KYBER.CPAPKE algorithms from [4] (simplified).

Inputs and outputs to all API functions of CRYSTALS-Kyber are byte arrays. Functions performing the conversion of byte arrays to individual elements and vice versa are called Decode_l and Encode_l, respectively. 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 individually and then concatenates the 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 defined by:

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

where the term $\lfloor a \rceil$ stands for rounding of a to the closest integer with ties being rounded up.

If $\mathsf{Compress}_q$ 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. The compression enables the removal of some low-order bits of the ciphertext without affecting the correctness probability of decryption significantly.

KYBER.CCAKEM.KeyGen()

1: $z \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $(pk, \mathbf{s}) = \text{KYBER.CPAPKE.KeyGen}()$
3: $sk = (\mathbf{s}, pk, \mathcal{H}(pk), z)$
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: $c = \text{KYBER.CPAPKE.Enc}(pk, m, r)$
5: $K = \mathsf{KDF}(\hat{K}, \mathcal{H}(c))$
6: **return** (c, K)

KYBER.CCAKEM.Decaps($sk = (\mathbf{s}, pk, \mathcal{H}(pk), z)$,c)

1: $m' = \text{KYBER.CPAPKE.Dec}(\mathbf{s}, c)$
2: $(\hat{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
3: $c' = \text{KYBER.CPAPKE.Enc}(pk, m', r')$
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: Description of KYBER.CCAKEM algorithms from [4] (simplified).

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)

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

3.3 Masking

Masking is a popular countermeasure against power and EM side-channel analysis [13]. In a ω-order masking, a sensitive variable x is partitioned into $\omega + 1$ shares: $x_1, x_2, \ldots, x_{\omega+1}$ such that $x = x_1 \circ x_2 \circ \ldots \circ x_{\omega+1}$. The type of the operation "$\circ$" depends on the masking method, for instance, in arithmetic masking "$\circ$" is the arithmetic addition, while in Boolean masking it is the XOR.

By performing operations separately on shares, the computations do not involve x directly, thereby preventing leakage of side-channel information about

x in theory. The shares are randomized at each execution. Typically, the randomization is performed by assigning random masks $x_1, x_2, \ldots, x_\omega$ to ω shares and deriving the last share as $x - (x_1 + x_2 + \ldots + x_\omega)$ for arithmetic masking or $x \oplus x_1 \oplus x_2 \oplus \ldots \oplus x_\omega$ for Boolean masking.

3.4 Welch's t-test

Welch's t-test [44] is commonly used in the Test Vector Leakage Assessment (TVLA) [18] as a metric for evaluating side-channel leakage and as a tool for feature extraction from side-channel measurements.

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

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

where μ_i and σ_i are the mean and the standard deviation of $\boldsymbol{T}_i$ and $|\boldsymbol{T}_i|$ is the cardinality of $\boldsymbol{T}_i$, for $i \in \{0, 1\}$. The two sets are considered to be noticeably different if the absolute value of t-test result is greater than 4.5.

4 Assumptions

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

4.1 Adversary model

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

The adversary is a clever outsider who has expertise in deep learning-based side-channel analysis and the necessary equipment and tools. The goal of the adversary is to recover the shared key K from side-channel information acquired from the device which runs the CRYSTALS-Kyber's decapsulation algorithm, referred to as the device under attack (DUA) in the sequel.

We assume that the adversary has a physical access to the DUA to measure its power consumption and is capable to query the DUA with ciphertexts of his/her choice.

4.2 Key establishment protocol

We assume that the protocol illustrated in Fig. 3 is used to establish a shared key between two parties communicating over a public channel.

To initiate a new key establishment session, Party 1 uses the key generation algorithm KYBER.CCAKEM.KeyGen() to create a fresh key pair (pk, sk) and sends pk to the Party 2. The Party 2 applies the encapsulation algorithm

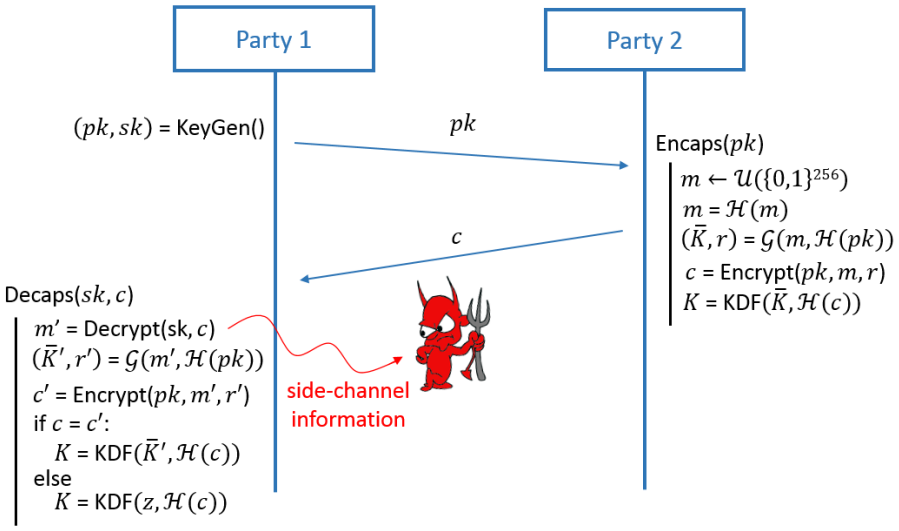

Fig. 3: A shared key establishment protocol.

KYBER.CCA- KEM.Encaps() to compute a ciphertext c encapsulating a shared key K and sends c to the Party 1. Finally, the Party 1 uses the decapsulation algorithm KYBER.CCAKEM.Decaps() to compute K from c.

We also assume that the decapsulation device of Party 1 accepts more than one query to decapsulate a ciphertext using the same secret key sk until a new key establishment session is initiated.

4.3 Attack scenario

At the profiling stage, the attacker first collects from the DUA a set of power traces captured during the execution of the decapsulation algorithm for ciphertexts encrypting messages known to the attacker. The attacker can generate such ciphertexts for any set of messages using the public key pk of the Party 1. Then, the attacker uses the collected traces to train a neural network $\mathcal{N}$ which models the leakage of the DUA.

At the attack stage, the adversary eavesdrops on the communication channel between the Party 1 and Party 2 to obtain the ciphertext c which encapsulates the shared key K. The adversary also measures the power consumption of the DUA during the execution of the decapsulation algorithm with c as input and records the resulting power trace T. The model $\mathcal{N}$ trained at the profiling stage is then used to extract the message m encrypted in c from the trace T.

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

Listing 1.1: A snippet of Verilog code implementing message decoding in [25]

```
module State_Polytomsg__masked_decode #(
  parameter COEFF_SZ = 16,
  ...
) (
  input                          clk,
  input                          rst_n,
  input                          ce,
  input  wire [COEFF_SZ-1:0]     c1,
  input  wire [COEFF_SZ-1:0]     c2,
  input  wire [COEFF_SZ-1:0]     PRNG_data,
  output reg                     data_valid,
  output reg                     m1,
  output reg                     m2
);
...
always @(posedge clk) begin
  data_valid <= w_P3_ready;
  m1 <= MSB(w_P3_y1);
  m2 <= MSB(w_P3_y2);
end
...
```

5 Target implementation

The presented attack is mounted on the first-order masked FPGA implementation of Kyber-512 by Kamucheka et al. [25]. The attack targets the message decoding operation at the decryption step of the decapsulation algorithm. In this section, we describe the implementation details of the message decoding module of [25].

The message decoding function of CRYSTALS-Kyber converts a polynomial f with n coefficients, $f[j]$, into an array of $n/8$ bytes representing a message m with n bits, $m[j]$ (see $\mathsf{Encode}_1(\mathsf{Compress}_q(v - \boldsymbol{s} \cdot \boldsymbol{u}, 1))$ at line 4 of KYBER.CPAPKE.Dec() in Fig. 1). If the value of $f[j]$ is closer to $q/2$ than to 0, then $f[j]$ is decoded to $m[j] = 1$. Otherwise, $f[j]$ is decoded to $m[j] = 0$.

In the implementation by Kamucheka et al. [25], the message decoding is realized by a module called `State_Polytomsg()` in which each polynomial coefficient is converted individually. Pipelining is applied to accelerate computations; it reduces the number of clock cycles required for message decoding from 6656 to 274. Algorithm 1.1 shows a snippet in the Verilog implementation of a submodule of `State_Polytomsg()` called `State_Polytomsg_ _masked_decode()`, which is related to the computations of message bits. At each clock cycle, `State_Polytomsg__masked_decode()` takes two arithmetic shares, $c1$ and $c2$, representing a polynomial coefficient $f[j] = (c1 + c2) \bmod q$ and computes

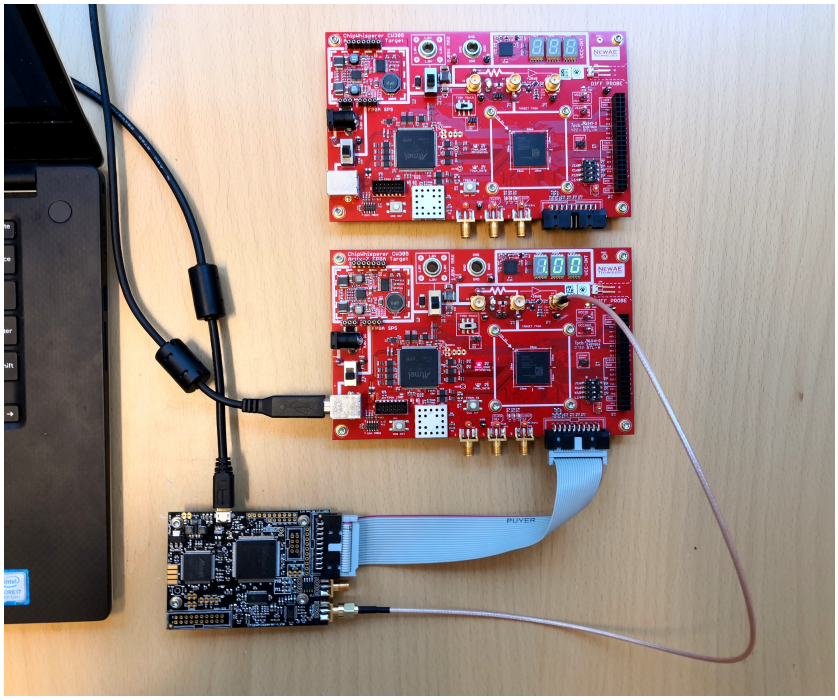

Fig. 4: Equipment for trace acquisition.

two Boolean shares, $m1$ and $m2$, representing the decoded message bit $m[j] = m1 \oplus m2$, where $\oplus$ is the Boolean XOR, and $+$ is the arithmetic addition.

6 Equipment

For trace acquisition, we use a ChipWhisperer-Lite board and two CW305 Artix FPGA target boards shown in Fig. 4. The target board CW305 contains an Artix-7 XC7A100T FPGA.

The ChipWhisperer-Lite captures power traces of the target device synchronously, with a maximum sampling rate of 105 MS/sec and a buffer size of 24,400 samples [1]. In our experiments, we capture four data points per clock cycle.

In the implementation of Kyber-512 by Kamucheka et al. [25], the decapsulation algorithm occupies 152,860 LUTs and 489.5 block random access memories (BRAMs), which exceeds the capacity of the Artix XC7A100T FPGA. Unfortunately, we do not have a larger FPGA in our possession[1]. To perform the experiments, we removed all modules which are not executed in parallel with

[1] The cost of an FPGA which can fit the implementation [25] is over 5,000 US$.

`State_Polytomsg()`. As simulation results presented in the Appendix show, this does not affect the total power consumption of the FPGA during the execution of `State_Polytomsg()`.

7 Neural network training strategy

As in the attack on a masked software implementation of CRYSTALS-Kyber presented in [16], we train a single universal neural network model on cut-and-joined and standardized traces. However, an essential difference from the method of [16] is that we create labels using the Hamming distance leakage model as opposed to the Hamming weight one. We are not aware of any side-channel attack on a PQC algorithm implementation using such a labeling strategy. We show that an attack based on the Hamming weight leakage model would not be effective on the implementation of [25].

7.1 Architecture

We use the neural networks architecture shown in Table 2. It is similar to the architecture used in the attacks on protected software implementations of CRYSTALS-Kyber, e.g. [6, 16], except for the input size and layer's width.

7.2 Training parameters

The MLPs are trained using a batch size of 1024 for a maximum of 300 epochs using early stopping with patience 20. We use Nadam optimizer with a learning rate of 0.01 and a numerical stability constant $epsilon = 1e\text{-}08$. Categorical cross-entropy is utilized 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.

7.3 Locating points of interest

To locate the points of interest, we train neural networks for individual message bits on full traces and examine the weights of the input Batch Normalization layer after training. Large weights reveal the points where the two Boolean shares corresponding to the bit are processed.

Fig. 5(a) shows a power trace representing the execution of the module `State_Polytomsg()` in the implementation of [25]. This trace is obtained by averaging 1,000 traces captured for ciphertexts encrypted messages selected at random.

Fig. 5(b) shows the weights of gamma parameter of input Batch Normalization layers of eight neural networks $\mathcal{N}_i$, trained on 10,000 full traces with $HD(m[i-1], m[i])$ as a label, for $i \in \{199, 207, 215 \ldots, 255\}$ (last bits of last eight bytes in the message, for demonstration). In the zoom-in segment shown in Fig. 5(c), one can clearly see regular high peaks on distance 32 points from each

Table 2: MLP architecture: The *input_size* is 1,500 for training on full traces and 32 for training on cut-and-joined traces.

Layer type	Output shape
Batch Normalization 1	*input_size*
Dense 1	32
Batch Normalization 2	32
ReLU 1	32
Dense 2	16
Batch Normalization 3	16
ReLU 2	16
Dense 3	8
Batch Normalization 4	8
ReLU 3	8
Dense 4	2
Softmax	2

other. These peaks correspond to the processing of individual bits of Boolean shares $m1$ and $m2$ in `State_Polytomsg__masked_decode()` module.

We also trained neural networks on 10,000 full traces with the Hamming weight of message bit $m[i]$, $HW(m[i])$, as a label. Fig. 5(d) shows the weights of of neural networks $\mathcal{N}_i$ in this case, for $i \in \{199, 207, 215 \ldots, 255\}$. We can see that there are no distinct peaks. Thus, the Hamming distance-based labels are more suitable than the Hamming weight-based labels in an attack on the implementation of [25].

7.4 Trace pre-processing

We cut each trace acquired from the DUA into 255 segments corresponding to the processing of the bit i of Boolean shares $m1$ and $m2$ by `State_Polytomsg_masked_decode()`, for all $i \in \{1, \ldots, 255\}$, and label the segments by the corresponding Hamming distances $HD(m[i-1], m[i])$, where $m = m1 \oplus m2$ is the message. The training set $\boldsymbol{T}$ is composed as a union of such segments from all traces.

Such a cut-and-join [34] technique allows us to train a single universal neural network model which is capable of predicting the Hamming distance between any two adjacent message bits. It also allows us to increase the size of the training set by a factor of 255 without having to acquire 255 times more traces from the DUA.

We also apply standardization to traces in $\boldsymbol{T}$. Each trace $T = (t_1, \ldots, t_{|T|}) \in \boldsymbol{T}$, is standardized to $T' = (t'_1, \ldots, t'_{|T|})$ such that:

$$t'_j = \frac{t_j - \mu_j}{\sigma_j},$$

where and μ_j and σ_j are the mean and the standard deviation of the elements of $\boldsymbol{T}$ at the jth data point, $j \in \{1, \ldots, |T|\}$.

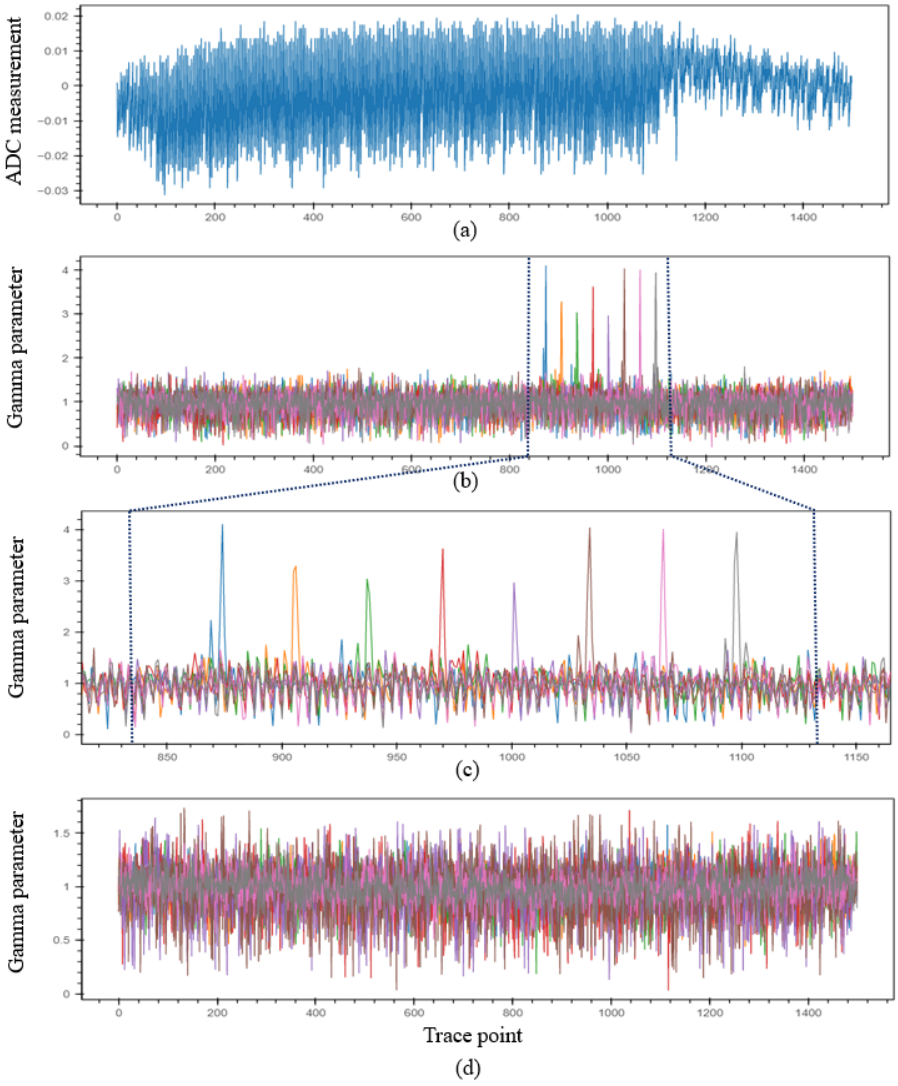

Fig. 5: (a) An average power trace representing the execution of `State_Polytomsg()` module in the implementation [25]; (b) weights of input Batch Normalization layer of eight neural networks $\mathcal{N}_i$, $i \in \{199, 207, 215 \ldots, 255\}$, trained on 10,000 full traces with $HD(m[i-1], m[i])$ as a label; (c) a zoomed-in view of (b); (d) weights of $\mathcal{N}_i$ trained with $HW(m[i])$ as a label.

8 Experiment results

In this section, we present the results of message recovery attack on the first-order masked FPGA implementation of Kyber-512 [12] by Kamucheka et al. [25].

The neural network models are trained and tested on a PC with an Intel Core i7-8750H CPU running an 2.2GHz and 16GB RAM.

8.1 Message recovery

At the profiling stage, we collect from the DUA 1,000 power traces to train neural networks. The traces are captured during the execution of the decapsulation algorithm KYBER.CCAKEM.Decaps() with input ciphertexts encrypting messages selected at random. Then, the trace set is expanded to 255,000 traces using the cut-and-join technique and labeled by the corresponding Hamming distance $HD(m[i-1], m[i])$, for $i \in \{1, \ldots, 255\}$. Finally, we use the resulting labeled dataset to train a universal MLP model, $\mathcal{N}$, with the architecture listed in Table 2 which can predict the Hamming distance between any the two adjacent message bits.

At the attack stage, we capture from the DUA test traces during the execution of KYBER.CCAKEM.Decaps() for input ciphertexts c encrypting 1,000 different messages m selected at random. For each message to be recovered, m', we use $\mathcal{N}$ to recover the Hamming distances $HD(m'[i], m'[i+1])$ for each $i \in \{1, \ldots, 255\}$. Then, we enumerate two possible values of $m'[0]$ and reconstruct m' for each case using the recovered Hamming distances. Finally, we verify which value of $m'[0]$ is the correct guess by encrypting m' using the public key of the DUA, pk, and comparing to the resulting ciphertext $c' =$ KYBER.CPAPKE.Enc(pk, m', r') to the ciphertext $c =$ KYBER.CPAPKE.Enc(pk, m, r) which is given as input when the trace is captured.

Table 3 shows the empirical probability to recover the Hamming distance $HD(m[i-1], m[i])$ from a single trace for each $i \in \{1, \ldots, 255\}$. We can see that this probability is too low, $p_{HD} = 0.61479$ on average. This gives us a negligible average probability of full message recovery, $p_{message} = (p_{HD})^{255} = 1.3 \times 10^{-54}$.

It may be worth noting that bits of the last byte in Table 3 have a higher p_{HD} on average. This is becuase the `Poly_tomsg()` module uses piplining for accelerating its operation. Thus, all message bits except the ones at the end of the pipeline are processed in parallel with several other bits. Note that the bits of each byte are processed in the order $7, 6, \ldots, 0$, e.g. bit 0 of byte 31 is processed last.

To improve the success rate, we repeat the attack using test traces captured with multiple repetitions of the same decapsulation and applying majority voting to the model's predictions. Tables 4 shows the results for up to 400 repetitions (the number of repetitions should be odd for majority voting). Table 5 also gives more detailed results for the probability to recover the Hamming distance $HD(m[i-1], m[i])$ from 299 trace for each $i \in \{1, \ldots, 255\}$.

We can see that, for 299 repetitions, the message recovery probability is 97.7%. For 399 repetitions, it is possible to recover messages with the probabil-

Table 3: Empirical probability (mean over 1K tests) to recover $HD(m[i-1], m[i])$ of a message bit $i = 8a + b$ from a single trace, for $i \in \{1, \ldots, 255\}$.

Byte, a	Bit position in a byte, b								**Avg.**
	0	1	2	3	4	5	6	7	
0	/	0.644	0.615	0.614	0.604	0.610	0.633	0.597	0.61671
1	0.613	0.631	0.624	0.621	0.616	0.630	0.622	0.606	0.62038
2	0.589	0.611	0.587	0.600	0.601	0.596	0.622	0.625	0.60388
3	0.627	0.605	0.622	0.614	0.607	0.624	0.622	0.610	0.61638
4	0.613	0.599	0.615	0.615	0.621	0.628	0.630	0.620	0.61763
5	0.613	0.634	0.606	0.604	0.602	0.632	0.635	0.642	0.62100
6	0.598	0.600	0.599	0.610	0.619	0.629	0.601	0.602	0.60725
7	0.624	0.618	0.613	0.603	0.615	0.617	0.622	0.605	0.61463
8	0.599	0.604	0.618	0.597	0.611	0.594	0.607	0.607	0.60463
9	0.656	0.581	0.613	0.604	0.634	0.604	0.635	0.602	0.61613
10	0.587	0.587	0.618	0.645	0.613	0.609	0.620	0.619	0.61225
11	0.609	0.620	0.616	0.640	0.600	0.609	0.604	0.587	0.61063
12	0.618	0.643	0.615	0.595	0.605	0.614	0.601	0.596	0.61088
13	0.609	0.594	0.607	0.631	0.609	0.610	0.631	0.628	0.61488
14	0.619	0.623	0.580	0.605	0.625	0.602	0.614	0.613	0.61013
15	0.589	0.595	0.610	0.591	0.621	0.606	0.612	0.606	0.60375
16	0.600	0.631	0.626	0.629	0.621	0.593	0.620	0.616	0.61700
17	0.621	0.615	0.593	0.643	0.616	0.604	0.596	0.606	0.61175
18	0.622	0.605	0.619	0.609	0.613	0.620	0.601	0.618	0.61338
19	0.594	0.617	0.606	0.587	0.622	0.619	0.612	0.630	0.61088
20	0.624	0.620	0.609	0.620	0.626	0.597	0.611	0.603	0.61375
21	0.589	0.638	0.625	0.609	0.605	0.629	0.637	0.605	0.61713
22	0.612	0.636	0.599	0.594	0.613	0.609	0.620	0.592	0.60938
23	0.633	0.631	0.586	0.640	0.617	0.592	0.596	0.616	0.61388
24	0.599	0.590	0.618	0.594	0.624	0.625	0.611	0.604	0.60813
25	0.576	0.597	0.609	0.623	0.615	0.590	0.599	0.626	0.60438
26	0.596	0.603	0.619	0.609	0.603	0.606	0.631	0.580	0.60588
27	0.600	0.624	0.611	0.607	0.620	0.605	0.595	0.602	0.60800
28	0.606	0.593	0.630	0.621	0.616	0.596	0.617	0.625	0.61300
29	0.621	0.596	0.608	0.630	0.617	0.611	0.631	0.613	0.61588
30	0.609	0.588	0.642	0.637	0.638	0.615	0.637	0.616	0.62275
31	0.613	0.681	0.694	0.675	0.630	0.618	0.591	0.611	0.63913
Avg.	0.60897	0.61419	0.61413	0.61613	0.61559	0.61072	0.61613	0.61025	**0.61479**

Table 4: Empirical probability (mean over 1K tests) to recover a full message from N traces representing decapsulation of the same ciphertext.

	N					
	1	9	99	199	299	399
p_{HD}	0.61479	0.75866	0.98555	0.99872	0.99991	0.99999
$p_{message}$	$1.3 \cdot 10^{-54}$	$2.5 \cdot 10^{-31}$	0.02444	0.72137	0.97731	0.99745

ity of 99.7%. Such a significant increase over the single-trace case is likely due to the independence of errors in the repeated measurements, since random masks are updated at each execution. A similar phenomenon has also been observed in the attacks on software implementations of CRYSTALS-Kyber, e.g. [16]. In contrast, in the attacks on an unprotected implementations of CRYSTALS-Kyber, multiple repetitions do not increase the success probability so significantly.

Table 5: Empirical probability (mean over 1K tests) to recover $HD(m[i-1], m[i])$ of a message bit $i = 8a + b$ from 299 traces representing decapsulation of the same ciphertext, for $i \in \{1, \ldots, 255\}$.

Byte, a	Bit position in a byte, b								**Avg.**
	0	1	2	3	4	5	6	7	
0	/	1	1	0.999	1	0.998	1	1	0.99957
1	1	1	1	1	0.999	1	1	1	0.99988
2	1	0.999	1	1	0.999	1	1	0.999	0.99963
3	1	1	1	1	1	1	1	1	1.00000
4	1	1	1	1	1	0.999	1	0.999	0.99975
5	1	1	1	1	1	1	1	1	1.00000
6	1	1	1	1	1	1	1	1	1.00000
7	1	1	1	1	1	1	1	1	1.00000
8	0.999	1	1	1	1	1	1	1	0.99988
9	1	1	1	1	1	1	1	1	1.00000
10	1	1	1	1	1	1	1	1	1.00000
11	1	1	1	1	1	1	1	1	1.00000
12	1	1	1	1	1	1	1	1	1.00000
13	1	1	1	1	1	1	1	1	1.00000
14	1	1	0.999	1	0.999	1	1	1	0.99975
15	1	1	1	1	1	1	1	1	1.00000
16	1	1	1	1	0.999	1	1	0.999	0.99975
17	1	1	1	1	1	0.999	1	1	0.99988
18	1	1	1	1	0.998	1	1	1	0.99975
19	1	1	1	1	1	1	1	1	1.00000
20	1	1	1	1	1	1	1	1	1.00000
21	1	1	1	1	1	1	1	1	1.00000
22	1	1	1	1	1	1	1	1	1.00000
23	0.999	1	1	1	1	1	0.999	1	0.99975
24	1	1	1	1	1	1	1	1	1.00000
25	1	1	0.999	1	1	1	1	1	0.99988
26	1	1	1	1	0.999	1	1	1	0.99988
27	1	1	1	1	1	1	1	1	1.00000
28	1	1	1	1	0.999	1	1	1	0.99988
29	1	1	1	0.999	1	0.999	1	1	0.99975
30	1	1	1	1	1	1	1	1	1.00000
31	1	1	1	1	1	1	1	1	1.00000
Avg.	0.99994	0.99997	0.99994	0.99994	0.99975	0.99984	0.99997	0.99991	**0.99991**

8.2 Minimizing training set size

If the DUA is used for profiling, the attacker may be interested to train neural networks on a training set containing as few traces as possible, in order to minimize the access time to the DUA. In this section, we investigate how many traces are required to obtain a model with a high message recovery probability.

We re-train neural networks with the architecture listed in Table 2 using up to 5,000 traces in the initial set, before expansion by cut-end-join. Table 6 shows the average empirical probabilities p_{HD} and full message recovery for the case when the same decapsulation is repeated 299 times and the final prediction is obtained by the majority voting[2]. From Table 6 we can see that with 100 initial traces captured for the DUA one can train a model which predicts messages with nearly 80% average probability.

[2] We also evaluated the case when model's predictions are combined by multiplication, but the results were worse.

Table 6: Empirical probability (mean over 1K tests) to recover a full message from 299 traces representing decapsulation of the same ciphertext using MLPs trained on datasets of different sizes.

	# Traces before cut-and-join						
	50	100	200	500	1000	2000	5000
p_{HD}	0.99242	0.99909	0.99972	0.99986	0.99991	0.99992	0.99993
$p_{message}$	0.14367	0.79282	0.93108	0.96493	0.97731	0.97981	0.98231

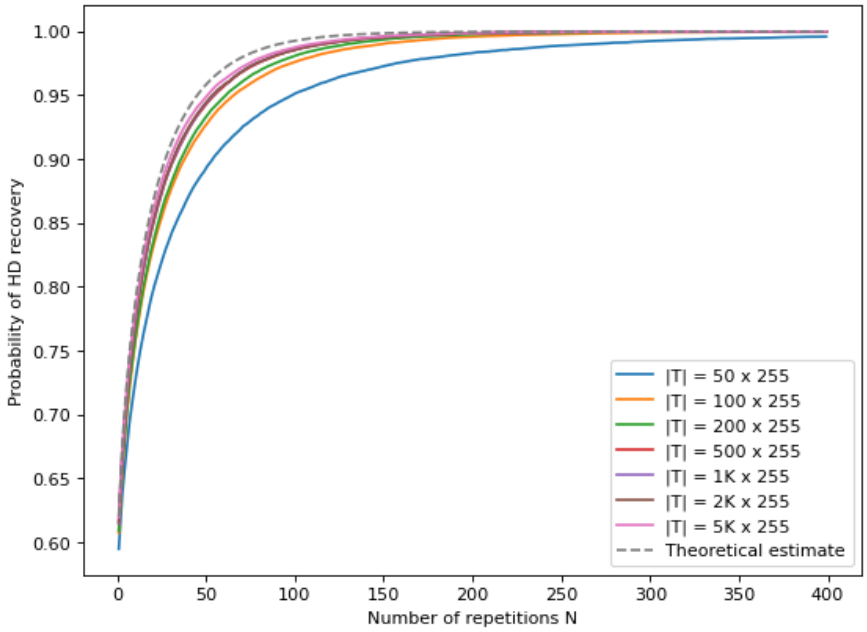

Fig. 6: The probability of recovering $HD(m[0], m[1])$ as a function of the number of repetitions N for different training set sizes.

Fig. 6 shows how the probability of recovering $HD(m[0], m[1])$ grows as a function of the number of repetitions N for different training set sizes. The dashed curve represents the estimated probability computed as:

$$P = \sum_{i=0}^{(N+1)/2} \binom{N}{i} p^{N-i}(1-p)^i,$$

for single-trace recovery probability $p = 0.62$. It defines an upper bound for the empirical probabilities.

Table 7: Empirical probability (mean over 1K tests) to recover a full message from N traces representing decapsulation of the same ciphertext for the case when a different device is used for profiling.

	N					
	1	9	99	199	299	399
p_{HD}	0.59329	0.71349	0.96306	0.99402	0.99877	0.99979
$p_{message}$	$1.5 \cdot 10^{-58}$	$4.1 \cdot 10^{-38}$	$6.8 \cdot 10^{-5}$	0.21665	0.73063	0.94785

8.3 Profiling on a different device

If the access time to the DUA too short to capture both profiling and attack traces, the attaker may attempt to do profiling on a different device. In this section, we evaluate how intra-device variability affects the probability of message recovery.

Following the same steps as in Section 8.1, we re-train a neural network with the architecture listed in Table 2 using 1,000 traces captured from a different FPGA device. Then, we test the resulting model using traces captured from the DUA with multiple repetitions of the same decapsulation and applying majority voting to the predictions. Table 7 shows the results for up to 400 repetitions.

As expected, the average empirical probabilities of message recovery are smaller than the ones in Table 4, where profiling is performed on traces from the DUA. For the case of 399 repetitions, the probability in Table 7 is 94.8% instead of 99.7% in Table 4. To minimize the negative effect of intra-device variability, one can use multiple devices for profiling [9, 43], or fine-tuning by iterative re-training [33].

8.4 A posteriori leakage analysis with known masks

To further understand why the Hamming distance leakage model performs better than the Hamming weight one in the implementation [25], we carry out leakage analysis with known masks. Note that traces for such kind of analysis can only be captured from a device fully controlled by an attacker, since the Verilog code of the implementation has to be modified.

First we perform a t-test with known masks. Fig. 7 shows the results for the Boolean shares, $m1$ and $m2$, computed by `State_Polytomsg_masked_decode()` for a set $\boldsymbol{T}$ of 1,000 traces with known masks. The t-test scores for all 256 bits of both shares are plotted.

We analyse both the Hamming distance and the Hamming weight leakage models. For the Hamming weight case, for the Boolean share $m1$, the set $\boldsymbol{T}$ is partitioned into two sets T_0 and T_1 as:

$$\boldsymbol{T}_0 = \{T \in \boldsymbol{T} \mid HW(m1[i]) = 0\}$$
$$\boldsymbol{T}_1 = \{T \in \boldsymbol{T} \mid HW(m1[i]) = 1\},$$

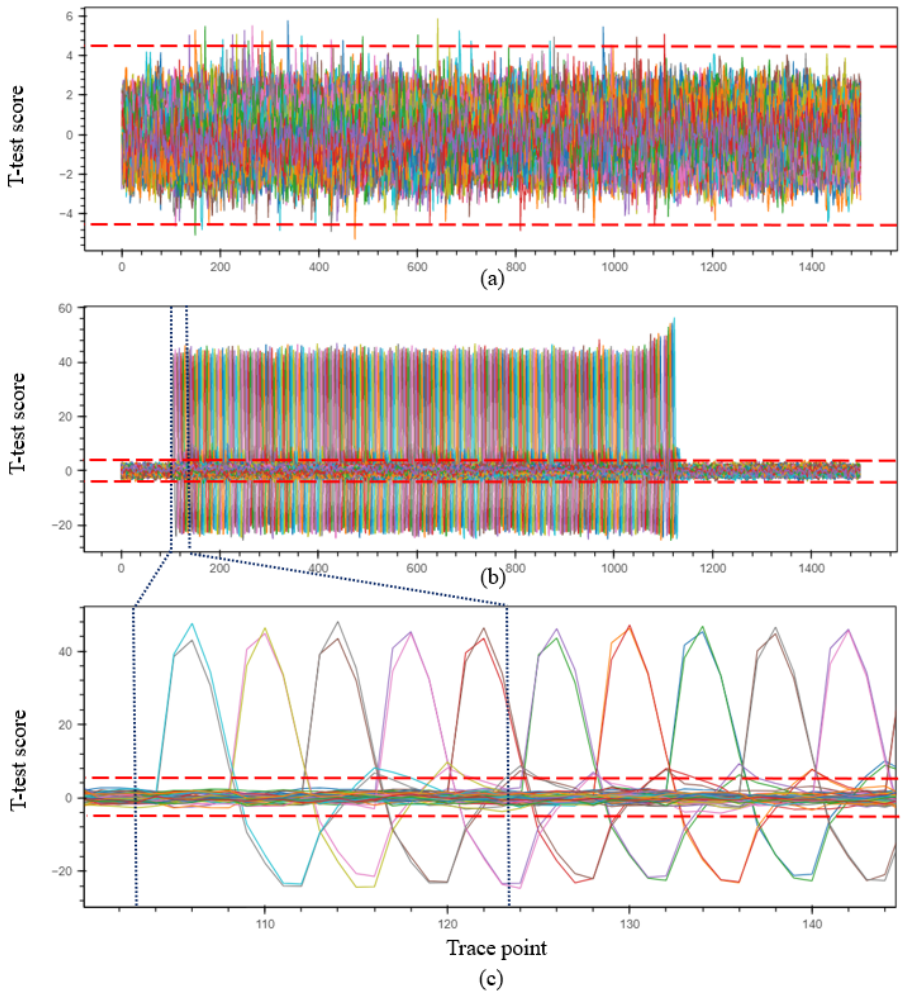

Fig. 7: T-test results for the Boolean shares $m1$ and $m2$ computed by the module `State_Polytomsg`: (a) for the Hamming weight of all bits of both shares; (b) for the Hamming distance between all adjacent bits of both shares; (c) a zoomed-in view of (b). The red lines mark the ± 4.5 t-test threshold.

where $m1[i]$ is the ith bit of $m1$, for all $i \in \{0, 1, \ldots, 255\}$. For the the Boolean share $m2$, $\boldsymbol{T}$ is partitioned similarly.

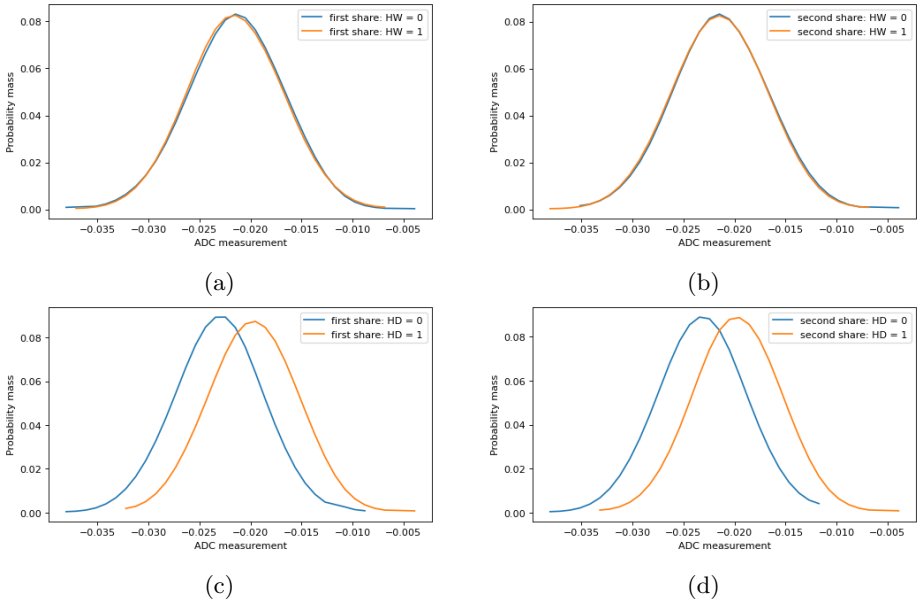

Fig. 8: Distributions of power consumption during the computation of a single bit of Boolean shares $m1$ and $m2$ by `State_Polytomsg__masked_decode()`: (a, b) for the Hamming weights $HW(m1[1])$ and $HW(m2[1])$, respectively; (c, d) for the Hamming distances $HD(m1[0], m1[1])$ and $HD(m2[0], m2[1])$, respectively.

For the Hamming distance case, for the Boolean share $m1$, $\boldsymbol{T}$ is partitioned as:

$$\boldsymbol{T}_0 = \{T \in \boldsymbol{T} \mid HD(m1[i-1], m1[i]) = 0\}$$
$$\boldsymbol{T}_1 = \{T \in \boldsymbol{T} \mid HD(m1[i-1], m1[i]) = 1\},$$

For the the Boolean share $m2$, $\boldsymbol{T}$ is partitioned similarly.

By comparing Fig. 8(a) and (b), we can conclude that leakage in the Hamming distance model is considerable stronger that in the Hamming weight one. From the zoomed-in view in Fig. 8(c), we can also see that the bits of both shares $m1$ and $m2$ are processed in parallel.

Next, we analyse distributions of power consumption during the processing of a single bit of Boolean shares $m1$ and $m2$ by `State_Polytomsg__masked_decode()`. Fig. 8 shows the distributions for each share separately, derived for a set 1,000 traces. We use the second bit as an example, so the Hamming weights for the shares $m1$ and $m2$ are computed as $HW(m1[1])$ and $HW(m2[1])$, respectively, and the Hamming distances are computed as $HD(m1[0],$
$m1[1])$ and $HD(m2[0],$ $m2[1])$, respectively. In both cases, the distributions are derived for the trace point with the maximum absolute t-test score. We can see that, in the Hamming weight case, the overlapping between the distributions is

Listing 1.2: Segment of the Verilog code of `State_Polytomsg`.

```
module State_Polytomsg #(
...
);
...
integer m;
always @(posedge clk or negedge rst_n) begin
  if (rst_n == 1'b0) begin
    oMsg  <= 0;
    m     <= 255;
  end else begin
    if (data_valid) begin
      oMsg[m] <= Cal_oMsg1 ^ Cal_oMsg2;
      m       <= m - 1;
    end
  end
end

State_Polytomsg__masked_decode P0 (
  .clk(clk),
  .rst_n(rst_n),
  .ce(Cal_enable),
  .c1({4'h0, Reduce_DecMp1_RData}),
  .c2({4'h0, Reduce_DecMp2_RData}),
  .PRNG_data(PRNG_data),
  .data_valid(data_valid),
  .m1(Cal_oMsg1),
  .m2(Cal_oMsg2)
);
endmodule
```

nearly complete. Therefore, is is easier for neural networks to classify the input data in the Hamming distance case.

9 Countermeasures

In this section we analyze the implementations of message decoding function and propose countermeasures against the presented attack.

Listing 1.2 presents a segment of the Verilog code of the module `State_Polytomsg` which is relevant to the presented attack. Recall from Section 5 that `State_Polytomsg` converts a polynomial into a message in a masked setting. At each clock cycle, the sub-module `State_Polytomsg__masked_decode` takes two shares of a polynomial coefficient, c_1 and c_2, and computes the corresponding shares of the message bit, m_1 and m_2. In the Verilog code, m_1 and m_2 are denoted

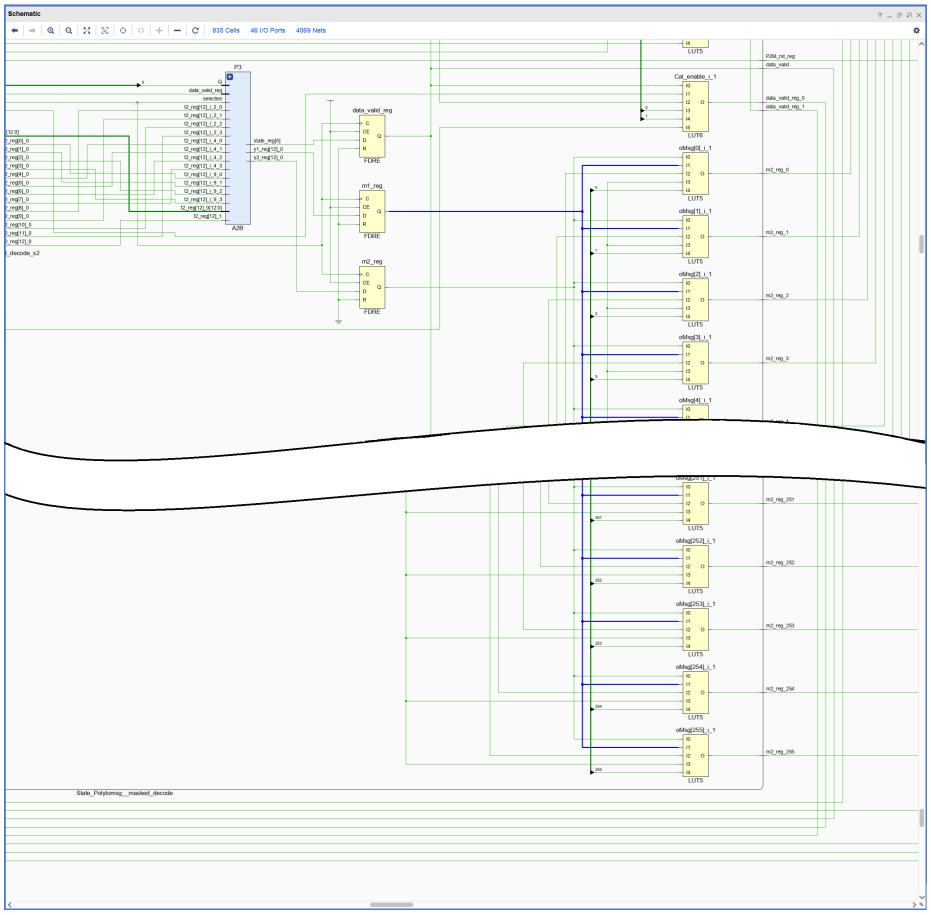

Fig. 9: Gate-level schematic of `State_Polytomsg`. Blue lines indicate the wiring of signal m_1.

by `Cal_oMsg1` and `Cal_oMsg2`, respectively. Once m_1 and m_2 are produced, the decoded message bit is computed as $m_1 \oplus m_2$ and stored in the register `oMsg`.

Fig. 9 shows the gate-level schematic of the `State_Polytomsg` compiled by Vivado Design Suite.We can see that the shares m_1 and m_2 are stored in the registers `m1_reg` and `m2_reg`, respectively. The outputs of theses registers are connected to the inputs of the LUTs `oMsg[j]_i_1` which compute $m[j] = m_1 \oplus m_2$ for $j \in \{0, \ldots, 255\}$. The outputs of the LUTs `oMsg[j]_i_1` are then connected to each entry of the 256-bit register `oMsg` through configurable logic block (CLB) interconnects.

According to Shang et al. [37], LUT inputs have the highest capacitance among all resources in a CLB. In an FPGA, the dynamic power consumption of a component is proportional to its capacitance. Therefore, if a bit of the share

Listing 1.3: Modified Verilog code of `State_Polytomsg`. The register `oMsg` is replaced by a shift register.

```
module State_Polytomsg #(...)
...
always @(posedge clk or negedge rst_n) begin
  if (rst_n == 1'b0) begin
    oMsg  <= 0;
  end else begin
    if (data_valid) begin
      oMsg <= {Cal_oMsg1 ^ Cal_oMsg2, oMsg[255:1]};
    end
  end
end
...
endmodule
```

m_1/ m_2 changes its value, high dynamic power is consumed due to the high fanout of the register `m1_reg`/ `m2_reg`. This causes the strong leakage on HD between the adjacent bits of m_1/ m_2.

We propose to mitigate this problem by replacing the 256-bit register `oMsg` with a 256-bit shift register. The modified Verilog code is shown in Listing 1.3. The sub-module `State_Polytomsg__masked _decode` which is not shown in the code remains the same. Fig. 10 in shows its gate-level schematic. We can see that now the outputs of the registers `m1_reg` and `m2_reg` are connected to a single LUT, `oMsg[255]_i_1`. The output of this LUT is then connected to the register `m2_reg_0` which is the first bit of the shift register `oMsg`. In a shift register, the power consumed at each clock cycle depends not only on the input bit, but also on the HD between the current and the next states. Such an "accumulative" type of leakage is more difficult to explore compared to the bit-wise leakage, as demonstrated by side-channel attacks on arbiter physical unclonable functions which typically store their output responses in a shift register [7].

To validate the effect of this countermeasure, first we perform a t-test with known masks on the modified implementation. Using the same setup as described in Section 6, we capture 1,000 traces during the execution of the decapsulation algorithm with the modified `State_Polytomsg`. Fig. 11 shows the t-test results for the shares m_1 and m_2 for the HD between the adjacent bits. The traces are partitioned into two groups. There are no visible peaks within the range of ± 4.5. Compared to the t-test results of the original design in Fig. 7(b), the leakage of the modified version is much weaker.

Second, we train an MLP model with the architecture shown in Table 2 on 1,000,000 power traces captured during the execution of the decapsulation algorithm using the modified `State_Polytomsg`, with the HD between the first and the second bit as a label. The validation accuracy of the resulting model is

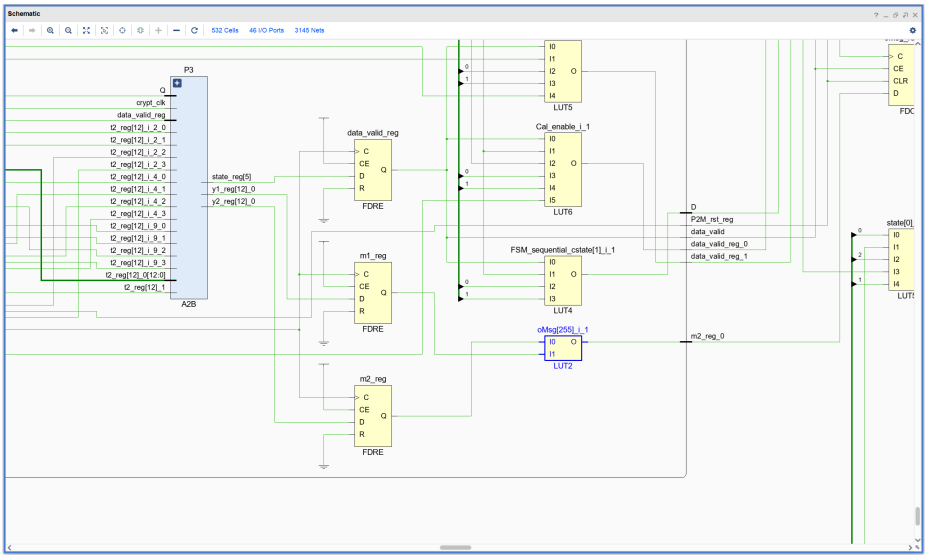

Fig. 10: Gate-level schematic of `State_Polytomsg` after modification.

51.5%, which is much lower than the average single-trace bit recovery accuracy of 61.5% in Table 3.

Using a shift register instead of regular registers for storing sansitive data seems to be a good choice for an FPGA implementation of any cryptographic algorithm. This reduces the SNR, making power and EM analysis more difficult. However, it is important to point out that the shift register alone, without masking, cannot fully protect against side-channel attacks because traces from an unprotected implementation can be denoised by averaging.

There are also other ways to mitigate the presented attack, or make it more difficult. One is to restrict the number of times the same ciphertext can be decapsulated using the same secret key. If this number is less than 100 then, by Table 7, the success probability of full message recovery is less than $6.8 \cdot 10^{-5}$. It may be necessary to allow for a few repetited decapsulations in order to tolerate random communication faults.

Another way to increase the attack difficulty is to convert multiple polynomial coefficients into the message bits in parallel during the message decoding, instead of individually converting each coefficient as currently implemented in the `State_Polytomsg` module.

Alternatively, stronger countermeasures against power analysis attacks can be employed, e.g. the duplication with clock randomization method proposed in [32]. The protected implementation consists of two identical cryptographic cores: a primary and a dummy. Both cores receive the same input data, however they are controlled by two different randomized clocks and use two different secret and public key pairs for their respective operations. Compared to masking,

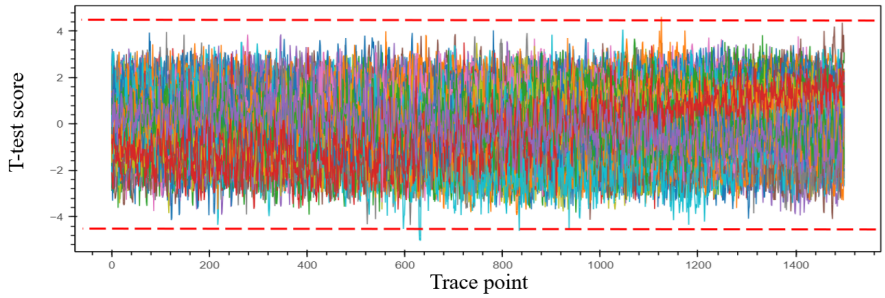

Fig. 11: T-test results for the modified `State_Polytomsg` for the HD between the adjacent bits of both shares m_1 and m_2.

such a method offers advantages of universal coverage, immunity to glitches, zero clock cycle overhead, and stronger resistance to repetition attacks.

10 Conclusion

We demonstrated a practical message recovery attack on the masked FPGA implementation of Kyber-512 by Kamucheka et al. [25] by profiled deep learning-based power analysis using the Hamming distance leakage model. We revealed a low-level vulnerability in the message decoding procedure of this implementation that makes the full shared key recovery possible with the success rate of 99%. We also recommended several approaches to strengthen the resistance of the implementation to power analysis.

11 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. 2025-00475).

References

1. CW-Analyzer Tool - Wiki, `https://wiki.newae.com/CW-Analyzer_Tool`
2. 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`
3. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Crypt. Hardware and Embedded Systems. pp. 29–45 (2003)
4. 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), `https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf`

5. Azouaoui, M., Kuzovkova, Y., Schneider, T., van Vredendaal, C.: Post-quantum authenticated encryption against chosen-ciphertext side-channel attacks. Cryptology ePrint Archive, Paper 2022/916 (2022)
6. Backlund, L., Ngo, K., Gartner, J., Dubrova, E.: Secret key recovery attacks on masked and shuffled implementations of CRYSTALS-Kyber and Saber. Cryptology ePrint Archive, Paper 2022/1692 (2022)
7. Becker, G.T., Kumar, R.: Active and passive side-channel attacks on delay based puf designs. Cryptology ePrint Archive (2014)
8. Belleville, N., Courousse, D., Heydemann, K., Charles, H.P.: Automated software protection for the masses against side-channel attacks. ACM Trans. Archit. Code Optim. **16**(4) (2018)
9. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: A warriors guide through realistic profiled side-channel analysis. In: Network and Distributed System Security Symposium (01 2020). https://doi.org/10.14722/ndss.2020.24390
10. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking Kyber: First- and higher-order implementations. IACR Trans. on Crypt. Hardware and Embedded Systems (4), 173–214 (Aug 2021)
11. Botros, L., Kannwischer, M.J., Schwabe, P.: Memory-efficient high-speed implementation of Kyber on cortex-m4. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology – AFRICACRYPT 2019. pp. 209–228. Springer International Publishing, Cham (2019)
12. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit: with application to lattice-based KEMs. IACR Trans. on Crypt. Hardware and Embedded Systems pp. 553–588 (2022)
13. 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)
14. Coron, J.S., Kizhvatov, I.: An efficient method for random delay generation in embedded software. In: Crypt. Hardware and Embedded Systems. pp. 156–170. Springer Berlin Heidelberg (2009)
15. D'Anvers, J.P., Beirendonck, M.V., Verbauwhede, I.: Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations. Cryptology ePrint Archive, Paper 2022/110 (2022)
16. Dubrova, E., Ngo, K., Gartner, J., Wang, R.: 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. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual international cryptology conference. pp. 537–554. Springer (1999)
18. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: NIST Non-Invasive Attack Testing Workshop. vol. 7, pp. 115–136 (2011)
19. Gross, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. pp. 3–3 (10 2016). https://doi.org/10.1145/2996366.2996426
20. Hoffmann, C., Libert, B., Momin, C., Peters, T., Standaert, F.X.: Towards leakage-resistant post-quantum CCA-secure public key encryption. Cryptology ePrint Archive, Paper 2022/873 (2022)

21. Huang, Y., Huang, M., Lei, Z., Wu, J.: A pure hardware implementation of CRYSTALS-Kyber PQC algorithm through resource reuse. IEICE Electronics Express **17**(17), 1–6 (2020)
22. Jati, A., Gupta, N., Chattopadhyay, A., Sanadhya, S.K.: A configurable CRYSTALS-Kyber hardware implementation with side-channel protection. Cryptology ePrint Archive, Paper 2021/1189 (2021)
23. Ji, Y., Wang, R., Ngo, K., Dubrova, E., Backlund, L.: A side-channel attack on a hardware implementation of CRYSTALS-Kyber. In: 2023 IEEE European Test Symposium (ETS'23) (2023)
24. Kamucheka, T., Fahr, M., Teague, T., Nelson, A., Andrews, D., Huang, M.: Power-based side channel attack analysis on PQC algorithms. Cryptology ePrint Archive, Paper 2021/1021 (2021)
25. Kamucheka, T., Nelson, A., Andrews, D., Huang, M.: A masked pure-hardware implementation of Kyber cryptographic algorithm. In: 2022 International Conference on Field-Programmable Technology (ICFPT). pp. 1–1. IEEE (2022)
26. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg (1999)
27. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
28. Kocher, P.C., Jaffe, J., Jun, B.: Using unpredictable information to minimize leakage from smartcards and other cryptosystems. US Patent 6,327,661
29. Ma, H., Pan, S., Gao, Y., He, J., Zhao, Y., Jin, Y.: Vulnerable PQC against side channel analysis-a case study on Kyber. In: 2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). pp. 1–6. IEEE (2022)
30. Maghrebi, H., Servant, V., Bringer, J.: There is wisdom in harnessing the strengths of your enemy: Customized encoding to thwart side-channel attacks. In: Fast Software Encryption. pp. 223–243 (2016)
31. Moody, D.: Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Nistir 8309 pp. 1–27 (2022), `https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf`
32. Moraitis, M., Brisfors, M., Dubrova, E., Lindskog, N., Englund, H.: A protected hardware implementation of AES using clock randomization and duplication. In: 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, California, USA (2023)
33. 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)
34. 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)
35. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: International Conference on Information and Communications Security. pp. 529–545. Springer (2006)
36. Rodriguez, R.C., Bruguier, F., Valea, E., Benoit, P.: Correlation electromagnetic analysis on an FPGA implementation of CRYSTALS-Kyber. Cryptology ePrint Archive, Paper 2022/1361 (2022)
37. Shang, L., Kaviani, A.S., Bathala, K.: Dynamic power consumption in virtex™-ii fpga family. In: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays. pp. 157–164 (2002)

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)
39. Takarabt, S., Guilley, S., Souissi, Y., Karray, K., Sauvage, L., Mathieu, Y.: Formal evaluation and construction of glitch-resistant masked functions. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST'2021). pp. 304–313 (2021)
40. Tsai, T.T., Huang, S.S., Tseng, Y.M., Chuang, Y.H., Hung, Y.H.: Leakage-resilient certificate-based authenticated key exchange protocol. IEEE Open Journal of the Computer Society **3**, 137–148 (2022)
41. Ueno, R., Xagawa, K., Tanaka, Y., Ito, A., Takahashi, J., Homma, N.: Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. IACR Tran. on Cryptographic Hardware and Embedded Systems **2022**(1), 296–322 (Nov 2021)
42. Veyrat-Charvillon, et al.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: Advances in Cryptology – ASIACRYPT 2012. pp. 740–757. Springer Berlin Heidelberg (2012)
43. Wang, H., Forsmark, S., Brisfors, M., Dubrova, E.: Multi-source training deep learning side-channel attacks. In: IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL'2020) (2020)
44. Welch, B.L.: The generalization of 'Student's´ problem when several different population variances are involved. Biometrika **34**(1-2), 28–35 (1947)
45. Xing, Y., Li, S.: A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-Kyber on FPGA. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 328–356 (2021)

1 Appendix: Timing simulation of decapsulation algorithm

Fig. 12 shows a timing simulation of the decapsulation algorithm in the implementation of Kyber-512 by Kamucheka et al. [25]. The `enable` and `Function_Done` signals for each submodule are shown in the figure, indicating the start and end points of the execution of these modules. It is clear that no other module is running in parallel with `State_Polytomsg()`.

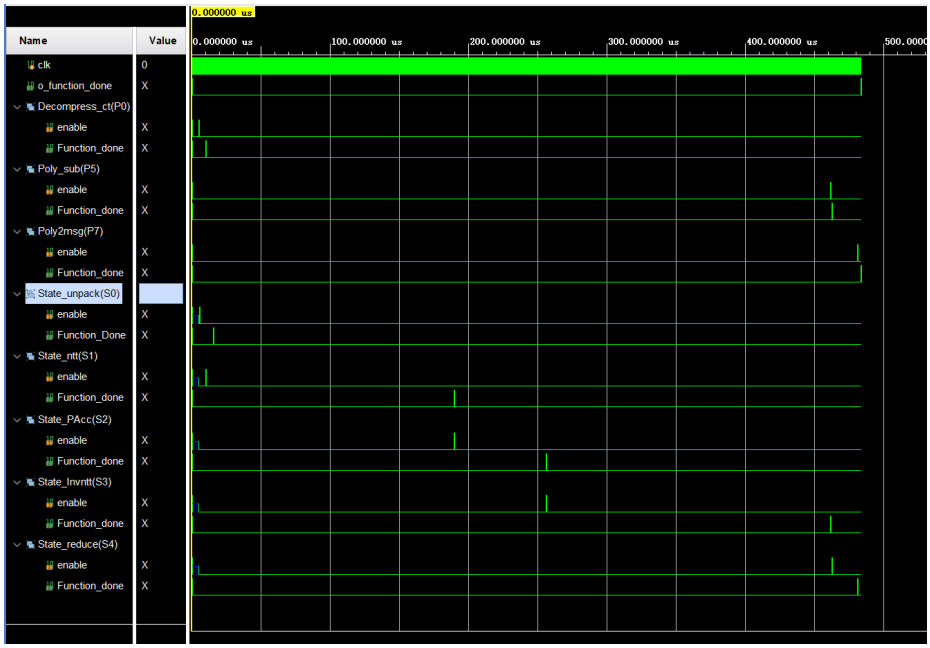

Fig. 12: A timing simulation using Xilinx Vivado Design Suite. The module called `Poly2msg(P7)` corresponds to `State_Polytomsg()`.