Power analysis attack on Kyber*

Alexandre Karlov[1] and Natacha Linard de Guertechin[1]

[1]Cysec SA, EPFL Innovation Park, Bâtiment D, 1015 Lausanne, CH

Abstract

This paper describes a practical side-channel power analysis on CRYSTALS-Kyber key-encapsulation mechanism. In particular, we analyse the polynomial multiplication in the decapsulation phase to recover the secret key in a semi-static setting. The power analysis attack was performed against the `KYBER512` implementation from pqm4 [1] running on STM32F3 M4-cortex CPU.

Keywords: Kyber, lattice-based cryptography, side-channel attacks, correlation power analysis

1 Introduction

The arrival of quantum computers will induce a problem for today's cryptography. Provided a quantum computer, Shor's algorithm [2] solves the problems of discrete logarithm, elliptic curve discrete logarithm and factorization of large numbers on which the RSA relies in polynomial-time. In order to respond to quantum computers, the National Institute of Standards and Technology (NIST) launched in 2016 the Post-Quantum Cryptography Standardization program to add post-quantum cryptography to their standards. Mid 2020, the final candidates were announced, including CRYSTALS-Kyber [3], a lattice-based key-encapsulation mechanism (KEM).

Since the late 90s, lattice-based cryptography has triggered a lot of interest. In 1997, Ajtai and Dwork [4] proposed a first lattice-based encryption scheme. Regev [5] then improved this scheme by introducing the notion of Learning With Errors (LWE) problem [6]. The LWE problem is the following: Given a matrix $A \in \mathcal{R}_q^{m \times n}$ and $As + e$ where $s \in \mathcal{R}_q^n$ is a secret vector and e is a random error vector, find s. This problem is assumed to be hard. In this paper, we focus on a lattice-based cryptography scheme named CRYSTALS-Kyber which is on the

*Released on September 14, 2021

finalists list of the NIST Standardization process. Kyber is a CCA-secure KEM based on the hardness of module-LWE which was built from a CPA-secure PKE scheme by applying the Fujisaki-Okamoto transform [7].

Despite the fact that post-quantum cryptosystems are resistant to theoretical attacks, they will be implemented in physical devices such as microcontrollers and will be subject to similar physical attacks as current cryptographic implementations, in particular side-channel attacks. Through side channel information such as timing, power consumption or electromagnetic radiation, an attacker can recover secret information if the cryptographic implementation does not take this kind of attacks into consideration and does not implement any countermeasure. In this paper, we exploit the information about power consumption to recover a secret key —a technique known as power analysis.

1.1 Contribution

In this work, we focus on the decapsulation phase of Kyber. We introduce a correlation power analysis [8] of the M4-Cortex implementation in the decapsulation phase of Kyber to recover the secret key.

1.2 Related works

Lattice-based cryptography is quantum resistant, however some implementations have already be shown to be vulnerable to side-channel attacks.

A power analysis on the encapsulation phase using a single trace is shown in [9]. The authors attack the message encoding in lattice-based KEMs to retrieve the message and then to obtain an ephemeral session key using the k-mean algorithms.

2 Preliminaries

Bytes: The notation $\mathcal{B}$ denotes the set of bytes. Then the set of byte array of length k (resp. arbitrary length) is denoted by $\mathcal{B}^k$ (resp. by $\mathcal{B}^\star$)

Polynomial rings: The ring $\mathbb{Z}[X]/(X^n+1)$ is denoted by $\mathcal{R}$ and the ring $\mathbb{Z}_q[X]/(X^n+1)$ by $\mathcal{R}_q$

Distribution: We denote the sampling x according to the distribution D by $x \leftarrow D$.
We define B_η as follows: sample $(a_1, ..., a_\eta, b_1, ..., b_\eta) \leftarrow \{0,1\}^{2\eta}$ and output $\sum_{i=1}^{\eta}(a_i - b_i)$.

Compression and Decompression [3]: The function $\texttt{Compress}_q(x, d)$ takes as input an integer $x \in \mathbb{Z}_q$ and outputs an integer in $\mathbb{Z}_{2^d-1}$ where $d < \lceil \log_2(q) \rfloor$.

The function $\texttt{Decompress}_q$ is defined such that

$$x' = \texttt{Decompress}_q(\texttt{Compress}_q(x, d)), d)$$

is an element close to x.

Hash function: We defined two hash functions $H : \mathcal{B}^\star \to \mathcal{B}^{32}$ and $G : \mathcal{B}^\star \to \mathcal{B}^{32} \times \mathcal{B}^{32}$, and a key-derivation function $\texttt{KDF} : \mathcal{B}^\star \to \mathcal{B}^\star$

3 Kyber

Kyber is a CCA-secure KEM, based on the hardness of solving the learning-with-errors (LWE) problem over module lattices [10]. Kyber is built from an IND-CPA secure public-key encryption scheme, called `Kyber.CPAPKE` by applying the Fujisaki-Okamoto transform [7].

Kyber can be used in a semi static setting. By convention the two participants will be Alice and Bob.

Semi-static setting

In a semi-static setting, Bob generates a *fixed* private key sk_B and a corresponding public key pk_B which is sent to Alice. Alice then encapsulates her private key sk_A, computes the shared secret K and sends to Bob her encapsulated key and her public key. Finally, Bob decapsulates Alice's private key in order to be able to compute the shared secret K.

If Alice and Bob want to establish additional shared secrets K, Bob keeps his pair of keys (sk_B, pk_B) and Alice generates a new pair of keys each time. This means that Bob's private key is fixed from one execution to another. This mode of operation is desirable because a fixed key allows a faster key exchange and less dependency on randomness —two desirable features, especially for embedded devices.

If an adversary can recover Bob's private key, then the shared secret for all communication can be recovered, performing thus a total break of confidentiality.

The goal of this paper is to show an attack to recover the secret key of Bob in a semi-static setting.

3.1 `Kyber.CPAPKE`

Key Generation:

During the key generation, a matrix A is generated with random elements. Two vectors s, e with small elements are generated according to the distributions B_{η_1} and B_{η_2}. The secret key is defined to be the vector s and the public key is the pair $(A, t = As + e)$.

Algorithm 1: `Kyber.CPAPKE.KeyGen()` from [3]

Output: Secret key sk
Output: Public key pk
Generate matrix $A \in \mathcal{R}_q^{k \times k}$
Sample $s \in \mathcal{R}_q^k$ from B_{η_1}
Sample $e \in \mathcal{R}_q^k$ from B_{η_2}
$pk := As + e$
$sk := s$

Encryption:

Three random vectors r, e_1, e_2 are generated with small elements. The output of the encryption algorithm is the pair (u, v) where $u = A^T r + e_1$ and $v = t^T r + e_2 + m$.

Algorithm 2: `Kyber.CPAPKE.Enc(`pk, m, r`)` from [3]

Input: Public key pk
Input: Message m
Input: Random coins r
Output: Ciphertext c
Generate matrix $A \in \mathcal{R}_q^{k \times k}$
Sample $r \in \mathcal{R}_q^k$ from B_{η_1}
Sample $e_1 \in \mathcal{R}_q^k$ from B_{η_2}
Sample $e_2 \in \mathcal{R}_q$ from B_{η_2}
$u := A^T r + e_1$
$v. = t^T r + e_2 + \texttt{Decompress}_q(m, 1)$
$c := (\texttt{Compress}_q(u, d_u) || \texttt{Compress}_q(v, d_v))$

Decryption:

To decrypt the ciphertext c, the message m is equal to $v - us$. Indeed,

$$\begin{aligned} v - us &= t^T r + e_2 + m - s^T (A^T r + e_1) \\ &= (As + e)^T r + e_2 + m - s^T (A^T r + e_1) \\ &= s^T A^T r + er + e_2 + m - s^T A^T r - s^T e_1 \\ &= m + e' \end{aligned}$$

where e' is a negligible error.

Algorithm 3: `Kyber.CPAPKE.Dec`(sk, c) from [3]

Input: Secret key sk
Input: Ciphertext c
Output: Message m

1 $u = \texttt{Decompress}_q(c_1, d_u)$
2 $v = \texttt{Decompress}_q(c_2, d_v)$
3 $m = \texttt{indcpa\_dec}(sk, m)$ // $m = v - sk^T u$
4 $m = \texttt{Compress}_q(m, 1)$

3.2 `Kyber.CCAKEM`

Kyber.CCAKEM is constructed from `Kyber.CPAPKE` using the Fujisaki-Okamoto transform.

Key Generation:

First, a random vector z is generated which will serve to create a random shared key K if the second participant tried to cheat. Then the secret key sk and the public key pk is generated by `Kyber.CPAPKE.KeyGen()`.

Algorithm 4: `Kyber.CCAKEM.KeyGen()` from [3]

Output: Secret key sk
Output: Public key pk

1 $z \leftarrow \mathcal{B}^{32}$
2 $(pk, sk') :=$`Kyber.CPAPKE.KeyGen()`
3 $sk := (sk'||pk||H(pk)||z)$

Key Encapsulation:

First, a random message is generated and encrypted using `Kyber.CPAPKE.Enc`(pk). The shared secret is computed by $KDF(\overline{K}||H(c))$.

Algorithm 5: `Kyber.CCAKEM.Enc`(pk) from [3]

Input: Public key pk
Output: Ciphertext c
Output: Shared key K

1 $m \leftarrow \mathcal{B}^{32}$
2 $m \leftarrow H(m)$
3 $(\overline{K}, r) := G(m||H(pk))$
4 $c :=$ `Kyber.CPAPKE.Enc`(pk, m, r)
5 $K := KDF(\overline{K}||H(c))$

Key Decapsulation:

The ciphertext is decrypted using `Kyber.CPAPKE.Dec`(s, c). Then the ciphertext is recomputed to verify if the second participant tried to cheat or not. Finally, the shared secret is computed depending on the fact whether the ciphertext is legitimate or not.

After receiving the ciphertext $c = (u||v)$, the message is extracted by $m' = v - s^T u$, then the secret r' is computed. The ciphertext is checked, i.e., if $c' = c$ then the shared secret $K = KDF(\overline{K}'||H(c))$, otherwise $K = KDF(z||H(c))$.

Algorithm 6: `Kyber.CCAKEM.Dec`(sk, c) from [3]

Input: Secret key sk
Input: Ciphertext c
Output: Shared key K

1 $pk := sk + 12kn/8$
2 $h := sk + 24kn/8 + 32$
3 $z := sk + 24kn/8 + 64$
4 $m' :=$`Kyber.CPAPKE.Dec`(s, c)
5 $(\overline{K}', r') := G(m'||h)$
6 $c' =$`Kyber.CPAPKE.Enc`(pk, m', r')
7 **if** $c = c'$ **then**
8 **return** $K := KDF(\overline{K}'||H(c))$
9 **else**
10 **return** $K := KDF(z||H(c))$

3.3 Correlation power analysis on Kyber

Attack on Kyber

The goal of the attack is to recover the secret key during the step 3 of the key decapsulation, specifically at the line 1 in `Kyber.CPAPKE.Dec`(sk, c), which is a multiplication of two polynomials in the NTT domain. The inputs of this function are the secret polynomial sk and a part of ciphertext c. Bob generates a fixed pair (sk_B, pk_B) of keys using the Algorithm 4. Then, using the Algorithm 5 Alice generates a random key, computes a ciphertext and sends it to Bob. Bob decapsulates the ciphertext with his fixed secret key using the Algorithm 6.

In a semi-static setting, Bob computes the multiplication of the secret fixed polynomial with different ciphertexts in line 1 of Algorithm 3. Our goal is to correlate the power traces of the polynomial multiplication that involves the secret key with the Hamming weight of the different output of this operation

with all possible keys.

Correlation power analysis

A correlation power analysis (CPA) [11] is a side channel power analysis attack based on the statistical correlation between outputs of a function and the corresponding power traces.

In this paper, we concentrate on a vertical CPA. A vertical CPA focuses on a fixed secret by collecting the power traces that correspond to the execution of an operation with the fixed secret and several different values.

In a power analysis attack, the attacker can measure the power consumption of the target. In this case the target device computes the decapsulation algorithm in a black-box way. The adversary listens to the communications between Bob and Alice, i.e., the attacker has the public parameters, Bob's public key and Alice's ciphertext.

The attacker then correlates the power traces and the potential outputs of the polynomial multiplication with different possibles keys to see which one corresponds the best using the Pearson's Correlation Coefficient (PCC).

The steps of the CPA attack can be summarized as:

1. Find an operation in the attacked algorithm that involves a secret value and a known (public) value.
2. Collect n power samples of this operation with different known inputs.
3. Take a guess on the secret values and compute the intermediate values for all the known inputs.
4. Correlate the Hamming weight of the intermediate values and the power traces.
5. Repeat steps 3-4 for all the possible secret values, the secret value with the highest correlation might be the good candidate value.

CPA on Kyber

In this section, we explain our CPA on Kyber. In the decapsulation Algorithm 6, the multiplication between a part of the secret key and a part of ciphertext are done $k \cdot n$ times, independently. The goal of the CPA on KYBER is to recover Bob secret key. The operation `basemul` in Algorithm 9 is the lowest operation that involves a part of the key and a part of the known ciphertext. The operation `basemul` is used for the multiplication of polynomials. In our attack on Kyber, we measure the power consumption of the operation `basemul` for each

iteration. We can remark that each `basemul` is independent and can be computed separately. Bob's secret key is partitioned in $2k \cdot \frac{n}{8}$ bytes parts and each part will be used in the operation `basemul`.

We focus on the first `basemul` operation. For other subsequent operations, the same idea is applied. The `basemul` takes as input two arrays of 2x2 bytes and outputs the multiplication of these two arrays. In a semi-static setting, the attacker intercepts N Alice's ciphertexts and measures the power consumption for the `basemul` operation. Then he computes the results of the `basemul` with all the possible parts of the key (i.e., 2^{16} possibilities). The guessed part of the key with the highest correlation between the Hamming weight of the intermediate results and the power traces is the one which is kept.

Algorithm 7: `indcpa_dec`(sk, c) from [1]

Input: Secret key sk
Input: Ciphertext c
Output: Message m

$mp =$ `polynomial_multiplication`$(c[0], sk[0])$
  // sk is a vector of polynomials, $sk[i]$ is the ith polynomial in sk, same for c
for $i = 1$ *to* k **do**
  $bp =$ `polynomial_multiplication`$(c[i], sk[i])$
  $bp = mp + bp$
$v =$ `polynomial_decompress`(v, c)
$mp = v - mp$
$m =$ `polynomial_tomessage`(mp);

Algorithm 8: `polynomial_multiplication`(a, b) from [1]

Input: Secret polynomial a
Input: Polynomial part of the ciphertext b
Output: Polynomial m

for $i = 1$ *to* $\frac{n}{4}$ **do**
  $m[4 \cdot i : 4 \cdot i + 4] =$
   `doublebasemul`$(b[4 \cdot i : 4 \cdot i + 4], a[4 \cdot i : 4 \cdot i + 4], zetas[i])$

Algorithm 9: `doublebasemul`(a, b) from [1]

Input: Array of 4x2 bytes a
Input: Array of 4x2 bytes b
Input: Constant zeta
Output: Array of 4x2 bytes c

$c[0:2] =$`basemul`$(a[0:2], b[0:2], zeta)$
$c[2:4] =$`basemul`$(a[2:4], b[2:4], -zeta)$

4 Experimental Results

We implemented the key recovery attack on a STM32F3 board as target which runs an unprotected instance of `KYBER512` from the library pqm4 [1].

4.1 Setup

The ChipWhisperer Pro toolkit [12] was used for the attack. The toolkit includes:

- A STM32F3 board with an ARM Cortex-M4 micro-controller
- The ChipWhisperer-Pro with CW308 UFO target

The implementation used is a `KYBER512` implementation adapted to Cortex-M4 from pqm4 [1]. The only modification done is the addition of a trigger before and after the `doublebasemul` operations to help the alignement of traces.

Moreover, we attacked Kyber with the lowest security parameter but the attack works even for the highest security parameter. The only difference between them is that there are more `basemul` operations to do.

The leakage model is the leakage in Hamming Weight, which means that the power consumption of the instructions of the microcontroller is expected to be proportional to the results of the processed operations. Indeed, the power consumption is linked to the number of bits updated in a register.

4.2 Traces collections

We recorded 200 power traces of 1000 points with the corresponding inputs for each `doublebasemul`.

To collect the traces, the attacker does the following steps:

1. Bob's pair keys is generated using Algorithm 4 on our personal computer and Bob's secret key is sent to the target board.
2. On the computer, Alice generates N ciphertexts using Algorithm 5 and Bob's public key.
3. When a ciphertext is sent, the target decaps the ciphertext using the Algorithm 6 and the oscilloscope collects the power traces of each `basemul` in Algorithm 7 operations.

Let T_j^i denote the ith point of the jth trace and T_i be the vector such that

$$T_i = [T_i^1, ..., T_i^{200}]$$

In our scenario, we used 200 different ciphertexts. But we will see later that the attack works with less ciphertexts being intercepted.

4.3 Attack procedure

We performed a vertical CPA on each `basemul` operations in 7. Each `basemul` operation takes as input 2×4 bytes of Bob's secret key. For each iteration of the loop, we make a guess for the last 4 bytes, hence 4^8 possibilities.

In Kyber from pqm4, the function `doublebsemul` is a function written in assembly which computes the two `basemul`. It takes as inputs a part of the key $kk' = k_0k_1k_2k_3\ k'_0k'_1k'_2k'_3$ (2×4 bytes) and a part of the ciphertext $cc' = c_0c_1c_2c_3\ c'_0c'_1c'_2c'_3$ (2×4 bytes), and computes `basemul`$(k, c, zeta)$ and `basemul`$(k, c, -zeta)$. We focus on the computation of `basemul`$(k, c, zeta)$ which is computed from the line 25 of `doublebasemul_asm` in Algorithm 1. We want to guess the 4 bytes of $k = k_0k_1k_2k_3$.
The steps of the attack are the following:

1. Make a guess for k_2k_3 (2^{16} possibilities) and compute the result $rst = [rst_0, ..., rst_{200}]$ where rst_i is the Hamming weight of the operation `smultt` on line 25 of `doublebasemul` using the ith ciphertext.
2. Compute Pearson correlation coefficient between T_i and rst for all i and keep the biggest value in absolute $PCC_{k_2k_3}$.
3. Repeat step 1-2 for all possibilities of k_2k_3 and keep a sample $S = \{k_2k_3$ such that $PCC_{k_2k_3} > x\}$. For example, $x = 0.6$,
4. Fix $k_2k_3 \in S$, make a guess for k_0k_1 and compute the result rst' of `pkhtb` for all of the ciphertexts. Then compute Pearson correlation between rst' and the power traces, keep the largest value in absolute $PCC_{k_0k_1k_2k_3}$.
5. Redo step 4 for all the $k_0k_1 \in S$.
6. The part of the key $k_0k_1k_2k_3$ with the largest $PCC_{k_0k_1k_2k_3}$ is the correct guess.

At the step 3, it may be that the right part of the key does not have the largest correlation but it is in the largest ones. For example, the Figure 1 shows the PCC of the first operation using the good part of key (in blue) and in a false one (in red). Then we will compute the step 4 for those two possibilities. The Figure 2 shows that the good 2×4 bytes will have a large correlation (in blue) but the wrong key will not have this strong correlation even if the correlation at step 3 was the biggest.

The same procedure can be done to recover the other 4 bytes of the keys used in `doublebasemul` as well as all the bytes of Bob's secret key. On our PC, the time to recover 2×4 bytes is about 30 minutes. The attack takes few days to recover all the bytes of the key. The inputs of each `doublebasemul` are independent. A speedup through parallelization can be done to improve the complete key recovery. We have managed to recover the secret key in 2 hours thanks to parallelization.

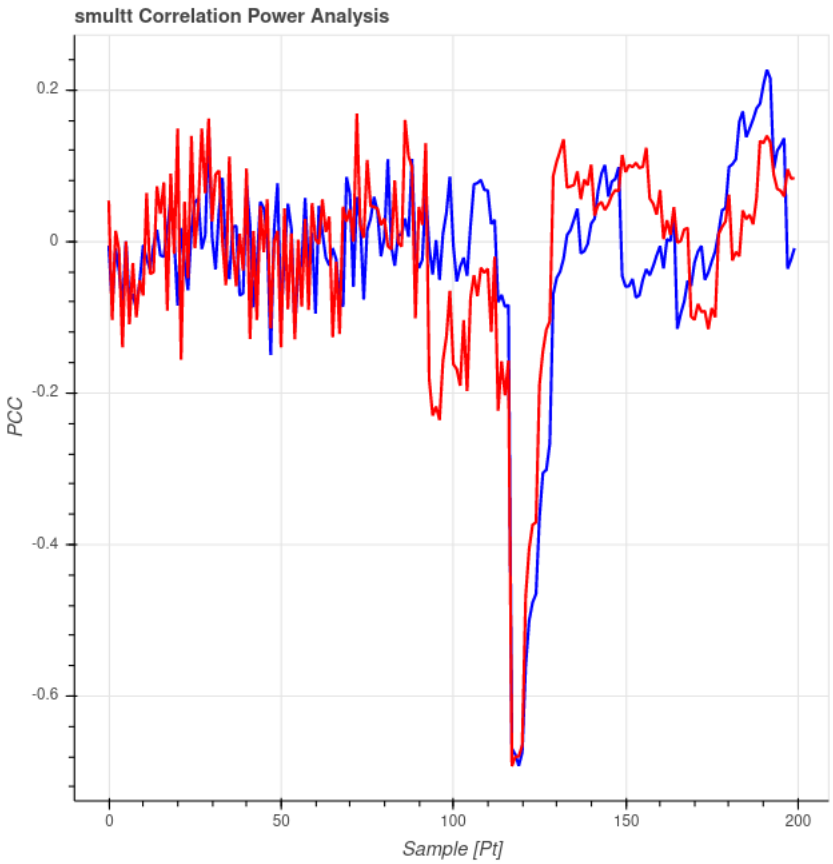

Figure 1: CPA on `smultt` operation

4.4 Minimal number of required traces

In our case we performed 200 ciphertext captures which was sufficient. The goal of this section is to find the minimum required number of ciphertexts to get 100% accurate recovery of the key.um of traces required We focused on first operations and did the follow steps:

1. Generate N random ciphertexts.
2. Collect the power traces during the first `basemul` in the decaps phase for the N ciphertexts.
3. Repeat the steps 1-2 20 times and count how many times the right part of the key is found.

We take $N \in \{200, 150, 100, 85, 80, 75, 50, 40\}$.

The graph on Figure 3 shows that if we have less than 80-100 traces, the

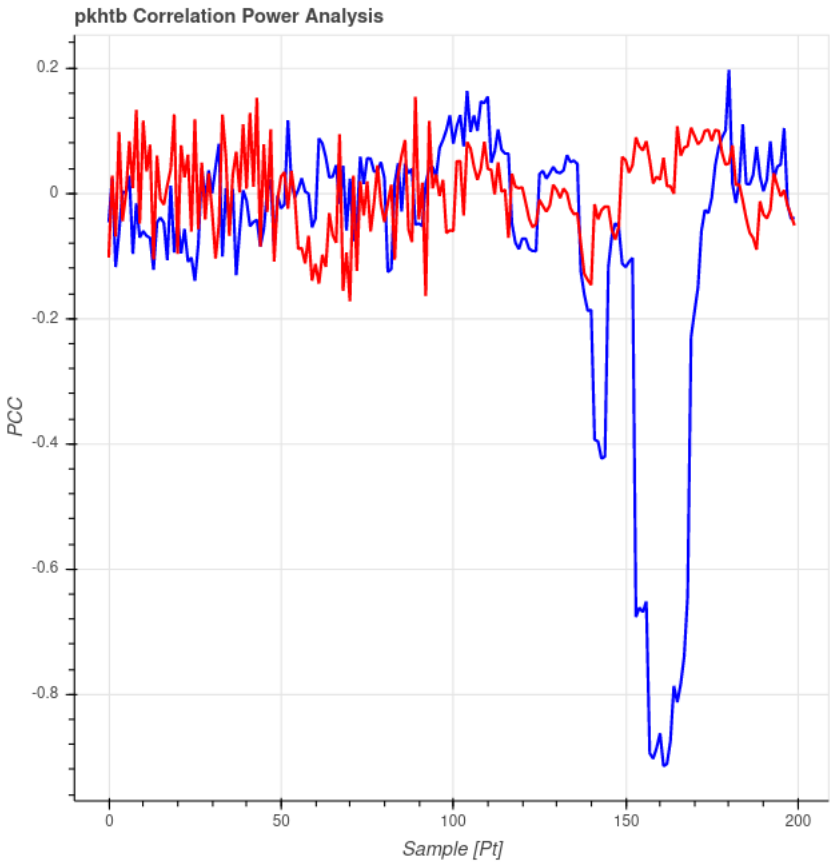

Figure 2: CPA on `pkhtb` operation

chances of finding the right part of the key is lower.

5 Countermeasure

One simple and straightforward countermeasure consists in avoiding using Kyber in a semi-static setting. Another countermeasure is to regenerate Bob's key every x communication where $x < 50$.

6 Conclusion

In this paper, we proposed a successful and practical correlation power analysis attack on `KYBER512` implementation from pqm4 to recover the secret key in the decapsulation phase. With a sufficient number of traces, the attack is $> 99\%$

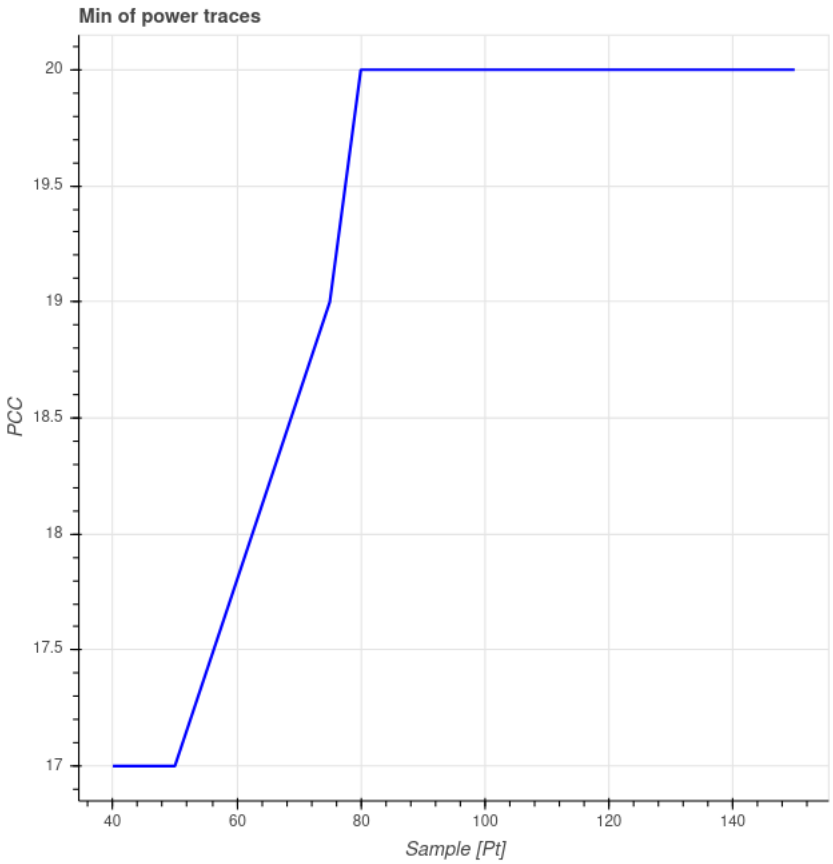

Figure 3: Minimum of traces required

successful and accurate. We recommend to use Kyber only in ephemeral setting to protect the confidentiality of the secret key. In the future works we may think of finding other countermeasures or figuring out if it is possible to adapt this attack to a correlation power analysis using only a single trace.

Acknowledgement

We would like to thank Dr. Andres Upegui from University of Applied Sciences Western Switzerland, HEPIA for his help, especially in providing the equipment to permit this attack implementation.

7 Appendix information

Listing 1: `doublebasemul_asm` function from [1]

```
doublebasemul_asm:
push {r4-r11, lr}

rptr  .req r0
aptr  .req r1
bptr  .req r2
zeta  .req r3
poly0 .req r4
poly1 .req r6
poly2 .req r5
poly3 .req r7
q     .req r8
qinv  .req r8
tmp   .req r9
tmp2  .req r10
tmp3  .req r11

movw  q, #3329
movt qinv, #3327

ldrd poly0, poly2, [aptr], #8
ldrd poly1, poly3, [bptr], #8

//basemul(r->coeffs + 4 * i, a->coeffs + 4 * i, b->coeffs + 4 * i, zetas[64 + i]);
smultt tmp, poly0, poly1
montgomery q, qinv, tmp, tmp2
smultb tmp2, tmp2, zeta
smlabb tmp2, poly0, poly1, tmp2
montgomery q, qinv, tmp2, tmp
// r[0] in upper half of tmp

smuadx tmp2, poly0, poly1
montgomery q, qinv, tmp2, tmp3
// r[1] in upper half of tmp3
pkhtb tmp, tmp3, tmp, asr#16
str tmp, [rptr], #4

neg zeta, zeta

//basemul(r->coeffs + 4 * i + 2, a->coeffs + 4 * i + 2, b->coeffs + 4 * i + 2, - zetas
smultt tmp, poly2, poly3
montgomery q, qinv, tmp, tmp2
smultb tmp2, tmp2, zeta
smlabb tmp2, poly2, poly3, tmp2
montgomery q, qinv, tmp2, tmp
// r[0] in upper half of tmp

smuadx tmp2, poly2, poly3
montgomery q, qinv, tmp2, tmp3
// r[1] in upper half of tmp3
pkhtb tmp, tmp3, tmp, asr#16
str tmp, [rptr], #4
```

```
        pop {r4-r11, pc}
```

References

[1] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.

[2] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[3] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-Kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2:4, 2017.

[4] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293, 1997.

[5] Oded Regev. New lattice-based cryptographic constructions. *Journal of the ACM (JACM)*, 51(6):899–942, 2004.

[6] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 7(30):11, 2010.

[7] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference*, pages 537–554. Springer, 1999.

[8] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[9] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on message encoding in lattice-based KEMs. *IEEE Access*, 8:183175–183191, 2020.

[10] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

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

[12] NewAE Technology Inc. CHIPWHISPERER — NewAE Technology, 2021. `https://www.newae.com/chipwhisperer`.