Cybersecurity

RESEARCH

Open Access

In-depth Correlation Power Analysis Attacks on a Hardware Implementation of CRYSTALS-Dilithium

Huaxin Wang[1], Yiwen Gao[1*], Yuejun Liu[1], Qian Zhang[2,3] and Yongbin Zhou[1,2,3]

Abstract

During the standardisation process of post-quantum cryptography, NIST encourages research on side-channel analysis for candidate schemes. As the recommended lattice signature scheme, CRYSTALS-Dilithium, when implemented on hardware, has seen limited research on side-channel analysis, and current attacks are incomplete or requires a substantial quantity of traces. Therefore, we conducted a more complete analysis to investigate the leakage of an FPGA implementation of CRYSTALS-Dilithium using the Correlation Power Analysis (CPA) method, where with a minimum of 70,000 traces partial private key coefficients can be recovered. Furthermore, we optimise the attack by extracting Point-of-Interests using known information due to parallelism (named CPA-PoI) and by iteratively utilising parallel leakages (named CPA-ITR). Our experimental results show that CPA-PoI reduces the number of traces by up to 16.67%, CPA-ITR by up to 25%, and both increase the number of recovered key coefficients by up to 55.17% and 93.10% using the same number of traces. They outperfom the CPA method. As a result, it suggests that the FPGA implementation of CRYSTALS-Dilithium is more vulnerable than thought before to side-channel analysis.

Keywords CRYSTALS-Dilithium, Post-Quantum Cryptography, Correlation Power Analysis, FPGA, Side-Channel Attack

Introduction

With quantum computers, the conventional public-key cryptosystems, such as RSA, DSA, etc., can be broken by Shor's algorithms (Shor 1994) without much effort. In response to the threats, National Institute of Standards and Technology (NIST) initiated a post-quantum cryptography (PQC) standardisation process in December 2016 (Moody 2016), and finally announced four algorithms for standardisation in July 2022, with CRYSTALS-Kyber (Avanzi et al. 2019) being selected as a post-quantum Key Encapsulation Mechanism (KEM), and CRYSTALS-Dilithium (Dilithium for short) (Ducas et al. 2018), Falcon (Fouque et al. 2018), and SPHINCS+ (Bernstein et al. 2015) being selected as post-quantum signature schemes. Amongst the signature schemes, NIST primarily recommends Dilithium, as it believes this is the primary algorithm for digital signatures.

Dilithium ensures Strong existential Unforgeability under Chosen Message Attacks (SUF-CMA), and its security is guaranteed by lattice-based hard problems. The security of cryptographic algorithms theoretically lies on their mathematical structures, but in practice, they are also under threats of side-channel attacks (SCAs). The importance of side-channel security for PQC is also emphasised in the NIST PQC standardisation process. Existing side-channel attacks for implementations of Dilithium typically target the random number generation, Number-Theoretic Transform (NTT), or polynomial

*Correspondence:
Yiwen Gao
gaoywin@gmail.com
[1] School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing, China
[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

multiplication operations. Side-channel leakage from these operations has been investigated using various methods, including Simple Power Analysis (SPA), Correlation Power Analysis (CPA) (Brier et al. 2004), and profiled attacks (Chari et al. 2002). However, those research mostly focuses on software implementations, having little concern on SCA to hardware implementations of Dilithium due to the hardness (Steffen et al. 2022). In the light of this, we study the security of hardware implementations of Dilithium under side channel attacks.

In this paper, we investigate the vulnerabilites of Dilithium and propose practical side-channel attacks to recover the private key of Dilithium by analysing a typical Field-Programmable Gate Array (FPGA) implementation of Dilithium.

Related work

Typical implementations of Dilithium is based on ARMs or FPGAs. For ARM-based implementations. Ravi et al. (2018) proposed a non-profiled side-channel attack targeting polynomial multiplication, by which partial private key was extracted and utilized to forge signatures. However, they believe that the attack goes beyond polynomial time. Subsequently, Chen et al. (2021) proposed a conservative CPA method to reduce the key guessing space and a fast two-stage method to further reduce the guessing space for attacking polynomial multiplication operations. As a result, they were able to fully recover the private key using the conservative CPA method with only 157 power traces. The hybrid method, combining the fast two-stage and conservative CPA, saved the attack's execution time by 87%. Qiao et al. (2023) proposed a new non-profiled attack method, called Public Template Attack (PTA), on both unprotected and protected implementations of Dilithium (Migliore et al. 2019), targeting the random polynomial $\mathbf{y}$, successfully recovering the private key. Non-profiled attacks focus primarily on polynomial multiplication, while profiled attacks focus primarily on NTT operations. Primas et al. (2017) proposed a generic method targeting NTT operations that requires establishing templates for all possible multiplications in butterfly operations, which is costly. Han et al. (2021) employed a machine learning-based method to attack NTT operations, recovering keys using 60,000 power traces. Berzati et al. (2023) reconstructed a given coefficient in a predicted vector to determine if it is zero, thus recovering the private key using linear algebra methods, with 700,000 power traces.

For FPGA-based implementations, Steffen et al. (2022) proposed elector-magnetic analysis targeting polynomial multiplication in Dilithium. They conducted two main attacks. Firstly, the profiled SPA method was used to attack the Decode and the first-stage NTT operations. Due to the parallelism of multiple key coefficients in the implementation, the authors assume that the adversary can fix for all key coefficients but one. The adversary capability is too strong. Secondly, the CPA method of least significant bit (LSB) is used to attack the polynomial multiplication implementation, and theoretically for the hardware implementation, multi-bits model can depict the information leakage more accurately, and the Hamming distance model is more suitable for the attack of the hardware implementation. Although it attacks both $c\mathbf{t}_0$ and $c\mathbf{s}_1$, $c\mathbf{t}_0$ does not affect the security of the Dilithium scheme. It only recovered one key coefficient for $c\mathbf{s}_1$ using 1,000,000 electromagnetic traces, and the Pearson correlation coefficient for the correct key guess was not significant.

At present, of the attacks targeting polynomial multiplication of Dilithium, Chen et al.'s attack (Chen et al. 2021) performs best. However, due to the pre-charging mechanism of CMOS circuits in ARM platform, the Hamming weight model is selected as the power consumption model. While the attack on the FPGA implementation needs to be selected according to the specific implementation, which is generally the Hamming distance. In addition, due to the parallelization of the hardware implementation, the signal-to-noise ratio is lower than that of the ARM implementation. Ma et al. (2022) attacks the two different Kyber implementations: one with three multiplications in parallel and the other with a single multiplication. The results show that attacking the three multiplications parallel implementation uses four times as many power traces as the other one. From Stefffen et al.'s work (Steffen et al. 2022), it can be seen that FPGA implementation of Dilithium has more parallel operations than the ARM implementation, such as keccak operation and parallel computation of multiple key coefficients, and it takes more time to attack it, so it is difficult to perform a complete analysis. The fast two-stage scheme of Chen et al.'s work (Chen et al. 2021) reduces the attack time, but the best key recovery can be achieved when the power trace used is 63 times that of the conservative CPA scheme, while the recovery of one key coefficient of the $c\mathbf{s}_1$ in the attack of Steffen et al. (2022) has used 1,000,000 electromagnetic traces, and the increase in the dataset of the fast two-stage scheme will be very significant, which is relatively high for the memory requirements of the computer. Hence, it is not practical to trade more traces for less time. Among the mentioned works, only Steffen et al. (2022) conducted attacks on an FPGA-based implementation of Dilithium, but they did not analyse more key coefficients.

In terms of attack methods, profiled attacks require more from the adversary's abilities, as it necessitates the attacker to access the same device during both the modeling phase and the attack phase. The attacker must have complete control over the device during the modeling phase. Futhermore, in the Dilithium implementation, NTT operations can be pre-calculated and cannot be subjected to profiled attacks. For profiled attacks on polynomial multiplication, the large key guessing space requires a substantial amount of data for modeling, and the lower signal-to-noise ratio in the FPGA implementation makes profiled attacks challenging. Therefore, this paper only focuses on non-profiled attacks.

Contributions

The contributions of this paper are summerised as follows:

- We provide a more comprehensive and feasible analysis using power leakages from a hardware implementation of Dilithium. We analyse its characteristics and then use the CPA method to attack. In this analysis, it has been demonstrated that partial key coefficients can be recovered with a minimum of 70,000 power traces.
- We precisely extract Point-of-Interests (PoIs) from power traces using parallel execution operations independent of the key coefficient. This method is referred to as CPA-PoI. Compared to the CPA attack, it reduces the scale of the Pearson calculation and also decreases the number of power traces required for the attack, with a reduction of up to 16.67%. Its average Guessing Entropy is lower than that of the CPA method. When attacking with the same number of power traces, the number of recovered key coefficients is increased by up to 55.17%.
- We propose a better method to reduce noise, called CPA-ITR, by using the leakage operations from other key coefficients in parallel. Compared to the CPA method, this method reduces the number of power traces used by up to 25%. At the same time, when attacking with the same number of power traces, the number of recovered key coefficients can be improved by up to 93.10%.

Organisations

The structure of this paper is as follows: Sect. 2 provides notations involved in this paper, and gives a brief introduction to Dilithium v3.1 signature scheme. It also introduces the target FPGA implementation and Correlation Power Analysis. Section 3 provides a detailed description of target operation and methods proposed in this paper. Section 4 introduces the experimental setups and analyses the experimental results of our attacks. Finally, conclusions are given in Section 5.

Preliminaries

Notations

Let n and q be two integers, where $n = 256$ and $q = 8380417 = 2^{23} - 2^{13} + 1$. We use R_q to denote the polynomial ring $\mathbb{Z}[x]/(x^n + 1)$, the infinity norm $||x||_\infty$ denotes the maximum absolute value among all coefficients of a polynomial x. For a polynomial vector, this norm is defined as the maximum infinity norm of all polynomials in the vector. Therefore, S_b denotes the set of polynomials in R_q with infinity norm equal to b, while $\tilde{S}_b$ denotes the same set but excluding coefficients with value $-b$. Additionally, the set of polynomials in R_q with exactly τ nonzero coefficients and infinity norm equal to 1 is denoted as B_τ. We use bold lowercase letters to denote vectors (e.g. $\mathbf{v}$), and bold uppercase letters to denote matrices (e.g. $\mathbf{A}$). Polynomials in the NTT domain are denoted with a hat (e.g. $\hat{c}$, $\hat{c} = \mathrm{NTT}(c)$). This notation is transitive, so $\hat{\mathbf{s}}$ denotes each polynomial in $\mathbf{s}$ being individually transformed into the NTT domain. Finally, we use $\circ$ to denote pointwise multiplication. $\mathrm{HD}(a \circ b)[i]$ denotes the value of the Hamming distance after calculating the i-th coefficient of the register storing the computation result of $a \circ b$.

Algorithm 1 Key Generation

```
1: $\zeta \leftarrow \{0,1\}^{256}$
2: $(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := \mathrm{H}(\zeta)$
3: $\mathbf{A} \in R_q^{k\times\ell} := \mathrm{ExpandA}(\rho)$
4: $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^\ell \times S_\eta^k := \mathrm{ExpandS}(\rho')$
5: $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
6: $(\mathbf{t}_1, \mathbf{t}_0) := \mathrm{Power2Round}_q(\mathbf{t}, d)$
7: $tr \in \{0,1\}^{256} := \mathrm{H}(\rho||\mathbf{t}_1)$
8: **return** $(\mathrm{pk} = (\rho, \mathbf{t}_1), \mathrm{sk} = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0))$
```

CRYSTALS-Dilithium

Dilithium consists of three algorithms: key generation, signature generation, and signature verification. Because the signature verification is unrelated to the attacks mentioned in this paper, it will not be further introduced here.

Key Generation The key generation process generates a private key for signature generation and a public key for verification, as shown in Algo. 1. From this, it can be seen that finding the private key from the public key is essentially equivalent to solving the M-LWE problem. Additionally, once an attacker obtains either value $\mathbf{s}_1$ or $\mathbf{s}_2$, they can derive the other value directly since $\mathbf{A}$ and $\mathbf{t}$ are public values. The Power2Round function is used to split the M-LWE problem instance $\mathbf{t}$ into high and low bits in order to compress the size of the public key.

*Signature Generation*The signature generation process is shown in Algo. 2. It begins by recomputing $\mathbf{A}$ and hashing the message M along with the hash value of the public key tr. The loop is terminated by generating noise $\mathbf{y} \in S^{\ell}_{(\gamma_1-1)}$ using the ExpandMask function. Compress the $\mathbf{w} = \mathbf{A}\mathbf{y}$ to $\mathbf{w}_1$ using the HighBits function. The hint $\mathbf{h}$ allows the verifier to recompute $\mathbf{w}_1$. The hash function H instantiates the random oracle required in the proof. It returns a sparse ternary polynomial $c \in B_{60}$, which has a Hamming weight of 60 and all non-zero coefficients equal to $+1$ or -1. The Decompose function returns both HighBits and LowBits of its input. Finally, the check is performed to determine if the current signature is rejected, and if so, it is recomputed. Otherwise, the signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ is generated.

Algorithm 2 Signature Generation

```
1: $\mathbf{A} \in R_q^{k\times\ell} := \mathrm{ExpandA}(\rho)$
2: $\mu \in \{0,1\}^{512} := \mathrm{H}(tr||M)$
3: $\kappa := 0,\ (\mathbf{z},\mathbf{h}) := \perp$
4: $\rho' \in \{0,1\}^{512} := \mathrm{H}(K||\mu)$
5: while $(\mathbf{z},\mathbf{h}) = \perp$ do
6:     $\mathbf{y} \in \tilde{S}^{\ell}_{\gamma_1} := \mathrm{ExpandMask}(\rho',\kappa)$
7:     $\mathbf{w} := \mathbf{A}\mathbf{y}$
8:     $\mathbf{w}_1 := \mathrm{HighBits}_q(\mathbf{w}, 2\gamma_2)$
9:     $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu||\mathbf{w}_1)$
10:    $c \in B_\tau := \mathrm{SampleInBall}(\tilde{c})$
11:    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
12:    $\mathbf{r}_0 := \mathrm{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
13:    if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ then
14:        $(\mathbf{z},\mathbf{h}) := \perp$
15:    else
16:        $\mathbf{h} := \mathrm{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
17:        if $\|c\mathbf{t}_0\|_\infty \geq \gamma_2$ or $\sum h_i > \omega$ then
18:            $(\mathbf{z},\mathbf{h}) := \perp$
19:        end if
20:    end if
21:    $\kappa := \kappa + \ell$
22: end while
23: return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$
```

*Dilithium Parameters*The first submission (Ducas et al. 2017) to the PQC competition underwent several parameter modifications during the NIST PQC standardisation process. The current version can be found in Bai et al. (2021), and compared to the previous submission (Ducas et al. 2019), the main adjustments were made to the k and ℓ dimensional parameters in order to better comply with NIST's security levels. The parameters of the current version 3.1 can be seen in the Table 1.

Table 1 Dilithium Parameters for version 3.1 (Dilithium v3.1)

Parameter set	Value		
Security level	2	3	5
τ [# of ±1's in c]	39	49	60
ω [max # of 1's in hint $\mathbf{h}$]	80	55	74
(k,ℓ) [Dimensions of $\mathbf{A}$]	(4,4)	(6,5)	(8,7)
η [secret key range]	2	4	2
q [Modulus]	8380417		
d [dropped bits from $\mathbf{t}$]	13		

Target FPGA implementation of Dilithium

Our attacks are conducted on the hardware implementation Dilithium by Beckwith et al. (2021). It is known for its good performance as an FPGA implementation with high speed. The algorithm of the signature generation is shown in Fig. 1, which is divided into *pre-computation phase* and *rejection loop phase*. During the *pre-computation phase*, the key is decoded and transformed into the NTT domain. At the same time, the calculations for $\mathbf{w}$ and $\mathbf{y}$, which are required in the *rejection loop* (Line 5 to 22 in Algo. 2), are computed in advance, and the calculation results are directly provided to generate the signature in the *rejection loop*. The calculation in the *rejection loop* is divided into two-stage pipelines, *Stage*-0 and *Stage*-1. Stage-0 prepares for the next Stage-1 calculation. If the generated signature σ passes in Stage-1, it is output as the signature without performing the calculations within the red dashed line. If it fails, the loop continues until a valid signature is generated.

In the implementation, polynomial multiplication (including NTT), addition, and multiplication are implemented through polynomial arithmetic units. It utilizes four butterfly units to process four coefficients in parallel for all operations. For the multiplication, reduction of the computed result is required. The hardware implementation uses Barrett reduction, which can be implemented using only shifting and addition operations.

Therefore, when analysing the implementation of polynomial multiplication, the *rejection loop* and the polynomial arithmetic units bear a high degree of parallelism, resulting in a large amount of algorithmic noise during side-channel analysis, which affects the attacking results.

Correlation Power Analysis

Side-channel analysis utilizes the power consumption or electromagnetic information generated by the execution of a cryptographic algorithm on a device in order to extract sensitive information. Correlation power analysis (CPA) is one of the methods used in side-channel analysis. It is essentially an improvement of DPA. In practical attacks, the classical CPA typically involves five steps:

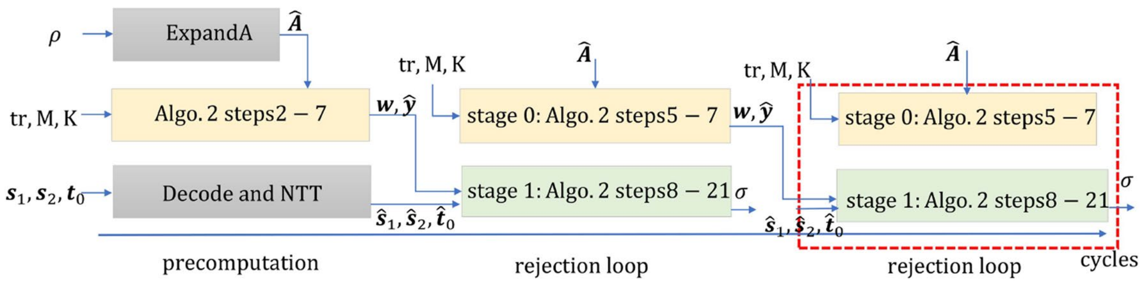


Fig. 1 The architecture of the target FPGA implementation of Dilithium

- Select an appropriate intermediate value as the attack position. The calculation function of this intermediate value takes the key (or a fixed value from which the key can be derived) and known variables as inputs.
- Collect the power traces of the targeted operation. Execute the signing process n times and store the power traces of each collection, with each trace consisting of m data points, in the matrix $T_{n\times m}$.
- Compute the intermediate value matrix $V_{n\times k}$ for the guessed key. Based on the range of key coefficient values, the size of key guessing space k can be determined. Using the assumed key and known variables, the intermediate values of the targeted operation can be calculated.
- Using an appropriate power consumption model, map the values of the intermediate value matrix $V_{n\times k}$ to the assumed power consumption matrix $H_{n\times k}$ one-to-one.
- Compute the correlation coefficients between the assumed power consumption matrix $H_{n\times k}$ and the actual power consumption matrix $T_{n\times m}$ for each column, and record them in the correlation matrix $R_{k\times m}$. The calculation of correlation coefficients can be done using Pearson's correlation coefficient formula, as shown in Eq. 1.

$$R_{i,j} = \frac{\sum_{x=1}^{n}(H_{x,i} - \overline{H}_i)\cdot(T_{x,j} - \overline{T}_j)}{\sqrt{\sum_{x=1}^{n}(H_{x,i} - \overline{H}_i)^2 \cdot \sum_{x=1}^{n}(T_{x,j} - \overline{T}_j)^2}} \tag{1}$$

The index of the maximum value in matrix R corresponds to the leakage point and the key used in the targeted operation.

The proposed side channel attacks

Vulnerable region

In the process of Dilithium signature generation, polynomial multiplications are calculated many times. The calculations of $c\mathbf{s}_1$ and $c\mathbf{s}_2$ involve the public variable c, while $\mathbf{s}_1$ and $\mathbf{s}_2$ are both parts of the private key, making them suitable for CPA. This paper focuses on attacking the hardware implementation of $c\mathbf{s}_1$. In the context of Dilithium, c is a polynomial with $n = 256$ coefficients, each ranging from -1 to 1. $\mathbf{s}_1$ is a polynomial vector consisting of ℓ polynomials, with a total of $\ell \times n$ coefficients, each ranging from $-\eta$ to η. To speed up the computation of polynomial multiplication, Dilithium algorithm introduces NTT. Therefore, before performing polynomial multiplication, the polynomials are mapped to the NTT domain and then multiplied using point-wise multiplication, as shown in Eq. 2.

$$c\mathbf{s}_1 = \mathrm{INTT}(\mathrm{NTT}(c) \circ \mathrm{NTT}(\mathbf{s}_1)) \tag{2}$$

In the FPGA implementation (Beckwith et al. 2021), the polynomial arithmetic unit is responsible for polynomial multiplication (including NTT), addition, and subtraction. The design utilises a 2×2 butterfly structure, which is applied to all polynomial arithmetic operations, enabling parallel processing of four coefficients. Barrett reduction is used for multiplication, which involves fewer multiplications than that of Montgomery reduction.

To avoid leaking the private key, Dilithium employs rejection sampling (Step 5 to 22 in Algo. 2), with the loop repetitions of 4.25. The parallelised pipeline design accelerates signature generation. However, it also introduces additional noise during side-channel analysis due to the simultaneously executed operations such as $\mathbf{y} = \mathrm{ExpandMask}(\rho', \kappa)$, which involves the `keccak` function, in the calculation of $(\hat{c} \circ \hat{\mathbf{s}}_1)$. As shown in **Fig.** 2, different colors are used in the figure to indicate the different states during the execution of `keccak`. This noise can affect the effectiveness and efficiency of the attack.

CPA on polynomial multiplication

In the C reference implementation of Dilithium, the NTT operation omits the reduction operation to reduce the computational load. After 8 levels of recursive butterfly calculations, a 256-dimensional polynomial yields coefficients in the NTT domain ranging from $[-\eta - 8(q-1), \eta + 8(q-1)]$. At security level 2, where

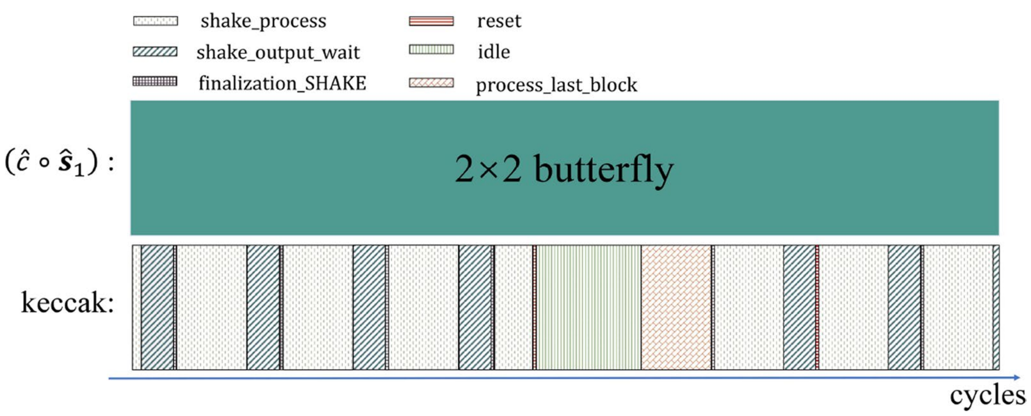


Fig. 2 The overlapping of `keccak` and $(\hat{c} \circ \hat{\mathbf{s}}_1)$ in the time domain

$\eta = 2$ and $q = 2^{23} - 2^{13} + 1$, the size of key guessing space in a CPA is close to 2^{27}. This leads to a relatively large computational complexity when performing CPA on the Dilithium implementation.

However, in the target implementation, Barrett reduction is applied after the butterfly multiplication, which limits the range of polynomial coefficients in the NTT domain to $[0, q)$. Therefore, in the attack on Dilithium, the key guessing space used by the attacker is $[0, q)$.

For software implementations, it is difficult to analyse the continuous numerical processing on the bus, so the Hamming weight of the targeted variable is often used as a leakage model (Chen et al. 2021; Qiao et al. 2023). For hardware implementations, it is possible to analyse the previous reference value of the operation by examining registers, hence the Hamming distance model is commonly used to model the power consumption generated by the targeted operation (Steffen et al. 2022). The actual power consumption L can be expressed as the Hamming distance between the values before and after (Note: the coefficient index is i) the register is assigned a value, as shown in Eq. 3.

$$L = \alpha \times \mathrm{HD}\big(\hat{c} \circ \hat{\mathbf{s}}_1\big)[i] + N \tag{3}$$

where α denotes the scaling factor, and N refers to the noise.

From Sect. 2.4, we learn that the number of points in a power trace directly affects the scale of calculating the correlation coefficient matrix. In hardware implementation, the value of a register can only change once per clock cycle. Therefore, assuming no clock delay, the variations in register values are closely related to the intervals between points on the power trace in consecutive clock cycles. Specifically, the number of sample points within one clock cycle can be calculated by dividing the sampling rate by the clock frequency. During an attack, the power trace can be sliced to obtain leakage points more accurately. This can be done by adding the current sample point index to the product of the number of sample points and the clock interval for register value changes, in order to determine the location of the leakage point. These leakage points can then be used to replace the entire segment of the power trace, to construct the correlation coefficient matrix.

It should be noted that the above description is based on ideal conditions. However, in practical applications, there is often some delay in the data. Therefore, the selected leakage points during the attack process will be adjusted by plus or minus 15 from their original positions to ensure that the leakage points fall within the selected sample points for the attack.

CPA-PoI on polynomial multiplication

In the attack on $c\mathbf{s}_1$ by Steffen et al. (2022), a million electromagnetic traces were utilised. Even when using a million power traces for the attack, the computation scale of the Pearson correlation is still enormous, even with the aforementioned CPA method. In CPA, if PoIs can be accurately selected, it not only reduces the computational scale of the correlation coefficient matrix but also enables a more direct determination of the location of information leakage. Therefore, we first select points by locating the leakage point, and then proceed with the CPA method for the attack, we refer to this attack method as CPA-with-PoI (abbr. **CPA-PoI**).

In side-channel analysis, any points in the power trace that exhibit significant differences are considered as PoIs. In profiled attacks, the selection of the number of PoIs directly affects the size of the Pearson correlation. Therefore, it is necessary to identify the PoIs in order to reduce the size of the templates. Similarly, in CPA, if the PoIs can be accurately selected, it not only reduces the computational scale of the correlation coefficient matrix but also allows for more direct localization of the leakage position, minimizing the impact of noise.

In the same clock cycle, the timing of value changes for different registers may overlap. This overlap can result in partial repetition or overlap of PoIs in the power traces. Therefore, PoIs of the target operation may coincide with PoIs of other operations. By observing the implemented architecture, it is found that during the computation of $(\hat{c} \circ \hat{\mathbf{s}}_1)[i]$, the $\hat{c}$ needed for subsequent calculations is fetched in advance, thereby generating power consumption. Additionally, due to the 2×2 butterfly structure, there are four consecutive indices of $\hat{c}$ coefficients that can be processed in parallel. The figures obtained by calculating the Pearson correlation using $\hat{c}$, $(\hat{c} \circ \hat{\mathbf{s}}_1)[i]$, and the power traces are shown in Fig. 3. It can be seen that the challenge $\hat{c}$ coincides with the PoIs of the key in the power trace, and the correlations with $\hat{c}$ are high. Since $\hat{c}$ is known, it can be used to extract PoIs.

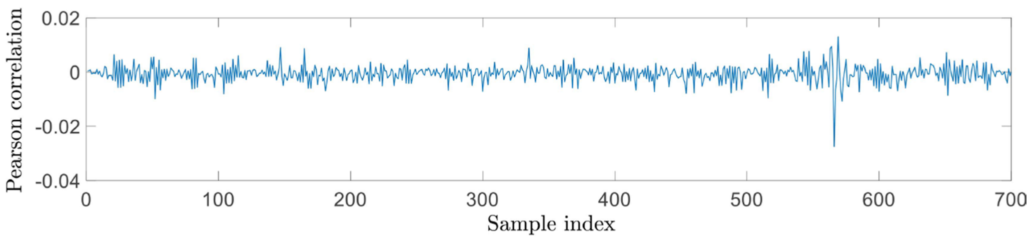
Based on this idea, after using the power trace segmentation method in the aforementioned CPA, we calculate the Pearson correlation coefficient between the Hamming distance of the $\hat{c}$ coefficients and power traces, then select several values with the absolute value of the relevant coefficient at the forefront, and use the points in the corresponding power trace to perform CPA. By using this method, we can extract PoIs more accurately.

CPA-ITR on polynomial multiplication

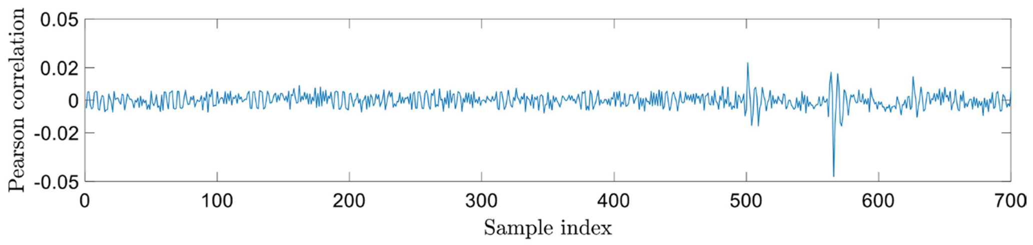
In CPA, the Hamming distance of a coefficient's target operation is used as an assumed power consumption. In the FPGA implementation of Dilithium, polynomial multiplication involves the simultaneous use of 2×2 butterfly structures, as shown in Fig. 4. The values within the square brackets in the figure denote the coefficient indices of $\hat{c}$ and $\hat{\mathbf{s}}_1$. A good power consumption model can help an attacker accurately identify, extract, or infer sensitive information from the target device. According to Eq. 3, for the actual power consumption L generated by the same power trace, the smaller the noise N, the closer the assumed power consumption $\left(\hat{c} \circ \hat{\mathbf{s}}_1\right)[i]$ is to H. In this case, the attack achieves the best results. A single coefficient's assumed power consumption may not be sufficiently close to the actual power consumption.

At the same moment, the coefficients being processed have indices i, $i+1$, $i+2$ and $i+3$, if only the operation with the coefficient index i is attacked, the other coefficients are considered as part of the noise N, resulting in higher noise levels. Meanwhile, in the scenario of parallel attack on all four coefficients, the size of key guessing space has changed from q to q^4, making the attack more difficult.

In practical attacks, for parallel computations occurring at the same moment, they are usually attacked in the order of their indices. This means that when attacking the operation with index $i+1$, the related value for the operation with index i is already known. Therefore, after recovering the key coefficient with index i using the CPA method (using Eq. 4 as the assumed power consumption), Eq. 5 can be used as the assumed power consumption to recover the key coefficient with index $i+1$, considering that Eq. 4 is known. This process can be repeated sequentially, modifying the assumed power consumption model (Eq. 6, Eq. 7), to better model the leakage and reduce the noise levels.



(a) The Pearson correlation between the HDs of $\hat{c}$ and the traces.



(b) The Pearson correlation between the HDs of $(\hat{c} \circ \hat{\mathbf{s}}_1)[i]$ and the traces.

Fig. 3 The Pearson correlation between the hypothetical leakages related to $c\mathbf{s}_1$ and the traces

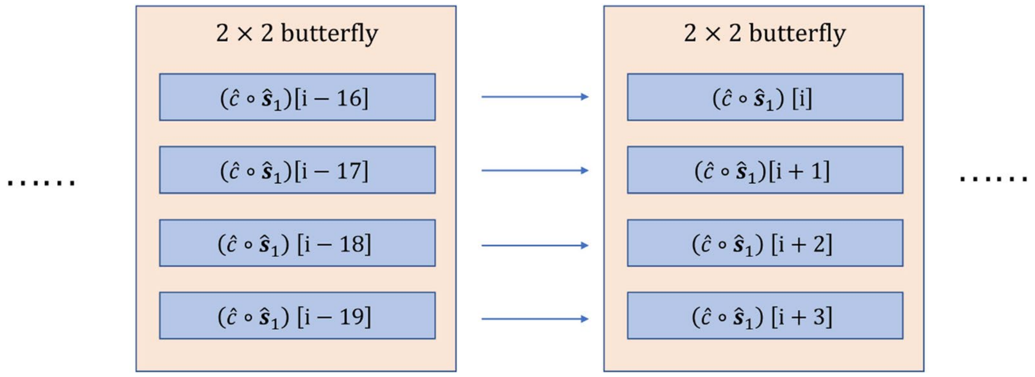

2 × 2 butterfly
$(\hat{c} \circ \hat{s}_1)[i-16]$
$(\hat{c} \circ \hat{s}_1)[i-17]$
$(\hat{c} \circ \hat{s}_1)[i-18]$
$(\hat{c} \circ \hat{s}_1)[i-19]$
2 × 2 butterfly
$(\hat{c} \circ \hat{s}_1)[i]$
$(\hat{c} \circ \hat{s}_1)[i+1]$
$(\hat{c} \circ \hat{s}_1)[i+2]$
$(\hat{c} \circ \hat{s}_1)[i+3]$
……
……

Fig. 4 The value of the 2×2 butterfly operation's registers at adjacent cycles

$$w_1 = \alpha \times \mathrm{HD}\big(\hat{c} \circ \hat{\mathbf{s}}_1\big)[i] \tag{4}$$

$$w_2 = w_1 + \alpha \times \mathrm{HD}\big(\hat{c} \circ \hat{\mathbf{s}}_1\big)[i+1] \tag{5}$$

$$w_3 = w_2 + \alpha \times \mathrm{HD}\big(\hat{c} \circ \hat{\mathbf{s}}_1\big)[i+2] \tag{6}$$

$$w_4 = w_3 + \alpha \times \mathrm{HD}\big(\hat{c} \circ \hat{\mathbf{s}}_1\big)[i+3] \tag{7}$$

where α denotes scaling factors.

Here, due to the parallel nature of the target operation, we store the information obtained from previous attacks and combine it with the assumed power consumption model for subsequent attacks, in order to reduce noise. We refer to the method of dynamically overlaying the assumed power consumption model in CPA attacks as CPA-with-Iteration (abbr. **CPA-ITR**).

Experimental results

Setup The power traces of polynomial multiplication execution in Dilithium are collected on the SAKURA-X development board for side-channel evaluation. The setup is shown in Fig. 5, which consists of a *ROHDE & SCHWARZ* PA303 30dB pre-amplifier, *PicoScope* 3206D oscilloscope, and SAKURA-X FPGA development board with *Xilinx* Kintex7 XC7K160T chip. The chip runs at 4MHz. The oscilloscope can simultaneously use two channels to sample with a 4*ns* interval (i.e. sampling rate 250MS/s). One channel is used for collecting traces of instant power consumption, and the other is used to trigger for sampling.

Target FPGA Implementation Our target implementation is the FPGA implementation of Dilithium v3.1 in Beckwith et al. (2021). For the purpose of analysis, the security level is set to 2 (increasing the security level will increase the number of key coefficients but will not make the attack more difficult). It is implemented using both

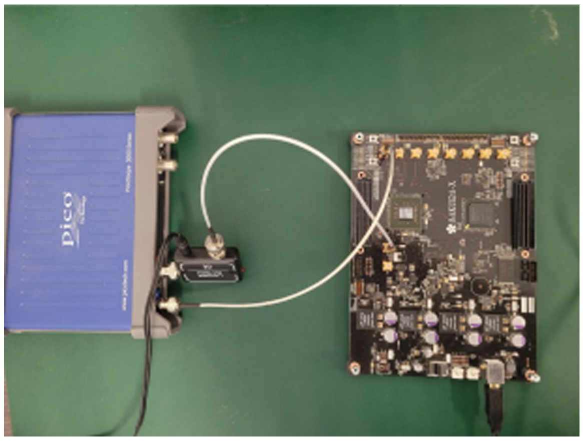


Fig. 5 The setup of our experiments

Verilog and VHDL languages. This introduces an delay-optimised FPGA design and covers the three security levels of Dilithium. We programme the signature generation of Dilithium into the SAKURA-X target board and send the message and its length from the PC to the target board for signing. We use the ISE−14.7 Design Suite to programme the Kintex7 XC7K160T chip.

Some Settings The power traces collected for the polynomial multiplication on $c\mathbf{s}_1$ are shown in Fig. 6, where a clear periodicity can be observed. Using these traces, an attacker can analyse the key coefficients sequentially. Due to the parallel execution of the `keccak` function during the polynomial multiplication process, as shown in Fig. 2, each state generates different algorithmic noise, which affects the attacks differently. In order to evaluate and compare the results of the attack experiments, we obtained the state of the `keccak` function execution for each clock cycle through timing simulation and conducted separate attack experiments accordingly. Fig. 7 shows the correlation trend of the most likely key guesses for 16,761 (i.e. $\lceil \frac{8380417}{500} \rceil$) under the six states of the

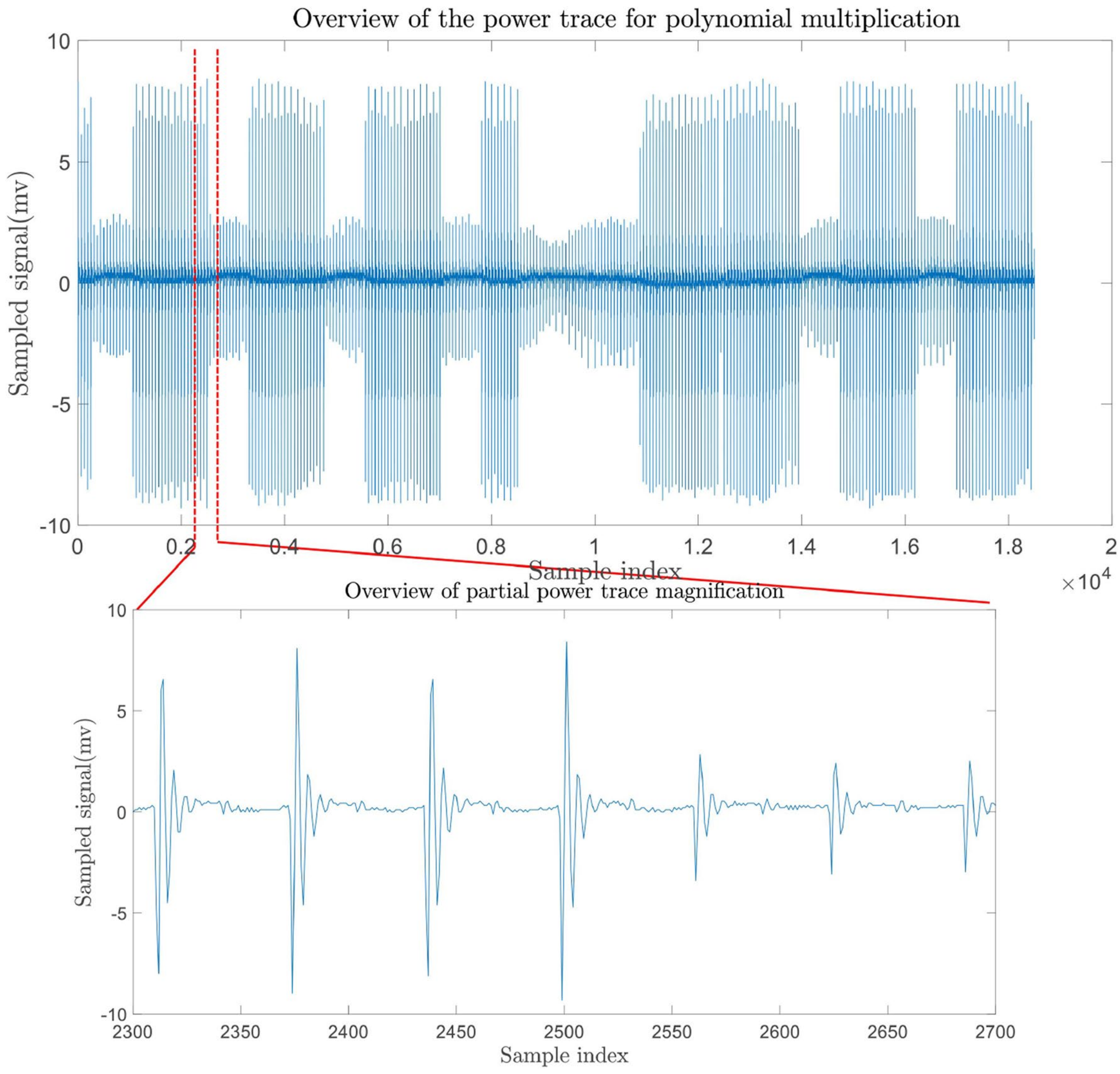


Fig. 6 Power traces of FPGA implementation of Dilithium in our setting

`keccak` function, based on different numbers of traces. The correct key guess is indicated in red, where the correct key guesses are marked in red and standing out with more traces. From the graph, it can be observed that a significant correlation is only observed after 700,000 power traces for the `shake_process` and `process_last_block` states, which leads to a longer attack time. Therefore, in the experimental process of this paper, not all key coefficients were targeted for attack, but only a selected few were chosen based on the target operations present in each state of `keccak`. For `reset` and `finalization_SHAKE` states, there are a total of 8 and 24 parallel target operations respectively, so all coefficients were chosen for attack. For `shake_output_wait` and `idle` states, which have a larger number of parallel target operations, 64 were selected as reference. Due to the excessively long attack time, only 32 were chosen as reference for the `shake_process` and `process_last_block` states.

Results of the CPA method

First, the CPA method is executed, using the power trace segmentation method for the attack. The calculation scale at this stage is $N \times P \times q$, where N denotes the number of power traces used and P corresponds to the number of sampling points selected per power trace, which is set to 31.

The experimental results of using different power traces for the attack are shown in Table 2. The critical number of power traces required to fully recover the key

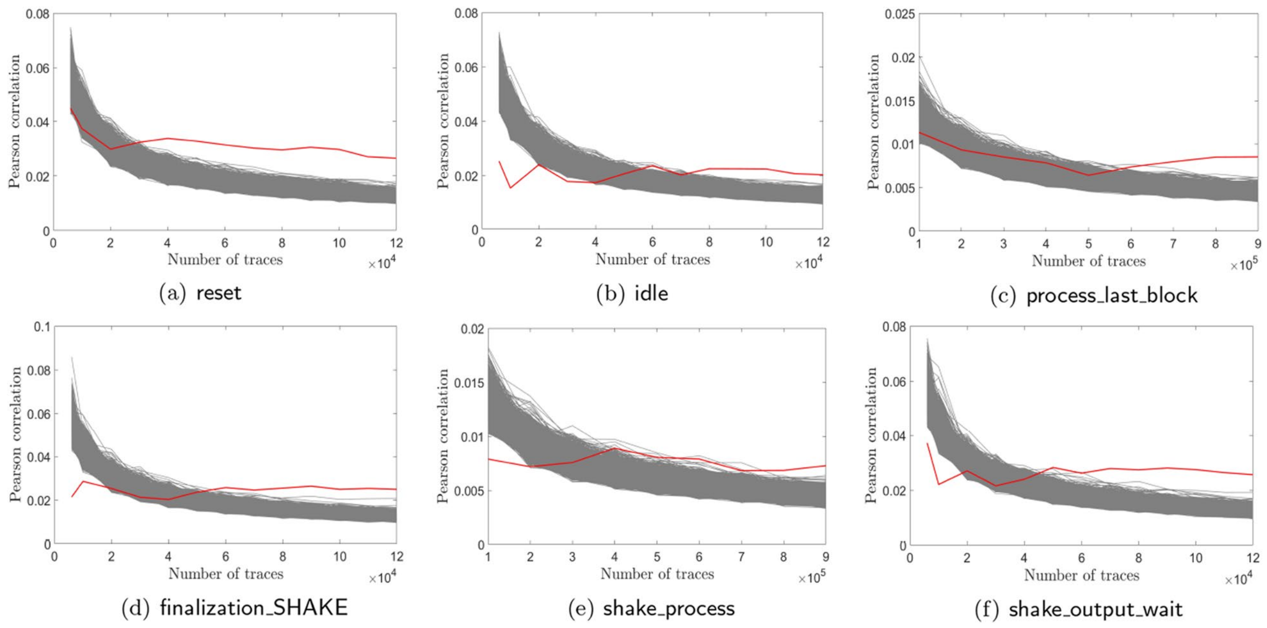


Fig. 7 Absolute correlation values for 16,761 most likely key guesses with different numbers of traces

Table 2 The results of key recovery using different methods in different keccak states

keccak state	Metric	CPA	CPA-PoI	CPA-ITR	CPA	CPA-PoI	CPA-ITR	CPA
reset	#Traces[1] $(\times 10^4)$	6	6	6	7	7	7	/[4]
	Recovery ratio[2]	5/8	6/8	8/8	8/8	8/8	8/8	/
	Average GE[3]	23/3	4/2	–[5]	–	–	–	/
idle	#Traces[1] $(\times 10^4)$	9	9	9	11	11	11	/
	Recovery ratio[2]	29/64	45/64	56/64	64/64	64/64	64/64	/
	Average GE[3]	226/3	27/2	7	–	–	–	/
finalization_SHAKE	#Traces[1] $(\times 10^4)$	7	7	7	8	8	8	/
	Recovery ratio[2]	20/24	22/24	23/24	24/24	24/24	24/24	/
	Average GE[3]	87/4	13/2	4	–	–	–	/
shake_output_wait	#Traces[1] $(\times 10^4)$	9	9	9	10	10	10	12
	Recovery ratio[2]	49/64	63/64	64/64	61/64	64/64	64/64	64/64
	Average GE[3]	7/3	4	–	4/2	–	–	–
process_last_block	#Traces[1] $(\times 10^4)$	80	80	80	83	83	83	88
	Recovery ratio[2]	22/32	26/32	26/32	22/32	32/32	32/32	32/32
	Average GE[3]	10/2	2	2	6/2	–	–	–
shake_process	#Traces[1] $(\times 10^4)$	80	80	80	87	87	87	98
	Recovery ratio[2]	21/32	29/32	32/32	28/32	32/32	32/32	32/32
	Average GE[3]	17/3	2	–	11/2	–	–	–

[1] #Traces: the number of traces used.

[2] Recovery ratio: e.g., $\frac{a}{b}$ denotes attacking key coefficients of b and successfully recovering a.

[3] Average GE: Average Guessing Entropy, in the context of $\frac{a}{b}$, where b denotes the number of sequence, containing consecutive key coefficients, in which some key coefficient recovery fail, and a denotes the sum of the GEs for the failed key coefficient recovery.

[4] /: indicates that the target key coefficients has been fully recovered.

[5] –: indicates that the guessed entropy for all key coefficients is 1.

coefficients is the threshold, and the average Guessing Entropy (GE) is the mean of the initial Guessing Entropy for incorrectly recovered key coefficient guesses. For the `shake_output_wait` state, recovering 64 key coefficients requires 120,000 power traces. For the `finalization_SHAKE` state, recovering 24 key coefficients requires 80,000 power traces. For the `reset` state, recovering the group key coefficient requires 70,000 power traces. For the `idle` state, recovering 64 key coefficients requires 110,000 power traces. Lastly, for the `shake_process` and `process_last_block` states, recovering 32 key coefficients requires 980,000 and 880,000 power traces, respectively. The number of power traces required to recover the key coefficients is consistent with the trend shown in Fig. 7, with a drastic increase in the number of power traces during the `shake_process` and `process_last_block` states, indicating a higher level of algorithmic noise during these states. However, Chen et al. (2021) used the CPA method on the ARM implementation of Dilithium to recover the private key with only 157 power traces. This clearly demonstrates a significant difference in the difficulty between attacking software and hardware implementations.

Results of the CPA-PoI method

Based on the CPA method, the CPA-PoI method was adopted to more accurately locate the PoIs. When selecting the PoIs, we used the Hamming distance parameter $\hat{c}$ and selected 3 points. The calculation scale in this case is $N \times P \times q$, where P is 3. Compared to the CPA method, the CPA-PoI method has a reduced calculation scale.

According to the results in Table 3, compared to the CPA method, the CPA-PoI method reduces the number of power traces required to fully recover the target key coefficients by [0, 16.67%]. Even when the number of power traces is not reduced, the average GE of the CPA-PoI method is generally lower than that of the CPA method, which indicates that the effect of the CPA-PoI method can be observed by further refining the number of power traces used in the attack.

In cases where the CPA method has not fully recovered the target key coefficients coefficients, when attacking with the same number of power traces, the CPA-PoI method significantly increases the number of recovered key coefficients, with an improvement range between [10%, 55.17%]. This indicates that, with the same number of power traces, the CPA-PoI method can recover more key coefficients and achieves better results compared to the CPA method.

Results of the CPA-ITR method

Based on the CPA method, we employed the CPA-ITR method to effectively leverage parallelization for information leakage. By comparing the experimental results with the CPA and CPA-PoI methods, we derived Table 3.

From Table 3, it can be seen that compared to the CPA method, the CPA-ITR method reduces the number of power traces required to fully recover the target key coefficients by [0, 25%], and compared to the CPA-PoI method, it reduces the number by [0, 14.28%]. Additionally, the CPA-ITR method significantly decreases the average GE, which is close to that of the CPA-PoI method. When attacking with the same number of power traces, the CPA-ITR method improves the number of recovered key coefficients compared to the CPA method by [15%, 93.10%]. In comparison to the CPA-PoI method, the improvement range is [0, 33.33%]. Overall, the CPA-ITR method demonstrates significant improvement

Table 3 Comparison of attacks using different methods in different keccak states

keccak state	#traces[1]		#coefficients[2]		#traces[1]	#coefficients[2]
	CPA		CPA		CPA-PoI	CPA-PoI
	CPA-PoI[3]	CPA-ITR	CPA-PoI[4]	CPA-ITR	CPA-ITR	CPA-ITR
reset	0	14.28%	20%	60%	14.28%	33.33%
idle	0	0	55.17%	93.10%	0	24.44%
finalization_SHAKE	0	0	10%	15%	0	4.54%
shake_output_wait	16.67%	25%	28.57%	30.61%	10%	1.58%
process_last_block	5.68%	5.68%	45.45%	45.45%	0	0
shake_process	11.22%	18.36%	38.09%	52.38%	8.05%	10.34%

[1] #traces: the number of traces used.

[2] #coefficients: the number of key coefficients recovered.

[3] The data in the column (Col. 2) means the percentage reduction of #traces with CPA-PoI compared to CPA. The data in Col. 3 & 6 are of the similar explanation.

[4] The data in the column (Col. 4) means the percentage reduction of #coefficients with CPA-PoI compared to CPA. The data in Col. 5 & 7 are of the similar explanation.

compared to the CPA method and offers certain advancements over the CPA-PoI method as well.

It is important to note that in the CPA-ITR attack, if the first attacked key coefficient is incorrectly recovered, it may cause interference in the attacks on the other three key coefficients in parallel. However, during our experiments, we observed that the recovery of the first coefficient was generally easier to achieve in our implementation.

Discussion

The proposed methods proposed are not only targeted for the implementations of Dilithium but also can be applied to other algorithms. For the CPA method, no parallel information is used, except for the difference in the power consumption model, which is generally applicable to the implementation of microprocessor. CPA-PoI utilizes parallel leakage of known information operations that are not related to the key coefficient, and is not applicable if there are no similar leakage for implementation on microprocessor. CPA-ITR uses a leakage where multiple key coefficients are executed in parallel, and this method can be applied to implementations that use similar parallel structures for computation. In summary, our methods can be applied to attack implementations of other algorithms.

Conclusion

The paper presents a practical attack on the FPGA implementation of Dilithium, This is a more comprehensive work to attack the FPGA implementation of Dilithium using power leakages. By fully utilizing the characteristics of FPGA implementation, we have improved CPA with two methods, namely CPA-PoI and CPA-ITR, both of which demonstrate better performance compared to CPA in our experiments. Our work demonstrates the feasibility of side-channel attack to polynomial multiplication operations on highly parallelised hardware. It suggests that the FPGA implementation of CRYSTALS-Dilithium is more vulnerable than thought before to side-channel analysis. In future work, we plan to explore more in-depth analysis of masked implementation using high-order CPA-PoI and CPA-ITR. However, due to the large key guessing space and the impact of algorithm noise, attacks on both unmasked and masked implementations take a long time. Therefore, we also aim to find better attack strategies to efficiently recover the private key with less time.

Abbreviations

CPA	Correlation Power Analysis
SCA	Side-Channel Attack
NTT	Number-Theoretic Transform
PoIs	Point-of-Interests
CPA-PoI	CPA-with-PoI
CPA-ITR	CPA-with-Iteration
FPGA	Field-Programmable Gate Array
HD	Hamming distance
GE	Guessing Entropy

Acknowledgements
We would like to thank the anonymous reviewers and editors for detailed comments and useful feedback.

Author contributions
HW and YG proposed the methods for attacking. HW conducted the experiments and wrote the manuscript. YG, YL, QZ, YZ and HW participated in discussions and paper reviews. All authors read and approved the final manuscript.

Funding
This work is supported in part by National Key R & D Program of China (No.2022YFB3103800), National Natural Science Foundation of China (No. U1936209, No.62202231 and No.62202230), the Defense Industrial Technology Development Program (No. JCKY2021606B013), China Postdoctoral Science Foundation (No.2021M701726), Jiangsu Funding Program for Excellent Postdoctoral Talent (No.2022ZB270), Yunnan Provincial Major Science and Technology Special Plan Projects (No.202103AA080015) and CCF-Tencent Rhino-Bird Open Research Fund (No.CCF-Tencent RAGR20230114).

Availibility of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 16 October 2023 Accepted: 15 January 2024
Published online: 03 June 2024

References

Fouque PA, Hoffstein J, Kirchner P, Lyubashevsky V, Pornin T, Prest T, Ricosset T, Seiler G, Whyte W, Zhang Z et al (2018) Falcon: Fast-Fourier lattice-based compact signatures over NTRU. Submiss NIST's Post-quantum Cryptogr Stand Process 36(5):1–75

Han J, Lee T, Kwon J, Lee J, Kim IJ, Cho J, Han DG, Sim BY (2021) Single-trace attack on NIST round 3 candidate Dilithium using machine learning-based profiling. IEEE Access 9:166283–166292. https://doi.org/10.1109/ACCESS.2021.3135600

Qiao Z, Liu Y, Zhou Y, Ming J, Jin C, Li H (2023) Practical public template attack attacks on CRYSTALS-Dilithium with randomness leakages. IEEE Trans Inf Forensics Secur 18:1–14. https://doi.org/10.1109/TIFS.2022.3215913

Avanzi R, Bos J, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schanck J, Schwabe P, Seiler G, Stehlé D (2019) CRYSTALS-Kyber (version 2.0)-algorithm specifications and supporting documentation (April 1, 2019). Submission to the NIST post-quantum project

Bai S, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D (2021) CRYSTALS-Dilithium: algorithm specifications and supporting documentation (version 3.1). NIST Post-Quantum Cryptography Standardization Round 3

Beckwith L, Nguyen DT, Gaj K (2021) High-performance hardware implementation of CRYSTALS-Dilithium. In: International conference on field-programmable technology (ICFPT) 2021, Auckland. IEEE, pp 1–10. https://doi.org/10.1109/ICFPT52863.2021.9609917

Bernstein DJ, Hopwood D, Hülsing A, Lange T, Niederhagen R, Papachristodoulou L, Schneider M, Schwabe P, Wilcox-O'Hearn Z (2015) SPHINCS: practical stateless hash-based signatures. In: Advances in cryptology–EUROCRYPT 2015–34th annual international conference on the theory and applications of cryptographic techniques, Sofia, Proceedings, Part

I. Springer, vol 9056, pp 368–397. https://doi.org/10.1007/978-3-662-46800-5_15
Berzati A, Viera AC, Chartouni M, Madec S, Vergnaud D, Vigilant D (2023) A practical template attack on CRYSTALS-Dilithium. IACR Cryptology ePrint Archive 50
Brier E, Clavier C, Olivier F (2004) Correlation power analysis with a leakage model. In: Cryptographic hardware and embedded systems—CHES 2004: 6th international workshop Cambridge, Proceedings, 3156. Springer, pp 16–29. https://doi.org/10.1007/978-3-540-28632-5_2
Chen Z, Karabulut E, Aysu A, Ma Y, Jing J (2021) An efficient non-profiled side-channel attack on the CRYSTALS-Dilithium post-quantum signature. In: 39th IEEE international conference on computer design (ICCD) 2021, Storrs. IEEE, pp 583–590. https://doi.org/10.1109/ICCD53106.2021.00094
Chari S, Rao JR, Rohatgi P (2002) Template Attacks. In: Cryptographic hardware and embedded systems–CHES 2002, 4th international workshop, Redwood Shores, Revised Papers, 2523. Springer, pp 13–28. https://doi.org/10.1007/3-540-36400-5_3
Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D (2019) CRYSTALS-Dilithium: algorithm specifications and supporting documentation. Round-2 submission to the NIST PQC project 35
Ducas L, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D (2017) CRYSTALS-Dilithium: digital signatures from module lattices. IACR Cryptology ePrint Archive 633
Ducas L, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D (2018) Crystals–Dilithium: digital Signatures from module lattices. Submission to NIST's post-quantum cryptography standardization process
Ma H, Pan S, Gao Y, He J, Zhao Y, Jin Y (2022) Vulnerable PQC against side channel analysis—a Case Study on Kyber. In: Asian hardware oriented security and trust symposium (AsianHOST) 2022, Singapore. IEEE, pp 1–6. https://doi.org/10.1109/AsianHOST56390.2022.10022165
Migliore V, Gérard B, Tibouchi M, Fouque PA (2019) Masking Dilithium—efficient implementation and side-channel evaluation. In: Applied cryptography and network security—17th international conference, ACNS 2019, Bogota, Proceedings. Springer, vol 11464, pp 344–362. https://doi.org/10.1007/978-3-030-21568-2_17
Moody D (2016) Post-quantum cryptography standardization: announcement and outline of NIST's Call for submissions. In: International conference on post-quantum cryptography (PQCrypto) 2016
Primas R, Pessl P, Mangard S (2017) Single-trace side-channel attacks on masked lattice-based encryption. In: Cryptographic hardware and embedded systems–CHES 2017–19th international conference, Taipei, Proceedings, 10529. Springer, pp 513–533. https://doi.org/10.1007/978-3-319-66787-4_25
Ravi P, Jhanwar MP, Howe J, Chattopadhyay A, Bhasin S (2018) Side-channel assisted existential forgery attack on Dilithium–A NIST PQC candidate. IACR Cryptology ePrint Archive 821
Ravi P, Jhanwar MP, Howe J, Chattopadhyay A, Bhasin S (2019) Exploiting determinism in lattice-based signatures–practical fault attacks on pqm4 implementations of NIST candidates. IACR Cryptol. ePrint Arch. 2019(769)
Shor PW (1994) Algorithms for quantum computation discrete logarithms and factoring. In: 35th annual symposium on foundations of computer science, Santa Fe, New Mexico. IEEE Computer Society, pp 124–134. https://doi.org/10.1109/SFCS.1994.365700
Steffen HM, Land G, Kogelheide LJ, Günaysu T (2022) Breaking and protecting the crystal: side-channel analysis of Dilithium in hardware. IACR Cryptol. ePrint Arch. 2022(1410)

Publisher's Note