Template and CPA Side Channel Attacks on the Kyber/ML-KEM Pair-Pointwise Multiplication

Sedric Nkotto

SRC Security Research & Consulting GmbH, Bonn, Germany
sedric.nkotto@src-gmbh.de

Abstract.
Kyber a.k.a ML-KEM is a quantum-safe Key Encapsulation Mechanism, that has been standardized by NIST under FIPS-203 and will definitely in the coming years be implemented in several commercial products. Following fors instance this report https://radar.cloudflare.com/adoption-and-usage?dateRange=52w, one can notice that since the end of 2024, the post-quantum encrypted share of the HTTPS request traffics has been increasing and is about to reach 50% now. However, the resilience of implementations against side channel attacks is still an open and practical concern. One of the drawbacks of the ongoing side channel analysis research related to PQC schemes is the availability of open-source datasets. Luckily some open-source datasets start popping up. For instance, the one recently published by Rezaeezade *et al.* in the paper 2025/811. This dataset captures power consumption during a pair-pointwise multiplication occurring in the course of ML-KEM decapsulation process and involving the decapsulation (sub)key and ciphertexts. In this paper we present both a profiled (template) and an unprofiled (CPA) side channel attacks targeting that pair-pointwise multiplication, which yield, in both cases, a complete recovery of the decapsulation secret (sub)key.

Keywords: Template Side-Channel Attack, Correlation Power Attack, Post Quantum Cryptography, Kyber, ML-KEM

Contents

Introduction

The NIST initiated in 2016 a standardization program to select algorithms resilient against attackers possessing a cryptographically relevant quantum computer. The program delivered the first standards in 2024, which encompass among others FIPS-203 (ML-KEM). This is the first step towards deployment for public use. But before deploying a given implementation, it is crucial to guarantee its resilience against side channel attackers. This paper presents a profiled and an unprofiled SCA attack on the ML-KEM pair-pointwise multiplication. The latter represents the coefficient-wise multiplication of two polynomials in the NTT-domain and is implemented by the `PQCLEAN-MLKEM768-basemul` subroutine in the reference implementation of ML-KEM (cf. PQClean Github repository[1]). Our attacks target the dataset provided in the recently published paper [3] by Rezaeezade *et al.* This dataset contains power measurements captured during a "pair-pointwise multiplication" in the decapsulation procedure of ML-KEM, running on an STM32F3 micro-controller at 7.372 MHz. For more details how the product of a pair of polynomial coefficients in the NTT domain are computed in ML-KEM, one can also consider the algorithm `BaseCaseMultiply` in [2, sec. 4.3.1, algorithm 12]. We also recall that the considered implementation is a pure C one and the dataset used an O3 level optimization. More information about the dataset can be found in [3].

1 ML-KEM

1.1 A simplified description of ML-KEM

ML-KEM is a quantum-safe lattice-based Key Encapsulation Mechanism standardized in 2024 by the US National Institute of Standards and Technology (NIST) under FIPS-203. ML-KEM encompasses three main algorithms namely,

1. **KeyGen: Key Generation**, which is used by one of the parties (we call **Amadou**) to produce a public key (**pk**) and a secret key (**sk**). The public key **pk** is subsequently sent to the other party (we call **Bineta**).
   Let us consider the quotient ring $R_q = \mathbb{Z}_q[X]/\left(X^{256}+1\right)$, with $q = 3329$.
   In order to produce **sk** and **pk**, Amadou pseudo-randomly samples a matrix $A \in R_q^{k\times k}$ and "small" vectors $s,\ e \in R_q^k$. The key pair (**sk**, **pk**) are defined as
   $\mathbf{sk} = s$ and $\mathbf{pk} = (A, t)$ with $t = A \cdot s + e(\mathrm{mod} q)$.

2. **Encapsulation: Encaps**
   Bineta uses Amadou's public key **pk**, to encapsulate a shared secret key m by encoding it into a polynomial in R_q and computing a ciphertext **c** from it. To do so, Bineta samples "small" random vectors $r,\ e_1 \in R_q^k$ and $e_2 \in R_q$ and computes the following:

$$u = A^T \cdot r + e_1(\mathrm{mod} q) \text{ and } v = t^T \cdot r + e_2 + Encode(m) \cdot \left\lfloor \frac{q}{2} \right\rfloor (\mathrm{mod} q)$$

   She subsequently sends $\mathbf{c} = (u, v)$ to Amadou.

3. **Decapsulation: Decaps**
   Amadou uses his secret key $\mathbf{sk} = s$ together with the ciphertext $\mathbf{c} = (u, v)$ received from Bineta, to produce the shared secret key m. He does it by computing

$$v' = v - \boxed{u^T \cdot s(\mathrm{mod} q)}$$

[1] https://github.com/pq-crystals/kyber

After decoding v', Amadou recovers the shared secret m. “Decoding” here means, coefficients of v' close to zero are decoded to 0 and the ones close to $\lfloor \frac{q}{2} \rceil$ are decoded to 1.
Remark: As mentioned before, this is a simplified description. In a more detailed one, the actual shared key is obtained from m by computing

$$K = \text{SHA3-512}\left(m \| \text{SHA3-256}(\mathbf{pk})\right)$$

Spoiler: In the subsequent sections, our side channel attacks will be targeting the above-highlighted intermediate value.

1.2 Polynomial multiplication in ML-KEM

Let us recall that all scalar products of vectors in R_q^k (e.g. $u^T \cdot s$) involve polynomials multiplications in R_q. These are among the most expensive operations in ML-KEM. To speed them up, a technique called Number Theoretic Transform (NTT) is used. This is the modular arithmetic version of the Fast Fourier Transform (FFT).
Let $f = f_0 + f_1 X + \cdots + f_{255} X^{255}$ be a polynomial in R_q, which can also be represented as the tuple $(f_0, f_1, \cdots, f_{255})$. The NTT-representation of f is given by

$$NTT(f) = \left(\widehat{f_{0,0}} + \widehat{f_{0,1}}X,\ \widehat{f_{1,0}} + \widehat{f_{1,1}}X,\ \cdots,\ \widehat{f_{127,0}} + \widehat{f_{127,1}}X\right)$$

where $\widehat{f_{i,0}} + \widehat{f_{i,1}}X = f \mod (X^2 - \zeta^{2BitRev_7(i)+1})$, with $\zeta = 17 (\mathrm{mod} q)$ a primitive 256-th root of unity $\mathrm{mod} q$ and $BitRev_7(i)$ the bit reverse of the 7-bit representation of i.

The NTT-transform has an inverse transformation $INTT$ and given two polynomials $f = f_0 + f_1 X + \cdots + f_{255} X^{255}$ and $g = g_0 + g_1 X + \cdots + g_{255} X^{255}$ in R_q, we have

$$f \times g = INTT\left(NTT(f) \circ NTT(g)\right)$$

$NTT(f) \circ NTT(g)$ is a component-wise multiplication. That means the i-th component of this product is

$$(\widehat{f_{i,0}} + \widehat{f_{i,1}}X) \cdot (\widehat{g_{i,0}} + \widehat{g_{i,1}}X) \quad \mod (X^2 - \zeta^{2BitRev_7(i)+1})$$

which yields after reduction

$$\widehat{f_{i,0}} \cdot \widehat{g_{i,0}} + \widehat{f_{i,1}} \cdot \widehat{g_{i,1}} \zeta^{2BitRev_7(i)+1} + \left(\widehat{f_{i,0}} \cdot \widehat{g_{i,1}} + \widehat{f_{i,1}} \cdot \widehat{g_{i,0}}\right) X$$

This operation is performed in FIPS-203 by the `BaseCaseMultiply` subroutine (cf. [2, Algorithms 11 and 12]), also called `basemul`[2] in the reference implementation we will be considering in the next sections.

2 Profiled SCA Attack

We perform in this section a profiled template-based DPA attack on the above-mentioned ML-KEM Implementation Under Test (IUT) targeting the power traces dataset published in [3].

[2] https://github.com/pq-crystals/kyber/blob/main/ref/ntt.c#L139-L146

2.1 Dataset

The authors highlighted that the measurements in the dataset focused on the pairwise coefficient multiplication in the NTT domain implemented by the **basemul** function below (cf. [3, sec. 2, Listing 1.])

```
void basemul(int16_t r[2], const int16_t a[2], const int16_t b[2],
    int16_t zeta)
{
1:  r[0]  = fqmul(a[1], b[1]);
2:  r[0]  = fqmul(r[0], zeta);
3:  r[0] += fqmul(a[0], b[0]);
4:  r[1]  = fqmul(a[0], b[1]);
5:  r[1] += fqmul(a[1], b[0]);
}
```

Listing 1: Kyber pair-pointwise multipllication in NTT Domain

We recall that `fqmul(a,b)` outputs the product of `a` and `b` followed by a Montgomery reduction[3].
Apart from the raw traces, the dataset provides

- the coefficients `a[0]` and `a[1]` of the secret key (fixed for all measurements),
- the coefficient `b[1]` of the ciphertext for each measurement,
- and the corresponding outputs of the multiplications `fqmul(a[0], b[1])` and `fqmul(a[1], b[1])`).

2.2 Targets

As we already announced in sec.1.1, our attack vector is $u^T \cdot s(\bmod q)$. More precisely, the component-wise multiplications of their NTT-representations.
We recall that u and s are vectors in R_q^k. i.e. the scalar product $u^T \cdot s(\bmod q)$ involves k polynomial multiplications. For each polynomial multiplication in the NTT-domain, 128 such multiplications are computed.

$$(a_0 + a_1 X) \cdot (b_0 + b_1 X) \mod (X^2 - \zeta)$$

which after reduction yields

$$a_0 \cdot b_0 + a_1 \cdot b_1 \zeta + (a_0 \cdot b_1 + a_1 \cdot b_0) X$$

That means, to recover the complete ML-KEM secret key, one should attack overall $128 \cdot k$ such operations, which are all realised by the `basemul` subroutine in our considered reference implementation. Throughout this paper, we only attack one of them because the available dataset only captures the power trace during a single one. Nevertheless, our attack can be replicated if the complete power trace is available. The advantage is also that, all the $128 \cdot k$ operations can be attacked in parallel. Therefore, as long as the attack complexity is concerned, there is no significant difference between attacking a single operation or all the $128 \cdot k$ ones .
We specifically target the secret sub-keys `a[0]` and `a[1]` in the operations

`fqmul(a[0], b[1])` and `fqmul(a[1], b[1])`

It is also worth mentioning that `a[0]`, `a[1]`, `b[0]` and `b[1]` are lying in the field $\mathbb{Z}_q$. This already provides information about the key-space of the secret sub-keys `a[0]` and `a[1]`.

[3] https://github.com/pq-crystals/kyber/blob/main/ref/ntt.c#L68-L69

2.3 Leakage detection

The traces were compressed with a factor of 10, i.e. by taking the mean over sections of 10 sample points for each trace. The traces values were also rescaled to 16-bit unsigned integers. No further pre-processing of traces was applied.
Figure 1 and figure 2 below show the correlation plots with respect to the Hamming weight of the MSByte of the intermediate values `fqmul(a[0], b[1])` resp. `fqmul(a[1], b[1])`.
Note: we computed the intermediate values `fqmul(a[0],b[1])` and `fqmul(a[1],b[1])` on-the-fly while performing the attack and did not use the values stored in the dataset.

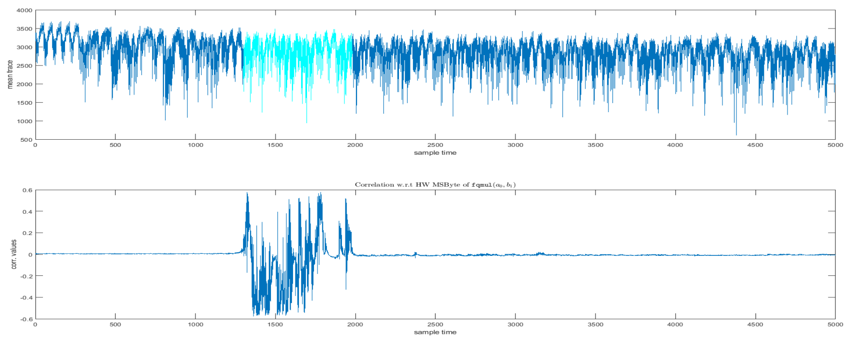


Figure 1: Correlation w.r.t. MSByte of `fqmul(a[0], b[1])`

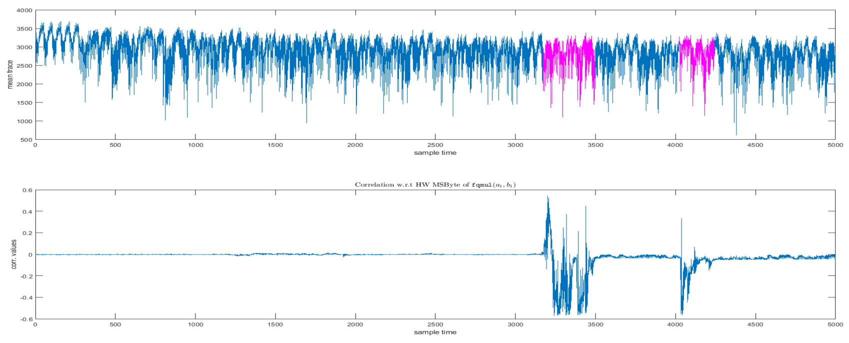


Figure 2: Correlation w.r.t. MSByte of `fqmul(a[1], b[1])`

One clearly observes from the correlation plots that the above-mentioned intermediate values are susceptible to correlation analysis and the regions where the corresponding operations occur are easily identifiable. For the specific case of `fqmul(a[1], b[1])`, two distinct regions of interest can be seen. This is probably due to the fact that for the considered dataset, `b[0]`= `b[1]`, which makes `fqmul(a[1], b[1])` to be computed twice (cf. Listing 1, line 1 and 5)

2.4 Template Attack

As mentioned before, we intend to perform a template attack by leveraging the leakage we found, with the aim of recovering the secret (sub)keys `a[0]` and `a[1]`.

The available traces were first of all split into two parts. 80% of the traces were used for profiling and 20% for matching/attack phase. The profiling traces were grouped into three classes depending on the Hamming weight of the MSByte of the outputs `fqmul(a[0], b[1])` and `fqmul(a[1], b[1])` as mentioned below:

Table 1: Traces shares

class	**characteristic (`i=0 or 1`)**
class0	HW(`MSByte of fqmul(a[i], b[1])`) $=$ 0 *or* 3
class1	HW(`MSByte of fqmul(a[i], b[1])`) $= 1$
class2	HW(`MSByte of fqmul(a[i], b[1])`) $= 2$

The templates were per class constructed as the means of the belonging traces within each class. For the Points of Interest selection, the SOSD (Sum of Square Difference) was computed, which formula for the three classes is $(\mu_0 - \mu_1)^2 + (\mu_0 - \mu_2)^2 + (\mu_1 - \mu_2)^2$, whereby μ_i represents the mean of the *class i*.
The figures below show the SOSD plots for the above-mentioned classes on top and the corresponding means at PoIs or RoIs at the bottom:

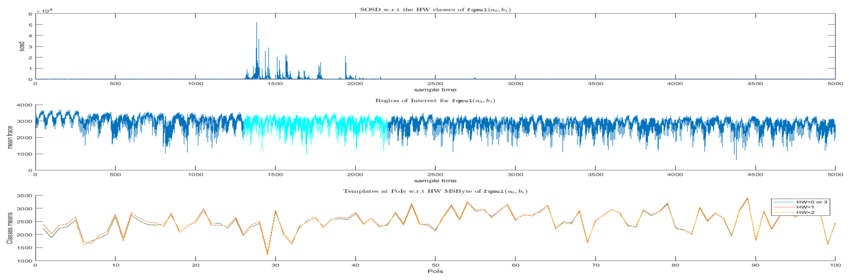


Figure 3: SOSD and templates w.r.t. `fqmul(a[0], b[1])`

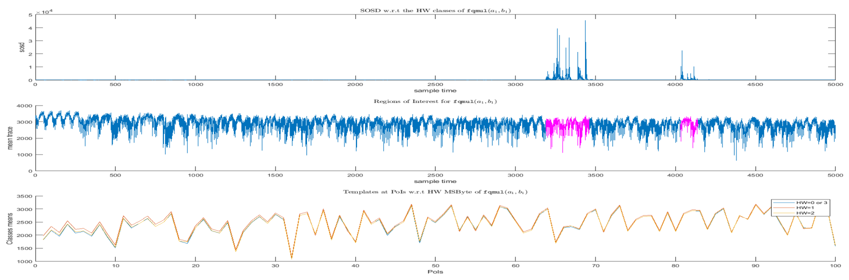


Figure 4: SOSD and templates w.r.t. `fqmul(a[1], b[1])`

Clear spikes can be observed in the SOSD plots (cf. plottings on top in figures 3 and 4), which highlight the sample times at which the three classes are better distinguishable. As one can observe on the plots at the bottom in figures 3 and 4, the mean traces of the classes are distinguishable at the PoIs.

2.5 Key recovery by reduced template attack

We further perform a template attack using the classes mentioned in the previous section. A template here consists only of the mean trace (for each group). We choose as covariance matrix the identity matrix. This is called in the literature reduced template analysis or the least square method (cf. [1, section 5.3]).
The score of a given trace in the class i, is therefore computed by the formula

$$score(t, i) = (t - \mu_i) \cdot (t - \mu_i)^T \tag{1}$$

Where t is the corresponding trace vector and μ_i the mean as mentioned above.
Let us for instance describe the attack procedure targeting the secret sub-key `a[0]`.
As mentioned before, 80% of traces (i.e. 80.000 traces) are used as profiling traces and the remaining 20.000 traces as attack traces.

Procedure (profiling phase):

1. Share the profiling traces into three classes with respect to the Hamming weight of the MSByte of the `fqmul(a[0], b[1])` output (cf. table 1).
2. Compute the mean for each classes (μ_0, μ_1 and μ_2)
   Your profiling phase is done.

Procedure (matching phase):

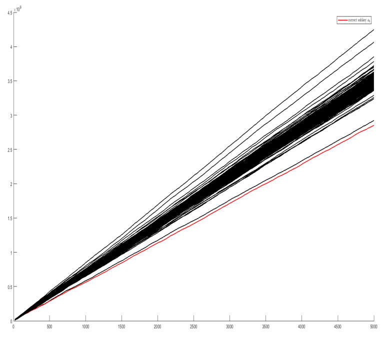
1. Consider (part of) the remaining traces kept aside for the matching phase as mentioned above.
2. To speed up your matching phase, a scores look-up table (LUT) can be precomputed. i.e. for each trace, compute the scores (see formula (1)) of that trace using each of the templates and store your scores LUT.
3. Run through the sub-key space. In the case of ML-KEM, `a[0]` $\in \mathbb{Z}_q$, which means the sub-key space is $[0,\ q) = [0,\ 3329)$.
4. For each possible guess of the (ML-KEM decryption) sub-key `a[0]` , sort the traces according to the Hamming weight of **MSByte (`fqmul(guesses,b[1])`)**.
   Compute subsequently the score of that guess by summing up the score of the traces accordingly. The previously pre-computed scores Lookup Table speeds up this step.
5. pick up the guess with the smallest score.

Remark: For the matching phase, we considered the traces and the templates only at specific points of interests (PoIs). We selected our PoIs by the highest SOSD values.

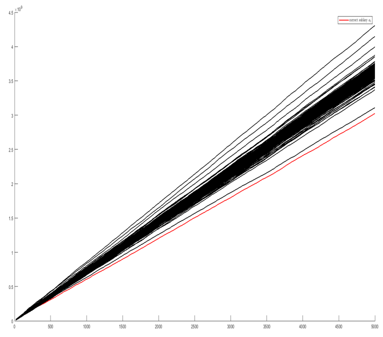
2.6 Results

As shown by the figures 5.1 and 5.2 below, the template attack yields the complete recovery of the ML-KEM decapsulation sub-keys. In both cases, the best guess, i.e. the one with the smallest score, matches the correct sub-key.

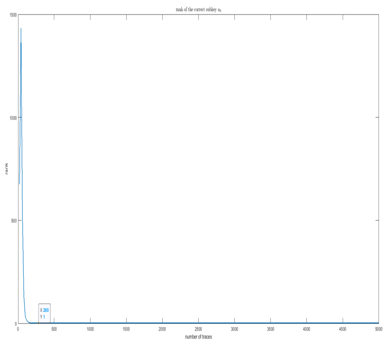
We can notice in the figures 6.1 and 6.2 below that only 340 traces are necessary to recover `a[1]` and even lower than that, 280 traces are enough to recover `a[0]`.



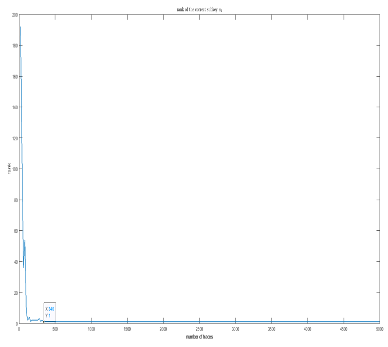
5.1: scores sub-key guesses for `a[0]`



5.2: scores sub-key guesses for `a[1]`



6.1: correct sub-key rank w.r.t number of attack traces guesses for `a[0]`



6.2: correct sub-key rank w.r.t number of attack traces for `a[1]`

3 Unprofiled Correlation Power Attack

We have seen in sec. 2.3 that our considered intermediate values are very sensitive to correlation analysis. We therefore decided to also perform a CPA attack on the pair-pointwise multiplication in ML-KEM. The CPA attack investigates the correlation between the targeted intermediate values and the physical measurements (in this case power consumption). We recall (as mentioned in sec. 2.2) that our targeted intermediate values are **`fqmul(a[0], b[1])`** when attacking **`a[0]`** and **`fqmul(a[1], b[1])`** when attacking **`a[1]`**. Let us describe the procedure for an attack targeting the secret sub-key **`a[0]`**. The procedure also applies mutatis mutandis to **`a[1]`**.

Procedure (CPA attack):

1. Run through the sub-key space (as mentioned before, **`a[0]`** $\in \mathbb{Z}_q$. i.e the sub-key space is $[0,\ q-1]$)

2. For each possible guess $keyGuess$ of the (ML-KEM decryption) sub-key **`a[0]`**, compute the hypothetical intermediate value $P(keyGuess)$, which is the Hamming weight of the most significant byte **MSByte(`fqmul(keyGuess,b[1])`)**.

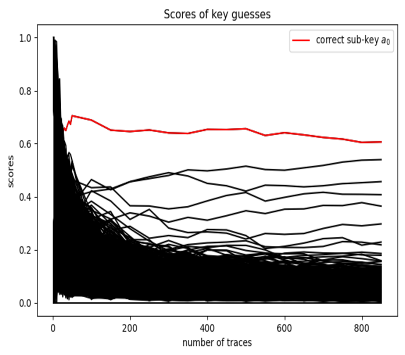
3. Compute subsequently the score of that guess as the maximum absolute value of the Pearson correlation ρ between the traces and $P(keyGuess)$

$$score(keyGuess) = \max_j \left(Abs\left(\rho\left(T_j, P(keyGuess)\right)\right)\right)$$

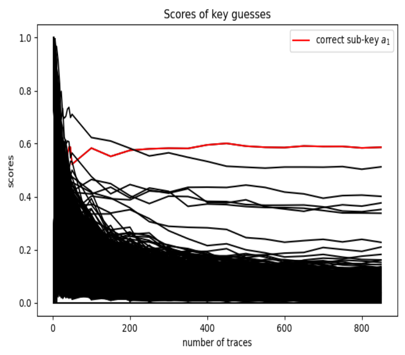
T_j denoting the trace vector at sample time j

4. pick up the guess with the highest score.

The figures below show the result of our CPA attack on the SCA dataset published in [3].

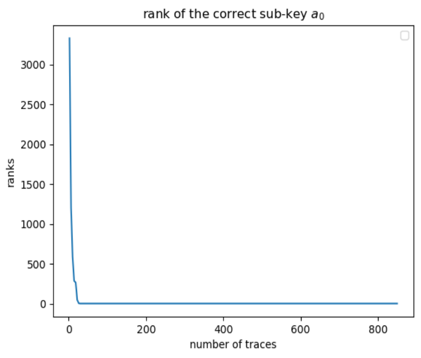


7.1: scores sub-key guesses for `a[0]`

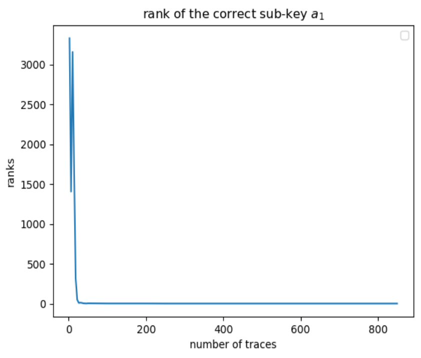


7.2: scores sub-key guesses for `a[1]`

Remark: One observes in figures 8.1 and 8.2 that only 30 traces are enough to recover the sub-key `a[0]`, whereas about 250 traces are required to recover `a[1]`.



8.1: correct sub-key rank w.r.t number of attack traces guesses for `a[0]`



8.2: correct sub-key rank w.r.t number of attack traces for `a[1]`

4 Conclusion

We recall that each pair-pointwise multiplication involves two coefficients of the secret key (cf. FIPS-203, sec. 4.3.1, algorithm 11). We have recovered two coefficients of the NTT-representation of the whole secret key, namely a_0 and a_1, out of the 256 coefficients. This is due to the fact that the available dataset only captures a single pair-pointwise coefficients multiplication. The same attack can be replicated in parallel over the 127 remaining pair-pointwise multiplications in case the corresponding traces are available. Even though this investigation was not a comparative study between template attack and CPA attack, one notices for this dataset that the CPA attack performs better, in the sense that, it succeeds with a much lower amount of traces compared to template attack.

References

[1] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[2] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Federal Information Processing Standards Publication FIPS 203. U.S. Department of Commerce, 2024. URL: https://doi.org/10.6028/NIST.FIPS.203.

[3] Azade Rezaeezade et al. “Side-Channel Power Trace Dataset for Kyber Pair-Pointwise Multiplication on Cortex-M4”. In: *Cryptology ePrint Archive* (2025).