Machine Learning based Blind Side-Channel Attacks on PQC-based KEMs - A Case Study of Kyber KEM

Prasanna Ravi, Dirmanto Jap, Shivam Bhasin
Temasek Labs
Nanyang Technological University
Singapore
prasanna.ravi,djap,sbhasin@ntu.edu.sg

Anupam Chattopadhyay
Temasek Labs and School of Computer Science and Engineering
Nanyang Technological University
Singapore
anupam@ntu.edu.sg

***Abstract*—Kyber KEM, the NIST selected PQC standard for Public Key Encryption and Key Encapsulation Mechanisms (KEMs) has been subjected to a variety of side-channel attacks, through the course of the NIST PQC standardization process. However, all these attacks targeting the decapsulation procedure of Kyber KEM either require knowledge of the ciphertexts or require to control the value of ciphertexts for key recovery. However, there are no known attacks in a *blind* setting, where the attacker does not have access to the ciphertexts. While blind side-channel attacks are known for symmetric key cryptographic schemes, we are not aware of such attacks for Kyber KEM. In this paper, we fill this gap by proposing the first blind side-channel attack on Kyber KEM. We target leakage of the pointwise multiplication operation in the decryption procedure to carry out practical blind side-channel attacks resulting in full key recovery. We perform practical validation of our attack using power side-channel from the reference implementation of Kyber KEM taken from the *pqm4* library, implemented on the ARM Cortex-M4 microcontroller. Our experiments clearly indicate the feasibility of our proposed attack in recovering the full key in only a few hundred to few thousand traces, in the presence of a suitably accurate Hamming Weight (HW) classifier.**

***Index Terms*—Post-Quantum Cryptography; Blind Side-Channel Attacks; Kyber; Lattice-based cryptography; Power-based Side-Channel Attacks**

I. Introduction

The impending threat of quantum computers towards public-key cryptography prompted the National Institute of Standards and Technology to initiate a global level standardization effort for Post-Quantum Cryptography (PQC), which is resistant to attacks against large-scale quantum computers [1]. The process which started in 2017, completed its third round in July 2022, when it announced the first set of algorithms to be standardized for PQC Key Encapsulation Mechanisms (KEMs) and Digital Signatures Schemes (DSS) [2]. Kyber KEM [3], based on the well-known Module Learning With Error (MLWE) problem was the only algorithm that was standardized for KEMs, owing to its fine balance of both security and efficiency guarantees [2]. The standardization of Kyber KEM will ensure its implementation on a wide-variety of computing devices, including resource constrained platforms such as embedded microcontrollers [4], [5]. This naturally makes them susceptible to side-channel attacks, which was also an important consideration during the NIST PQC standardization process [6].

Given the importance of protecting Kyber against SCA, there were several works published on both SCA as well as protected development of protected implementations of Kyber on embedded devices [7]–[9]. Refer to [6] for a detailed survey of the various side-channel attacks on Kyber KEM. The decapsulation procedure of Kyber KEM serves as the main target for an attacker, as it manipulates the secret key for multiple executions, allowing the attacker to observe multiple side-channel traces for key recovery. Existing attacks on the decapsulation procedure can be categorized into two broad categories, based on the attacker's based on the attacker's knowledge of the input to Kyber's decapsulation procedure. They are (1) Known Ciphertext Attacks (KCA) and (2) Chosen Ciphertext Attacks (CCA). In KCA-style attacks, the attacker only requires the knowledge of inputs to the decapsulation procedure, and in CCA-style attacks, the attacker requires to control the input to the decapsulation procedure [6].

However, it is possible in certain scenarios that the attacker does not obtain access to the input of the DUT (Refer to Page 128 of [10]) or when using Kyber in any setting that limits the adversary's access to the I/O of the target device. In such a setting, the attacker only has access to the side-channel traces from the decapsulation procedure, and this prompts the question if an attacker can still perform key recovery without the knowledge of the DUT's ciphertext inputs. In this work, answer this question positively by demonstrating the first *blind* side-channel attack on Kyber KEM, and the concrete contributions of this work as manifold.

Contribution

1) To the best of our knowledge, we propose the first blind side-channel attack on Kyber KEM, by targeting leakage from the pointwise-multiplication operation of the secret key with the ciphertext in the decryption procedure.

2) We observe that the pointwise-multiplication operation manipulates the secret key one coefficient at a time. This enables us to exploit the joint distribution of the input and output coefficients of this operation, and recover single coefficients of the secret key polynomials, without any knowledge of the ciphertext input to the decapsulation procedure.
3) We performed extensive simulations of our attack for two cases: (1) assuming perfect Hamming Weight (HW) classifier, we were able to to recover the secret coefficients with 100% success rate in just 820 traces (2) assuming an imperfect HW classifier with 95% accuracy, we are able to recover the secret coefficients with 100% success rate in about 7805 traces. Our attack can also work with HW classifiers of lower accuracy, but requiring more number of traces.
4) We perform practical validation of our attack using power side-channel leakage obtained using the ChipWhisperer platform [11] on the reference implementation of Kyber KEM taken from the *pqm4* library [4], implemented on the ARM Cortex-M4 microcontroller. We tested our attack to recover 20 random secret coefficients and were able to successfully recover all these coefficients only using ≈ 35 to 5000 traces. Our experiments indicate the feasibility to efficiently recover the secret key, thereby highlighting the capability of blind side-channel attacks on Kyber KEM.

II. Lattice Preliminaries

A. Notations

We denote the ring of integers modulo a prime q as $\mathbb{Z}_q$. The polynomial ring $\mathbb{Z}_q(x)/\phi(x)$ is denoted as R_q. We denote $\mathbf{r} \in R_q^{k\times\ell}$ as a *module* of dimension $k \times \ell$. Polynomials in R_q and modules in $R_q^{k\times\ell}$ are denoted in bold lower case letters. The i^{th} coefficient of a polynomial $\mathbf{A} \in R_q$ is denoted as $\mathbf{A}[i]$. Product of polynomials $\mathbf{a}$ and $\mathbf{b}$ in the ring R_q is denoted as $\mathbf{c} = \mathbf{a} \times \mathbf{b}$, while coefficient-wise multiplication is denoted using the symbol $\circ$. Kyber utilizes the well-known Number Theoretic Transform (NTT) for polynomial multiplication. The output of NTT over a polynomial $\mathbf{a} \in R_q$ is denoted as $\hat{\mathbf{a}}$. The product $\mathbf{c} = \mathbf{a} \times \mathbf{b}$ using NTT is computed as $\mathbf{c} = \mathsf{INTT}(\mathsf{NTT}(\mathbf{a}) \circ \mathsf{NTT}(\mathbf{b}))$. Byte arrays of length n are denoted as $\mathcal{B}^n$. The i^{th} bit in an element $x \in \mathbb{Z}_q$ is denoted as x_i.

B. Kyber KEM

Kyber is a chosen-ciphertext secure (CCA-secure) KEM based on the hardness of Module-Learning With Errors (MLWE) problem [3]. The search MLWE problem requires the attacker to solve for $(\mathbf{s}, \mathbf{e}) \in R_q^k$ given polynomially many LWE instances of the form $(\mathbf{a}, \mathbf{t} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \in (R_q^{k\times k} \times R_q^k)$, where coefficients of $\mathbf{a}$ are uniformly in random in the range $[0, q]$, while coefficients of $\mathbf{s}$ and $\mathbf{e}$ are sampled from a smaller range $[-\eta, \eta]$ based on a Centered Binomial Distribution (CBD) with $\eta << q$.

1) Algorithmic Description: Kyber offers three security levels: Kyber512 (NIST Security Level 1), Kyber768 (Level 3) and Kyber1024 (Level 5) with $k = 2, 3$ and 4 respectively. It operates over the anti-cyclic polynomial ring R_q with a prime modulus $q = 3329$ and degree $n = 256$. The CCA-secure Kyber contains in its core, a Chosen-Plaintext secure encryption scheme of Kyber (i.e.) IND-CPA secure Kyber PKE scheme. Refer to Algorithm 1 for a simplified description of the IND-CPA secure Kyber PKE scheme. The function Sample_U denotes sampling from a uniform distribution; the function Expand inflates a small seed into a uniformly random matrix in $R_q^{k\times k}$; and the function Sample_B uses a short seed to sample coefficients from the Centered Binomial Distribution (CBD) in $[-\eta, \eta]$ respectively.

Algorithm 1 IND-CPA secure Kyber PKE Scheme

1: **procedure** KeyGen
2: $seed_A \in \mathcal{B}^{32} \leftarrow \mathsf{Sample}_U()$
3: $seed_B \in \mathcal{B}^{32} \leftarrow \mathsf{Sample}_U()$
4: $\hat{\mathbf{a}} \in R_q^{k\times k} \leftarrow \mathsf{Expand}(seed_A)$ ▷ Sample $\hat{\mathbf{a}}$
5: $\mathbf{s} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_B, coin_s)$ ▷ Sample $\mathbf{s}$
6: $\mathbf{e} \in R_q^k \leftarrow \mathsf{Sample}_B(seed_B, coin_e)$ ▷ Sample $\mathbf{e}$
7: $\hat{\mathbf{t}} = \hat{\mathbf{a}} \circ \mathsf{NTT}(\mathbf{s}) + \mathsf{NTT}(\mathbf{e})$ ▷ Compute $\mathsf{NTT}(\mathbf{t})$
8: Return $(pk = (seed_A, \hat{\mathbf{t}}), sk = (\mathsf{NTT}(\mathbf{s}))$
9: **end procedure**

10: **procedure** CPA.Encrypt$(pk, m, seed_R)$
11: $\hat{\mathbf{a}} \in R_q^{k\times k} \leftarrow \mathsf{Expand}(seed_A)$
12: $\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2 \in (R_q^k \times R_q^k \times R_q) \leftarrow \mathsf{Sample}_B(seed_R)$
13: $\mathbf{u} \in R_q^k \leftarrow \mathsf{INTT}(\mathbf{A}^T \circ \mathsf{NTT}(\mathbf{r})) + \mathbf{e_1}$
14: $\mathbf{v}_p \in R_q \leftarrow \mathsf{INTT}(\hat{t}^T \circ \mathsf{NTT}(\mathbf{r})) + \mathbf{e_2}$
15: $\mathbf{v} = \mathbf{v}_p + \mathsf{Encode}(m)$
16: Return $ct = \mathsf{Compress}(\mathbf{u}, d_1), \mathsf{Compress}(\mathbf{v}, d_2)$
17: **end procedure**

18: **procedure** CPA.Decrypt(sk, ct)
19: $(\mathbf{u}', \mathbf{v}') \in R_q^k = \mathsf{Decompress}(\mathbf{u}, \mathbf{v})$
20: $\mathsf{w}^* = \mathsf{NTT}(\mathbf{u}') \circ \mathsf{NTT}(\mathbf{s})$
21: $\mathbf{m}' = \mathbf{v}' - \mathsf{INTT}(\mathsf{w}^*)$
22: $m' \in \mathcal{B}^* = \mathsf{Decode}(\mathbf{m}')$
23: Return m'
24: **end procedure**

Referring to Alg.1, the key-generation procedure (KeyGen) of Kyber PKE simply involves computation of an MLWE instance $\mathbf{t} \in R_q^k = (\mathbf{a} \cdot \mathbf{s} + \mathbf{e})$ (Line 7), where $\hat{\mathbf{a}} \in R_q^{k\times k}$ is sampled from a public seed $seed_A \in \mathcal{B}^{32}$ and $\mathbf{s} \in R_q^k$ is the secret and $\mathbf{e} \in R_q^k$ is the error component of the LWE instance. The tuple $(seed_A, \mathbf{t})$ forms the public key, while $\mathsf{NTT}(\mathbf{s})$ (i.e.) $\hat{\mathbf{s}}$ forms the secret key (Line 8).

The encryption procedure (Encrypt) involves generation of two LWE instances $\mathbf{u} \in R_q^k$ and $\mathbf{v}_p \in R_q$ where $\mathbf{u} = \mathbf{a}\cdot\mathbf{r}+\mathbf{e}_1$ (Line 13) and $\mathbf{v}_p = \mathbf{t} \cdot \mathbf{r} + \mathbf{e}_2$ (Line 14). Subsequently, the message $m \in \mathcal{B}^{32}$ is encoded into a polynomial $\mathbf{m} \in R_q$ using the Encode function, and added to $\mathbf{v}_p$ resulting in $\mathbf{v} \in R_q$

(Line 15). The ciphertext is nothing but compressed versions of the tuple $(\mathbf{u}, \mathbf{v})$ (Line 16).

The decryption procedure (Decrypt) extracts the decompressed ciphertext components ($\mathbf{u}' \in R_q^k, \mathbf{v}' \in R_q$) and simply computes the noisy message polynomial $\mathbf{m}' = \mathbf{v}' - \mathbf{u}' \cdot \mathbf{s}$ (Line 21). It is important to note that the aforementioned computation is performed using NTT operations. The resultant message polynomial $\mathbf{m}'$ will be correctly decoded to m (Line 22). The parameters of Kyber are chosen so as to have a negligible decryption failure rate ($\approx 2^{-165.2}$ for recommended parameters of Kyber).

III. Prior Works and Motivation

Kyber KEM, along with other KEMs based on the LWE/LWR problem have been shown to be vulnerable against a wide-variety of side-channel attacks [6]. Most of the reported SCA have targeted its decapsulation procedure, since it manipulates the long-term secret key over multiple executions. In this respect, we can broadly classify such attacks on the decapsulation procedure into two broad categories, based on the attacker's knowledge of the input (i.e.) ciphertext ct, to Kyber's decapsulation procedure. They are (1) Known Ciphertext Attacks (KCA) and (2) Chosen Ciphertext Attacks (CCA).

A. Known Ciphertext Attacks (KCA)

These attacks are those that require the attacker to have knowledge of ciphertext to recover the long-term secret key sk. In this respect, Primas *et al.* [12] and Pessl *et al.* [13] showed that a single trace from the INTT operation (Line 21) for a known ciphertext can be used to recover the input to the INTT operation (i.e.) $\mathbf{w}' \in R_q$. This enables full key recovery using a single trace from the decapsulation operation on the ARM Cortex-M4 microcontroller for a known ciphertext.

Subsequently, Mujdei *et al.* [14] reported Correlation Power Analysis (CPA) style attacks targeting leakage from the computation of $\mathbf{w}^* \in R_q$ (i.e.) the pointwise multiplication operation (Line 20 of the CPA.Decrypt procedure). They demonstrated that traditional CPA can be used to recover the complete key in about 200 side-channel traces. More recently, a couple of works also demonstrated the power of template attacks targeting leakage from the pointwise multiplication operation for single trace key recovery [15], [16].

B. Chosen Ciphertext Attacks (CCA)

These attacks are those that require the attacker to query the decapsulation procedure with chosen ciphertexts. This is a strong assumption compared to KCA attacks, as the attacker requires to communicate with the DUT, while KCA attacks only require the knowledge of the ciphertexts.

Remarkably, Ravi *et al.* [9] were the first to demonstrate that an attacker can craft malicious ciphertexts so as to amplify the leakage of the secret key from the decapsulation procedure. Several operations in the decapsulation procedure after decryption, could be used to instantiate a decryption oracle for full key recovery in a few thousand chosen-ciphertext queries. Subsequent attacks reported in [17]–[19] demonstrated the ability to perform more efficient side-channel assisted chosen-ciphertext attacks, through exploitation of finer leakages from the decapsulation procedure. All these attacks require the attacker to construct specially crafted inputs, so as to evoke increased leakage from the decapsulation procedure for key recovery.

C. Motivation

All reported attacks targeting the decapsulation procedure for key recovery belong to either of these two categories, where the attacker either requires to know the ciphertexts or control the ciphertexts fed to the decapsulation procedure. However, it is possible in certain scenarios that the attacker does not obtain access to the input of the DUT [10]. In such a setting, the attacker only has access to the side-channel traces, and it begets the question if an attacker can still perform key recovery without the knowledge of the DUT's ciphertext inputs.

Prior works have shown that it is possible to mount side-channel attacks on symmetric cryptographic schemes such as AES, without any knowledge of the plaintext or the ciphertext [20], [21]. Such attacks are commonly referred to as *blind* side-channel attacks. However, to the best of our knowledge, we are not aware of any *blind* side-channel attacks on PQC schemes. Since the fundamental mathematical structure of PQC schemes are completely different from that of schemes such as AES, it is not clear whether blind side-channel attacks are possible on PQC schemes.

In this work, we answer this question positively by demonstrating the first *blind* side-channel attack on Kyber KEM, targeting the pointwise polynomial multiplication operation within the decryption procedure.

IV. Blind Side-Channel Attack Methodology

A. Attack Intuition

Our attack idea is inspired from prior blind side-channel attacks on software implementations of the AES algorithm [20], [21]. We now briefly describe the intuition behind prior attacks, to motivate our attack on Kyber KEM. These attacks work by targeting leakage from the plaintext (m) as well as the output of the SBox operation (y), where $y = \mathsf{SBox}(m \oplus k)$ and k is the secret key, and this operation is performed in a bytewise manner. We consider a single byte of every component (i.e.) m, y and k, but use the same notation for simplicity. The key observation is that the joint distribution of HW of m and y (i.e.) tuple ($\mathsf{HW}(m)$, $\mathsf{HW}(y)$) is unique for every possible value of the secret key byte k (i.e.) $k = [0, 255]$. The attack works in three phases.

1) • Phase-I: This phase involves building a joint distribution HWset_{hist} of the tuple $(\mathsf{HW}(m), \mathsf{HW}(y))_k$ for every possible value of the secret key byte k (i.e.) $k \in [0, 255]$. This can be generated by computing y for random values of $m \in [0, 255]$, and repeat this for all values from $k \in [0, 255]$. One needs to collect sufficient number of data points of m such that the pdf converges.

2) • Phase-II: This phase involves collection of multiple attack traces (N) t_i for $i \in [0, N-1]$, from the target device and the leakage of m_i and y_i in each trace t_i is used to obtain the corresponding tuple $(\mathsf{HW}(m_i), \mathsf{HW}(y_i))$ using a suitable HW Classifier. Let the corresponding HW tuple set obtained be denoted as HWset_{attack}. In order to build the HW classifier, we assume that the attacker has access to a clone device, on which he/she can profile the leakage of m_i and y_i, by controlling the input m and the secret key k.
3) • Phase-III: This phase involves application of the Maximum Likelihood (ML) approach to the tuple set HWset_{attack} obtained from the N attack traces, on top of the HWset_{hist} to recover the correct value of the subkey byte k. This process is repeated for all the secret key bytes k_i for $i \in [0, 15]$ to recover the entire secret key of AES.

The attack was mainly possible due to bytewise manipulation of the key (i.e.) (k_i) for $i \in [0, 15]$, which ensures that the attacker can profile leakage of intermediate variables related to only a single byte of the key. This ensures that the attacker only has to distinguish between 256 different joint-distributions for recovering the key byte (subkey), and higher the number of hypotheses for the subkey, higher is the difficulty to obtain unique distinguishability. This motivated us to look for similar operations in the decapsulation procedure of Kyber KEM that manipulates single coefficients of the secret key module $\mathbf{s}$.

B. Targeting Pointwise Multiplication in Decryption

Referring to the decryption procedure (i.e.) CPA.Decrypt, we observe that computation of $\mathbf{w}^* \in R_q$ through the *pointwise multiplication* of $\mathsf{u}^* = \mathsf{NTT}(\mathbf{u}) \in R_q^k$ and $\mathbf{s}^* = \mathsf{NTT}(\mathbf{s}) \in R_q^k$, involves manipulation of single coefficients of $\mathbf{s}^*$. This operation therefore serves as a natural target for blind side-channel analysis. The component $\mathbf{w} \in R_q$ is computed as follows:

$$\mathbf{w} = \sum_{i=0}^{i=k-1} \mathbf{u}^*[i] \circ \mathsf{s}^*[i] \tag{1}$$

where individual polynomials of $\mathbf{u}^*$ are pointwise-multiplied with the corresponding polynomials of s^* and subsequently accumulated to compute $\mathbf{w} \in R_q$. The pointwise multiplication of two polynomials say $\mathbf{u}^*[i]$ and $\mathsf{s}^*[i]$ for $i \in [0, k-1]$ is carried out using the *basemul* function (Refer Alg.2).

For brevity, we denote the input polynomials as $\mathbf{a}$ (resp. $\mathsf{s}^*[i]$) and $\mathbf{b}$ in R_q (resp. $\mathsf{u}^*[i]$), and the *basemul* function operates on pairs of coefficients of $\mathbf{a}$ and $\mathbf{b}$ denoted as $\mathbf{a}[2] \in \mathbb{Z}_q^2$ and $\mathbf{b}[2] \in \mathbb{Z}_q^2$, and a constant ζ as shown in Alg.2, and the output coefficients are denoted as $\mathbf{r}[2] \in \mathbb{Z}_q^2$. In Alg.2, the function fqmul multiplies two coefficients directly and reduces the result using montgomery reduction. So, the result $\mathbf{r}[2]$ is computed using a total of five fqmul operations on single coefficients of $\mathbf{a}$ and $\mathbf{b}$.

We refer to Line 2, which corresponds to fqmul operation on $\mathbf{a}[1]$ and $\mathbf{b}[1]$, whose output is stored in $\mathbf{r}[0]$. This operation involves loading of $\mathbf{a}[1]$ and $\mathbf{b}[1]$ from memory into registers, and subsequently storing the result $\mathbf{r}[0]$ back into memory. This results in side-channel leakage of $\mathsf{HW}(\mathbf{b}[1])$ and output $\mathsf{HW}(\mathbf{r}[0])$ respectively. If the joint distribution of $(\mathsf{HW}(\mathbf{b}[1]), \mathsf{HW}(\mathbf{r}[0]))$ is unique for every possible value of $\mathbf{a}[1] \in [0, q]$ where $q = 3329$, this enables key recovery of the exact value of $\mathbf{b}[0]$. The same can also be applied to the fqmul operation in Line 5, to recover $\mathbf{a}[0]$ through the joint distribution of the tuple $(\mathsf{HW}(\mathbf{b}[0]), \mathsf{HW}(\mathbf{r}[1]))$. This process can be repeated to recover the complete polynomial $\mathbf{a}$, two coefficients at a time.

Algorithm 2 Pseudo Code of Basemul and Fqmul Function

1: **procedure** BASEMUL($r[2] \in \mathbb{Z}_q^2, a[2] \in \mathbb{Z}_q^2, b[2] \in \mathbb{Z}_q^2, \zeta$)
2: $r_0 \in \mathbb{Z}_q = \mathsf{fqmul}(a[1], b[1])$
3: $r_0 = \mathsf{fqmul}(r[0], \zeta)$
4: $r_0 += \mathsf{fqmul}(a[0], b[0])$
5: $r_1 \in \mathbb{Z}_q = \mathsf{fqmul}(a[0], b[1])$
6: $r_1 += \mathsf{fqmul}(a[1], b[0])$
7: Return $r[0], r[1]$
8: **end procedure**

C. Analyzing the Feasibility of Key Recovery: Simulated Attack

We used simulated traces (based on HW leakage), to test the possibility of obtaining unique distinguishability for every possible value of the secret coefficient in the range $[0, q]$ given sufficient number of traces". We performed an attack on simulated traces targeting recovery of $\mathbf{a}[1]$ in Line 2. We first built a joint distribution HWset_{hist} consisting of $q = 3329$ tuples $(\mathsf{HW}(\mathbf{b}[1]), \mathsf{HW}(\mathbf{r}[0]))$ for every possible value of the secret coefficient $\mathbf{a}[1] in [0, q]$. This joint distribution can be pre-computed from the specification of the scheme, and does not require any access to the DUT. In Phase-II, we generate a set of simulated attack traces (with and without Gaussian noise), to imitate the real world traces. We then extract the set of noisy HW tuples HWset_{attack} corresponding to the leakage of $\mathbf{b}[1]$ and $\mathbf{r}[0]$, assuming the presence of a suitable HW classifier. In Phase-III, we compute the Maximum Likelihood of HWset_{attack} using the distribution HWset_{hist} for all possible values of the secret coefficient $\mathbf{a}[1]$, to recover the correct value of $\mathbf{a}[1]$.

Thus, to construct an actual attack, we require a HW classifier in Phase-II, and an appropriate Maximum Likelihood (ML) technique that can result in key recovery. We first explain the Maximum Likelihood principle used for key recovery in the following discussion.

1) Maximum Likelihood Approach for Key Recovery: For simplicity, we will henceforth denote the input coefficient $\mathbf{b}[1]$ as m and output coefficient $\mathbf{r}[0]$ as y, and the secret coefficient $\mathbf{a}[1]$ to be recovered as k. Let the true HW tuple of (m, y) be denoted as (h_m^*, h_y^*). However, the recovered HW tuple using the HW classifier on the side-channel trace is denoted as $(h_m, h_y) = (h_m^* + n_m, h_y^* + n_y)$, where n_m and n_y are Gaussian noise with appropriate standard deviation.

The probability of the the coefficient to be k given a single observation (h_m, h_y) is given by the Bayes formula as follows:

$$\mathsf{Pr}(k|(h_m, h_y)) = \frac{\mathsf{Pr}((h_m, h_y)|k)) \cdot \mathsf{Pr}(k)}{\mathsf{Pr}(h_m, h_y)} \tag{2}$$

$$\tag{3}$$

It is important to note that the denominator term is independent of the key k, and since we are only interested in comparison between the different values of the coefficient, we can ignore the denominator. Moreover, the value of k is uniformly distributed in the range $[0, q]$, and thus $\mathsf{Pr}(k) = 1/q$ for all $k \in [0, q]$. Thus, the probability of the coefficient to be k given a set of observations $((h_m, h_y)_i)_{i=0,1,\ldots,N}$ is denoted as:

$$\mathsf{Pr}(k|(h_m, h_y)_i)_{i=0,\ldots,N} \tag{4}$$

$$\propto \mathsf{Pr}((h_m, h_y)_N|k) \cdot \mathsf{Pr}(k|(h_m, h_y)_i)_{i=0,\ldots,N-1} \tag{5}$$

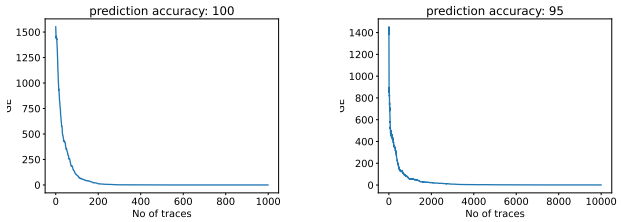
and thus this expression can be calculated in an iterative way for all possible values of k for N observations (traces). The value that yields the highest probability over n traces is defined as the most likely key candidate. This process can be repeated for all the coefficients separately to recover the entire secret polynomial.

2) Evaluation of Our Simulated Attack: We first performed our attack on ideal simulated traces with zero noise (i.e.) perfect HW classifier, and attempted to recover hundred (100) random coefficients, while also repeating each of our attack over 10 runs. We were able to obtain an average Guessing Entropy (GE) of 0 for all the coefficients in about $N \approx 671$ traces. We also investigated the effect of noisy HW predictions on our attack efficacy. We then assumed a HW classifier with 95% accuracy. The classifier returns an erroneous prediction based on a Gaussian Distribution with standard deviation $\sigma = 1$, and the value is rounded to nearest non-zero integer which is then added to actual HW value. We also clip the value to ensure that it falls within valid prediction range (between 0 and max HW). We were able to obtain an average GE of 0 for $N \approx 8524$ traces ($\approx 12.7\times$ increase compared to 100% accuracy). Refer to Fig.1(a)-(b) for the GE plot versus number of traces in the presence of a perfect HW classifier and HW classifier with 95% accuracy respectively. These experiments clearly demonstrate the capability of our attack to work, even in the presence of errors in the HW predictions. While our attack can also potentially work with a lesser accurate HW classifier, it corresponds to a higher number of traces for key recovery. In the following, we present experimental validation of our attack using power side-channel traces obtained from the software implementation of Kyber KEM on the ARM Cortex-M4 microcontroller.

V. Experimental Evaluation

A. Experimental Setup

We validated the capability of our attack on the reference implementation of Kyber KEM, taken from the public *pqm4* library [4], a benchmarking and testing framework for PQC schemes on the 32-bit ARM Cortex-M4 microcontroller,



(a) HW Classifier (100%) (b) HW Classifier (95%)

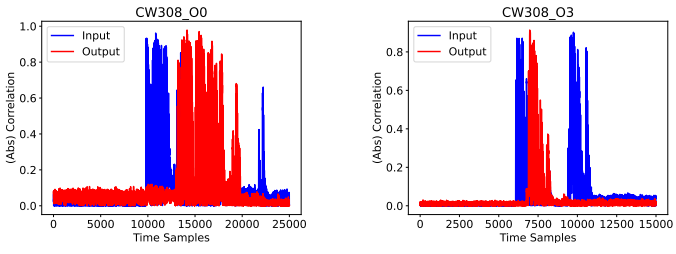
Fig. 1. Plot of Guessing Entropy of the secret key coefficients versus the number of side-channel traces for our simulated attack

which is also a NIST recommended optimization target for embedded software implementations. We utilized power side-channel leakage, captured from an STM32F3 microcontroller (running at 7.372 MHz) using the ChipWhisperer CW308 setup [11]. The measurements are then collected using a Lecroy 610Zi oscilloscope, at a sampling rate of 500×10^6 samples per second. We also used a 48 MHz analog low-pass filter to remove high frequency noise in our traces. We performed experiments on two types of implementations: (1) `O0`-optimized: no optimization and (2) `O3`-optimized: highest compiler optimization.

B. Constructing a HW Classifier

For classification of the HW classes, we used Random Forest, which has been used in previous works [22]. Random Forest [23] is an ensemble learning algorithm that is based on the construction of multiple decision trees. The predictions from trees are combined in order to achieve a more accurate prediction. The individual decision tree is sensitive to small changes in the training data, and when grown large enough, will usually tend to overfit the training data. RF is designed to address these problems of instability in the decision tree, and multiple trees are allowed to grow large, without the need of post-processing. These decision trees are trained on different subsets of the training data using bootstrapping methods (the training data is sampled uniformly with replacement), and the decision is then made by taking the average or majority voting of the decisions given by the trees.

We construct a HW classifier for the input and output coefficients (i.e.) m and y of the target *basemul* function,



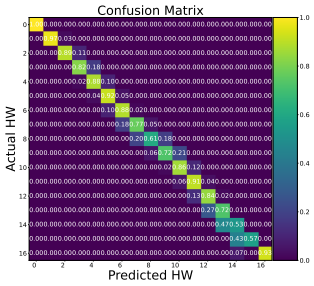
(a) `O0` Implementation (b) `O3` Implementation

Fig. 2. Correlation Power Analysis to identify Points of Interest (PoI) for the input variables m and output variables y, used in the target *basemul* function

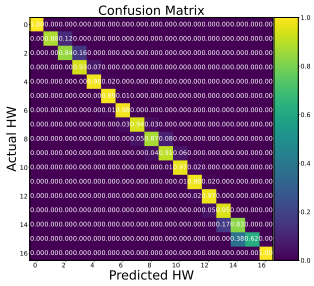
used in the pointwise-multiplication operation. The first step is to locate the Points of Interest (PoI) corresponding to manipulation of the variables m and y. We use leakage from a clone device, where the attacker can control both the input and secret key. We utilize the classic Correlation Power Analysis (CPA) technique [24] to locate the PoIs corresponding to x and y, using knowledge of the secret coefficient k. Refer to Fig.2(a)-(b) for the CPA plots corresponding to the `O0` and `O3` optimized implementations respectively, which clearly contains peaks corresponding to manipulation of the first input and output coefficients of the involved polynomials within the *basemul* function.

We observe that there are more leakage points for the `O0`-optimized implementation compared to the `O3`-optimized implementation, since the `O0`-optimized implementation manipulates the variables using much higher number of assembly instructions compared to the `O3`-implementation. We choose a suitable threshold $\mathsf{Th_{Sel}}$ to choose a PoI set, separately for m and y, which is a parameter of the experimental setup. We then use this PoI set to build HW classifiers separately for both x and y, using the Random Forest approach.

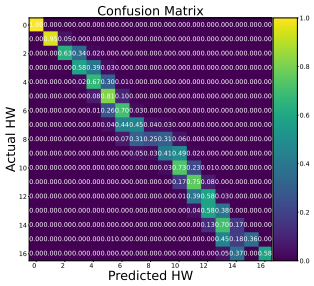
We represent the result of our HW classifier, in the form of a confusion matrix, where a given element (i, j) corresponds to the probability of actual HW = i and predicted HW = j. Thus, the confusion matrix for an ideal HW classifier is an ideal matrix I. Refer to Fig.3(a)-(d) for the confusion matrix plots obtained for the input and output coefficients corresponding to the first *basemul* operation in the pointwise-polynomial multiplication operation, from both the `O0` and `O3`-optimized implementations respectively. We can observe that the deviation from the diagonal is minimal, thereby demonstrating the accuracy of our HW classifier for both the `O0` and `O3`-optimized implementations.



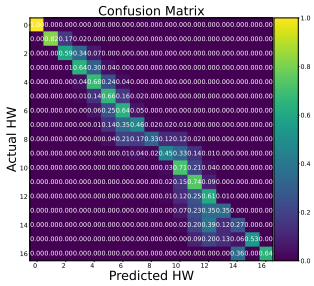
(a) Input m (`O0`-optimized)



(b) Output y (`O0`-optimized)



(a) Input m (`O3`-optimized)



(b) Output y (`O3`-optimized)

Fig. 3. Confusion Matrix of Predicted versus True HW, obtained using our constructed HW classifier on leakages from the `O0` and `O3` optimized reference implementations of Kyber KEM.

For the `O0`-optimized implementation, we achieved a high accuracy of 69-80% and 81-85% for the HW classifier for both the input and output coefficients respectively (Refer to Fig. 3(a)-(b)). Though we do not achieve a perfect HW classifier, we observe that the deviation is only minimal from the actual HW. As we show later in Section V-C, we are still able to recover all the secret coefficients, that we attempted to recover. For the `O3`-optimized implementation, we achieved a slightly lower accuracy between $\approx$ 70% and $\approx$ 72% for the HW classifier on the inputs and outputs respectively (Refer to Fig. 3(c)-(d)). This observation is expected given that the number of PoIs for the `O3`-optimized implementation is lower than that of the `O0`-optimized implementation. However, prior works have shown that it is possible to construct HW classifiers with much higher accuracy of over 90% [22], which also increases the capability of our attack.

C. Practical Evaluation of our Attack

We obtained a set of attack traces corresponding to an unknown key, and analyzed recovery of the first 20 coefficients, from the `O0`-optimized implementation. We used our HW classifier to recover the intermediates in the pointwise multiplication operation, and subsequently fed these HW values to construct the joint distribution leakage HWset_{attack} for all the 20 coefficients. Application of the Maximum Likelihood principle on HWset_{attack} enabled recovery of all the 20 coefficients using $35 - 5000$ traces. All coefficients cannot be recovered with the same number of traces due to the randomness in the constructed joint distribution leakage. Though we did not perform experiments to recover all the secret coefficients, our experiments provide sufficient evidence of the capability of our attack, in the presence of a sufficiently accurate HW classifier. We expect to perform key recovery with much fewer traces in the presence of HW classifiers with higher accuracy. We leave the improvement of our HW classifier accuracy out of the scope of our current work.

VI. Conclusion

In this work, we propose the first blind side-channel attack on Kyber KEM, targeting leakage from the pointwise multiplication in the decryption procedure, resulting in full key recovery. We performed practical validation of our attack using power side-channel from the reference implementation of Kyber KEM taken from the *pqm4* library [4], implemented on the ARM Cortex-M4 microcontroller. Our experiments clearly indicate the feasibility of our proposed attack in recovering the full key in only a few hundred to few thousand traces. We intend to explore assembly optimized implementations as well as side-channel protected implementations of Kyber, as part of future work.

REFERENCES

[1] Moody, D, "Post-Quantum Cryptography Standardization: Announcement and outline of NIST's Call for Submissions," in *International Conference on Post-Quantum Cryptography-PQCrypto*, 2016.

[2] Alagic, Gorjan and Apon, Daniel and Cooper, David and Dang, Quynh and Dang, Thinh and Kelsey, John and Lichtinger, Jacob and Miller, Carl and Moody, Dustin and Peralta, Rene and others, "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," National Institute of Standards and Technology, Tech. Rep., 2022.

[3] Avanzi, Roberto and Bos, Joppe W. and Ducas, Leo and Kiltz, Eike and Lepoint, Tancrede and Lyubashevsky, Vadim and Schanck, John and Schwabe, Peter and Seiler, Gregor and Stehlé, Damien, "CRYSTALS-Kyber (version 3.0): Algorithm specifications and supporting documentation (October 1, 2020)," *Submission to the NIST post-quantum project*, 2020.

[4] Kannwischer, Matthias J and Rijneveld, Joost and Schwabe, Peter and Stoffelen, Ko, "pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4," in *Second PQC Standardization Conference: University of California, Santa Barbara and co-located with Crypto 2019*, 2019, pp. 1–22.

[5] Abdulrahman, Amin and Hwang, Vincent and Kannwischer, Matthias J and Sprenkels, Daan, "Faster Kyber and Dilithium on the Cortex-M4," *Cryptology ePrint Archive*, 2022.

[6] Prasanna Ravi and Anupam Chattopadhyay and Anubhab Baksi, "Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results," *IACR Cryptol. ePrint Arch.*, p. 737, 2022. [Online]. Available: https://eprint.iacr.org/2022/737

[7] Bos, Joppe W and Gourjon, Marc and Renes, Joost and Schneider, Tobias and van Vredendaal, Christine, "Masking Kyber: First-and Higher-Order Implementations." *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 483, 2021.

[8] Gokulnath Rajendran and Prasanna Ravi and Jan-Pieter D'Anvers and Shivam Bhasin and Anupam Chattopadhyay, "Pushing the Limits of Generic Side-Channel Attacks on LWE-based KEMs - Parallel PC Oracle Attacks on Kyber KEM and Beyond," *IACR Cryptol. ePrint Arch.*, p. 931, 2022. [Online]. Available: https://eprint.iacr.org/2022/931

[9] Ravi, Prasanna and Roy, Sujoy Sinha and Chattopadhyay, Anupam and Bhasin, Shivam, "Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 307–335, 2020.

[10] E. B.-I. C. C. Specifications, "for payment systems, book 1: Application independent icc to terminal interface requirements, book 2: Security and key management, book 3: Application specification, book 4: Cardholder attendant and acquirer interface requirements, v. 4.3, emvco," 2011.

[11] Colin O'Flynn and Zhizhang (David) Chen, "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research," in *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, ser. Lecture Notes in Computer Science, E. Prouff, Ed., vol. 8622. Springer, 2014, pp. 243–260. [Online]. Available: https://doi.org/10.1007/978-3-319-10175-0_17

[12] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer and N. Homma, Eds. Cham: Springer International Publishing, 2017, pp. 513–533.

[13] Pessl, Peter and Primas, Robert, "More Practical Single-Trace Attacks on the Number Theoretic Transform," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2019, pp. 130–149.

[14] C. Mujdei, L. Wouters, A. Karmakar, A. Beckers, J. M. B. Mera, and I. Verbauwhede, "Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication," *ACM Transactions on Embedded Computing Systems*, 2022.

[15] B. Yang, P. Ravi, F. Zhang, A. Shen, and S. Bhasin, "Stamp-single trace attack on m-lwe pointwise multiplication in kyber," *Cryptology ePrint Archive*, 2023.

[16] E. A. Bock, G. Banegas, C. Brzuska, Ł. Chmielewski, K. Puniamurthy, and M. Šorf, "Breaking dpa-protected kyber via the pair-pointwise multiplication," *Cryptology ePrint Archive*, 2023.

[17] Ravi, Prasanna and Bhasin, Shivam and Roy, Sujoy Sinha and Chattopadhyay, Anupam, "On Exploiting Message Leakage in (few) NIST PQC Candidates for Practical Message Recovery Attacks," *IEEE Transactions on Information Forensics and Security*, 2021.

[18] Xu, Zhuang and Pemberton, Owen and Roy, Sujoy Sinha and Oswald, David, "Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems with Chosen Ciphertexts: The Case Study of Kyber," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 912, 2020.

[19] Hamburg, Mike and Hermelink, Julius and Primas, Robert and Samardjiska, Simona and Schamberger, Thomas and Streit, Silvan and Strieder, Emanuele and van Vredendaal, Christine, "Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 88–113, 2021.

[20] C. Clavier and L. Reynaud, "Improved blind side-channel analysis by exploitation of joint distributions of leakages," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 24–44.

[21] C. Clavier, L. Reynaud, and A. Wurcker, "Quadrivariate improved blind side-channel analysis on boolean masked aes," in *Constructive Side-Channel Analysis and Secure Design: 9th International Workshop, COSADE 2018, Singapore, April 23–24, 2018, Proceedings 9*. Springer, 2018, pp. 153–167.

[22] B. Colombier, V.-F. Drăgoi, P.-L. Cayrel, and V. Grosso, "Profiled side-channel attack on cryptosystems based on the binary syndrome decoding problem," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3407–3420, 2022.

[23] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[24] Brier, Eric and Clavier, Christophe and Olivier, Francis, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.