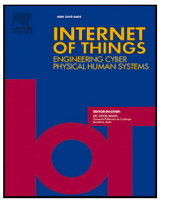
Internet of Things 27 (2024) 101331

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot



A comprehensive side-channel leakage assessment of CRYSTALS-Kyber in IIoT

Zitian Huang [a], Huanyu Wang [b,*], Bijia Cao [a], Dalin He [a], Junnian Wang [a]

[a] School of Physics and Electronic Science, Hunan University of Science and Technology, Xiangtan 411201, China
[b] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China

ARTICLE INFO

Keywords:
Post-Quantum Cryptography
CRYSTALS-Kyber
Hardware security
Deep learning
Side-Channel Attacks
Countermeasures

ABSTRACT

Following the establishment of the draft standardization for Post-Quantum Cryptography (PQC), cryptographic systems across various sectors have undergone a paradigm shift. Although the theoretical strength of PQC has provided a robust foundation for securing communications against quantum threats, physical implementations of PQC algorithms remain vulnerable to Side-Channel Attacks (SCAs). Existing SCA studies predominantly focus on the attack process, lacking thorough side-channel leakage assessments and comparisons of inherent vulnerabilities at different attack points and with different countermeasures. In this paper, we first present a comprehensive assessment of side-channel leakage and resistance of four attack points within an ARM Cortex-M4 implementation of Kyber, including its masked version. This assessment employs a range of countermeasures such as noise addition, random delays, clock jitter, and their combinations. Besides, we also build deep-learning models for attacking, thereby verifying the results of the leakage assessments. By collaboratively utilizing three distinct leakage assessment approaches and deep learning-based attack results, we experimentally demonstrate that different algorithmic intermediate values of Kyber are suited to different countermeasures, which advances our understanding of the capacity and vulnerability of PQC implementations.

1. Introduction

Cryptography is the cornerstone of information security, and traditional standard encryption algorithms, like RSA, are widely used in various fields, including network communications, the Internet of Things (IoT), and digital currencies, etc. However, the security of existing standard cryptographic algorithms is threatened due to the rise of quantum algorithms and quantum computing such as Shor. Due to the high standards of security and reliability required of passwords in the application field of the Industrial Internet of Things (IIoT), it has a more urgent need for advancements in cryptography. Recognizing the urgency of this issue, research on cryptographic algorithms in the post-quantum era has come into focus. The National Institute of Standards and Technology (NIST) published an announcement of the first standardization call for PQC in 2016 and promulgated the first batch of standardized algorithms in August 2023. The CRYSTALS-Kyber, the basis for the module-lattice-based Key Encapsulation Mechanism (KEM) algorithm in Federal Information Processing Standards (FIPS) draft 203, has received significant attention and is slated for implementation in the coming years. It will gradually replace traditional encryption methods and be deeply integrated with the IIoT to further enhance data security in the post-quantum era.

CRYSTALS-Kyber is a module-lattice-based post-quantum KEM algorithm that achieves Chosen-Ciphertext Attacks (CCA) secure through a Fujisaki-Okamoto (FO) transform [1]. FO transform is widely used in post-quantum cryptography design. This transform

* Corresponding author.
E-mail address: huanyu@hnust.edu.cn (H. Wang).

https://doi.org/10.1016/j.iot.2024.101331
Received 14 June 2024; Received in revised form 3 August 2024; Accepted 9 August 2024
Available online 14 August 2024

performs security enhancement by re-encrypting the decrypted message and comparing the re-generated ciphertext with the received ciphertext. As a more powerful KEM algorithm, it can be integrated with various fields of security, such as key management [2] and communication security [3]. Kyber is bound to be widely implemented in the field of IIoT to ensure security. However, in a practical scenario, a high correlation exists between the side-channel information leaked by the re-encryption step and the message allows attackers to bypass the theoretical strength security provided by the FO transform, making the re-encryption step a critical vulnerability for the side-channel attack on Kyber. To mitigate the effect caused by side-channel attacks, cryptographic implementations typically incorporate countermeasures such as masking [4], shuffling [5], random delays [6], and clock jitter [7] to secure the encryption process on the physical plane.

SCAs were first proposed in [8,9]. The attack methods mainly include power analysis [9], electromagnetic analysis [10], timing attacks [8], and various combined attack approaches. Among them, the power analysis also includes a variety of methods such as Simple Power Analysis (SPA), Differential Power Analysis (DPA), Correlation Power Analysis (CPA), and template attack, which speculate the key by analyzing the correlation between the power consumption of cryptographic devices and the key-dependent intermediate value. Compared to other IoT devices, IIoT devices are more susceptible to side-channel attacks as the scale and complexity of IIoT systems increase. Therefore, when deploying them, it is crucial to consider the threat of side-channel attacks and take corresponding countermeasures. In the implementation of Kyber, a commonly adopted countermeasure is masking [11]. But with the advances of deep learning technology and its application in the field of side-channel attacks, many side-channel countermeasures have been compromised as deep-learning models are good at extracting features from raw data [12,13]. Currently, there have been many studies on side-channel attacks on implementations of Kyber. Some of the research on the weak points in the Kyber, such as [14,15], in turn, promoted improvements and refinements in subsequent implementations of Kyber, as seen in [16]. However, these studies are focused on key recovery and lack a comprehensive leakage assessment of the Kyber implementations. A systematic leakage assessment study could help us understand the algorithm properties more deeply and contribute to further improvement of countermeasures.

1.1. Contributions

In this paper, we primarily focus on two types of attack points: the encoding and decoding operations common in both unmasked and masked Kyber implementations. We analyze these functions in both implementations, concentrating on the leakage assessment of their power traces and the accuracy of deep learning-based side-channel attacks. The main contributions of this paper are as follows:

1. We conduct the first horizontal assessment of internal vulnerabilities in Kyber implementations. In the experiment, we assess the side-channel leakage of power traces captured from an ARM Cortex-M4 at four attack points using three different techniques under various countermeasures and their combinations. We compared these results and thereafter analyzed their characteristics and the difficulty level of the attacks.
2. We build a deep-learning model suitable for the four attack points for verifying the leakage assessment results. This provides a more accurate reflection of the variations in traces of attack points when applying different countermeasures, as well as the varying difficulty of attacks under different conditions. We compare these results with leakage assessment and provide additional explanations.
3. We adapt and explain the countermeasures applicable to different attack points in the implementation of Kyber.

2. Background

In this section, we provide a concise overview of the CRYSTALS-Kyber, masked Kyber, SCAs on implementations of Kyber, leakage assessment techniques, and side-channel countermeasures.

2.1. Notations

Let $q \in N$ be a prime and $f(x) = x^n + 1$ where $n = 2^{n'-1}$, $n' \in Z^+$. This ensures that $f(x)$ is irreducible over rational numbers. $\mathbb{Z}_q$ is the ring of integers modulo q. Let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ be the ring of integer polynomials modulo both $f(x)$ and q. $\mathcal{R}_q^{k\times k}$ represents a matrix of dimension $k \times k$ over $\mathcal{R}_q$. We use $\mathcal{U}$ to denote the uniform distribution over $\mathcal{R}_q$, χ_η to represent the centered binomial distribution, and symbol $\circ$ to denote point-wise multiplication. Typically, we fix $n = 256$ and $q = 3329$, and we control the security level of the algorithm by changing the dimension k of the matrix. In this paper, we set $k = 3$, which is for Kyber768.

2.2. CRYSTALS-Kyber

Kyber [17] is the only KEM standardized algorithm published by NIST in the PQC domain. Its theoretical security is based on the Module Learning With Errors (M-LWE) problem [18] and achieves CCA-secure. By combining with the implementation of Kyber, IIoT systems can ensure long-term security and resist classical and quantum computing threats. Specifically, Kyber contains an Indistinguishability under Chosen-Plaintext Attack (IND-CPA) secure Public Key Encryption (PKE) and uses an FO transformation to construct an IND-CCA secure KEM. The critical step in this transformation is the re-encryption in KEM decapsulation. The encapsulation and decapsulation operational processes, Kyber = (KeyGen, Encaps, Decaps), are outlined in Algorithms 1 and 2.

Let k, d_u, d_v be positive integer parameters and set $n = 256$ and $q = 3329$ [17]. The encapsulation process operates on the public key pk, message m, and a random coin r. Through operations such as NTT, NTT^{-1}, and compress, the ciphertext ct is generated at the end. This process is called again in Kyber.KEM.Dec() to form re-encryption, and it is this process that we are mainly concerned with.

In the decapsulation process, the inputs are the private key sk and the ciphertext ct. The decapsulation process calculates the decrypted message m' using Kyber.PKE.Dec(), then re-encrypts it to obtain the ciphertext c'. By comparing c' with the received ciphertext ct, the algorithm maintains IND-CCA. If they are equal, the function returns the session key K. Otherwise, it returns a pseudorandom string.

In particular, Decode is the function that deserializes an array into a polynomial as described in [19], while Encode is its inverse function. In these algorithms, $pk = (\text{seed}_A, \hat{b})$, $sk = (sk' \| pk \| \mathcal{H}(pk) \| z)$, where seed_A is a random seed, $\hat{b}$ is a polynomial vector and z is pseudo-random value. Both $\mathcal{G}$ and $\mathcal{H}$ are hash functions, and KDF represents the Key Derivation Function.

In the submitted scheme, Kyber contains three different versions (Table 1), Kyber512, Kyber768, and Kyber1024, which correspond to three different security levels, AES-128, AES-192, and AES-256, respectively. This difference is mainly achieved by adjusting the matrix dimension k. In addition, Kyber also provides a "90s" version of each algorithm, in which AES and SHA2 are used instead of SHAKE and SHA3. Nonetheless, this difference does not affect our following leakage assessment and attack experiments.

Algorithm 1 IND-CPA secure PKE in Kyber

Procedure Kyber.PKE.Enc(pk, m, r):
  $\hat{\mathbf{A}}^T \leftarrow \mathcal{U}(\mathcal{R}_q^{k\times k}, \text{seed}_A)$
  $r \leftarrow \chi_{\eta_1}(\mathcal{R}_q^{k\times 1}; r)$
  $e_1 \leftarrow \chi_{\eta_2}(\mathcal{R}_q^{k\times 1}; r)$
  $e_2 \leftarrow \chi_{\eta_2}(\mathcal{R}_q; r)$
  $\hat{r} \leftarrow \text{NTT}(r)$
  $u \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{r}) + e_1$
  $v \leftarrow \text{NTT}^{-1}(\hat{\mathbf{b}}^T \circ \hat{r}) + \text{Decompress}_q(\textbf{Decode}(\text{m})) + e_2$
  $c_1 \leftarrow \textbf{Encode}(\text{Compress}(u, d_u))$
  $c_2 \leftarrow \textbf{Encode}(\text{Compress}(v, d_v))$
  return $ct = (c_1 || c_2)$
End Procedure
Procedure Kyber.PKE.Dec(sk, ct):
  $u \leftarrow \text{Decompress}_q(u, d_\text{u})$
  $v \leftarrow \text{Decompress}_q(v, d_\text{v})$
  $\hat{s} = \textbf{Decode}(sk)$
  $m = \textbf{Encode}(\text{Compress}_q(v - \text{NTT}^{-1}(\hat{s} \circ \text{NTT}(u)), 1))$
  return m
End Procedure

Algorithm 2 FO transformation to a IND-CCA secure KEM[11]

Procedure Kyber.KEM.Enc(pk):
  $m \leftarrow \mathcal{U}(\{0,1\}^{256})$
  $(\bar{K}, r) = \mathcal{G}(m || \mathcal{H}(pk))$
  $ct = \text{Kyber.PKE.Enc}(pk, m, r)$
  $K = \text{KDF}(\bar{K} \| \mathcal{H}(ct))$
  return (ct, K)
End Procedure
Procedure Kyber.KEM.Dec(sk, ct):
  $m' = \text{Kyber.PKE.Dec}(sk, ct)$
  $(\bar{K}', r') = \mathcal{G}(m' || \mathcal{H}(pk))$
  $c' = \text{Kyber.PKE.Enc}(pk, m', r')$
if $ct = c'$ **then**
    $K = \text{KDF}(\bar{K}' \| \mathcal{H}(ct))$
else
    $K = \text{KDF}(z \| \mathcal{H}(ct))$
end if
  return K
End Procedure

Table 1
Parameters differences among various versions of Kyber.

Algorithm	k	n	q	pk bytes	sk bytes	ct bytes
Kyber-512	2	256	3329	800	1632	768
Kyber-768	3	256	3329	1184	2400	1088
Kyber-1024	4	256	3329	1568	1568	3168

2.3. Masked Kyber

To mitigate the effects caused by side-channel attacks, Chari et al. proposed masking [4], a countermeasure that categorizes s into $s_1, s_2, \ldots, s_\omega$, totaling ω shares, and perform all operations on each share to resist ω-1 order attacks. Encryption algorithms used in IIoT systems usually need to have the characteristics of high efficiency and low overhead and lightweight encryption algorithms are mainly used. In the existing mask implementation of Kyber, we choose to test the first-order masking countermeasures, which result in two repeated computations and more complex operations in the module. Due to the complexity of Kyber, the implementation of higher-order masks would incur significant computational costs, which is why we currently focus on first-order masking. We chose the first-order masked Kyber768 [11] implemented by Daniel Heinz et al. for our study. They apply the masking to four modules: decryption, re-encryption, SHA3, and ciphertext comparison. This represents a widely adopted masking scheme for Kyber at present. In this experiment, when we refer to masked Kyber, we are usually referring to this first-order masked Kyber algorithm.

2.4. Leakage assessment technology

In our leakage evaluation, we usually use three methods to evaluate the leakage level of the attack point, which are: T-test, Signal-to-Noise Ratio (SNR), and correlation power analysis.

T-test T-test is frequently employed within the context of Test Vector Leakage Assessment (TVLA) for evaluating the susceptibility of cryptographic algorithms to side-channel leakage [20]. It is often used to examine whether there is a significant difference in the mean values of side-channel power consumption or other leakage at different inputs. The version we use is proposed by Welch [21], which is widely used in the field of leakage detection. In designing the masked algorithm of Kyber, as in [11], the T-test is also used as a leakage evaluation method.

For two captured traces sets D_1, D_2, the T-test value (v) is calculated as follows:

$$v = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}},$$

where μ_i and σ_i are the mean and variance of D_i, and n_i is the number of traces in D_i, for $i \in \{0, 1\}$. When the T-test value is larger than 4.5, it is considered that there is a significant difference between the two sets, indicating a potential leakage of information.

Signal-to-noise ratio SNR is the ratio between the strength of a signal and the intensity of noise. In the field of SCA, SNR can be used to measure the extent of information leakage during the execution of a cryptographic algorithm. The level of SNR reflects the ability to clearly detect side-channel information related to the execution of cryptographic algorithms, and a high SNR usually indicates easier access to cryptographic information as well as higher potential security risks. SNR provides an important tool for assessing the risk of information exposure.

For SNR, we define Q as the power consumed by the function and N as the additive noise [22]. Then, the total power consumption is written as $D_{ij} = Q_{ij} + N_{ij}$. SNR (s) calculation formula is as follows:

$$s = \frac{\hat{\mathrm{var}}_i(\hat{\mathbb{E}}_i(D_i))}{\hat{\mathbb{E}}_i(\hat{\mathrm{var}}_i(D_i))},$$

where $\hat{\mathbb{E}}$ represents the sample mean and $\hat{\mathrm{var}}$ is the sample variance of dataset [23].

Correlation power analysis CPA is a common means of analyzing the correlation between power consumption and key operation, proposed in [24] for application in power analysis of cryptographic devices. In [25], they successfully extract the key from lattice-based post-quantum KEM based on correlation analysis. In this method, by monitoring the power consumption changes of the cryptographic device under different key values, the attacker obtains information about the internal operation of the encryption algorithm through statistical analysis, and then gradually guesses the key. Correlation is typically calculated using the Pearson correlation coefficient (c), and its formula is as follows:

$$c = \frac{\sum (D_{ij} - \mu_1)(H_{ij} - \mu_2)}{\sqrt{\sum (D_{ij} - \mu_1)^2 \cdot \sum (H_{ij} - \mu_2)^2}},$$

where D_{ij} denote the jth observations of the ith datasets, H is the corresponding Hamming Weight (HW) label, and μ_i is the mean of D_i. The advantages of CPA in terms of efficiency, robustness, and precision make it more suitable for complex cryptographic devices and algorithms.

2.5. SCAs on implementations of Kyber

SCAs is an attack method that relies on analyzing the leaked information from physical devices rather than depending on computational strength to deduce cryptographic keys. This characteristic makes it particularly relevant in the context of post-quantum cryptographic analysis. When conducting side-channel attacks, a common approach is to employ deep learning techniques.

In recent years, due to the advantages of deep learning, it has been more and more widely used in all fields, including side-channel attacks. This makes side-channel attacks against IoT also gradually increase [26]. Commonly used neural networks [27] for such attacks include Multilayer Perceptron (MLP) [28], and Convolutional Neural Networks (CNNs) [29], among others. In this paper, we use MLP as the tool, which typically consists of one input layer, one output layer, and several hidden layers. MLP is like a mathematical function that maps a set of input values to output values, composed of much simpler functions [30]. During the training stage, we input side-channel information and use the corresponding key-related information as the target output. By adjusting weights and biases, the MLP learns the profile between side-channel leakage and the key, which can lead to the loss of information security.

A common strategy for SCAs on Kyber implementations is to recover the message for extracting the key. For instance, [31] presents a single-trace attack on the message decoding operation in lattice-based KEMs. Similarly, [32] introduces a general-purpose side-channel attack based on FO transform that can be used for Kyber. Furthermore, in [33], they leverage message recovery and four carefully chosen ciphertexts to recover the entire secret key through SPA. Expanding on this, [34] conducts the attack on the bit-wise incremental storage in memory during the message decryption process. They define the concept of ciphertext malleability for learning with errors/rounding based PKE/KEMs, exploiting targeted flips of message bits for more effective message recovery.

After the publication of masked Kyber, corresponding second-order attacks were also proposed, combining traditional side-channel attack methods with deep learning. [35] implements the first practical attack on message decoding in masked Kyber. The closest related work is presented in [36], a new deep learning-based message recovery attack is proposed. Through recursive learning of neural networks, it can recover message bits from Kyber with α-order masking ($\alpha \leq 5$).

2.6. Side-channel countermeasures

With the deepening of the research on side-channel attacks, countermeasures against SCAs also came into being. The efficiency of side-channel attacks is affected by many different practical factors. In the following experiments, we focus on factors such as noise, random delays, and clock jitter.

Noise The types of noise include impulse noise, Gaussian noise, white noise, and colored noise, among others, with Gaussian noise being the most common type of noise in side-channel information gathering. Transistors, data buses, transmission lines to oscilloscopes, and even the working environment can be sources of Gaussian noise [37]. In our experiments, we generate Gaussian noise of varying degrees by adjusting the SNR (N_{SNR}) and overlying the collected power traces to simulate different levels of noise environments.

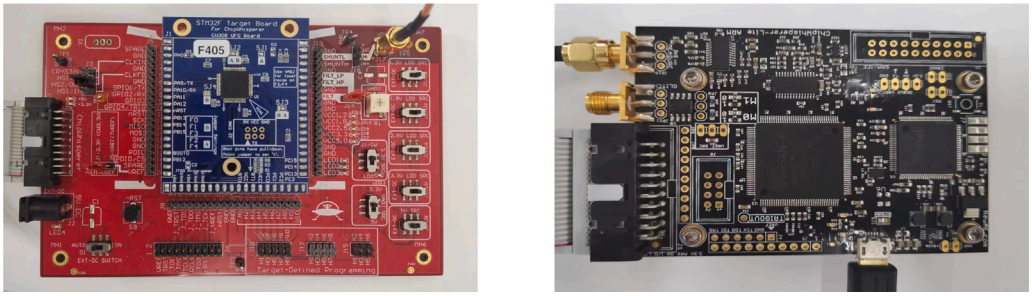
Random delay The introduction of random delays during the execution of encryption algorithms [6,38] is an effective countermeasure against side-channel attacks. Random delays are often modeled using Shifting Deformation (SH_{T^*}) [7]. Let the original length of the trace be L. The SH_{T^*} can change the original size of the trace to $L' = L - T^*$, where T^* represents the maximal amplitude of the trace and L' is the core window size of the trace. During the simulation, a uniform random $t \in [0, T^*]$ needs to be initialized. Then, a window of L' length starting from the tth point is taken each time to form the new random delay dataset. We try different SH_T values of $T \leq T^*$ to compare the impact of different levels of random delay on the leakage of the power trace of the Kyber.

Clock jitter Clock jitter is a classical physical countermeasure against side-channel attacks by introducing clock instability [7]. In this paper, the clock jitter effect is simulated by Add-Remove Deformation (AR_R), which simulates the jitter effect by inserting and deleting time samples in captured traces. In the simulation, the position of the modified time sample follows uniform random selection, and the inserted value is the arithmetic average of the values of the two points before and after. In each trace, we insert and remove R time samples respectively, $R \sim \mathcal{N}(0, \sigma^2)$. Depending on the variance σ^2, high and low jitter datasets are formed. On this basis, we also specify the value of R to simulate different degrees of random jitter for comparison.

In addition to the above three side-channel countermeasures, we also conduct a leakage assessment for scenarios involving low sampling rates. Inevitably, the sampling frequency in practice does not match that of the experimental. We evaluate the effectiveness of these countermeasures in experiments, and these assessments help us better understand and apply these side-channel countermeasures.

3. Trace acquisition

In this section, we introduce the equipment used for trace acquisition and the attack points used in the following experiments.



(a) The target board (b) The ChipWhisperer-Lite board

Fig. 1. Equipment for trace acquisition.

3.1. Equipment

For all subsequent experiments, we use the CW308T-STM32F4 (Fig. 1) as the target device, which consists of a ChipWhisperer-Lite board, a CW308 UFO board, and an ARM Cortex-M4-based STM32F405 target board. ChipWhisperer is a complete open-source toolchain for validating the side-channel resistance of embedded devices [39]. It focuses on power analysis and provides a maximum sampling buffer size of 24,400 samples. Meanwhile, the STM32F4 microcontroller, as a communication interface, acquisition interface, and master control unit, is widely used in IIoT systems.

On the device, we implement the algorithm of Kyber-768 (m4speed version) from **pqm4** library [40] which is submitted to NIST, and its first-order masked algorithm from **mkm4** [11]. Both are PQC schemes based on the ARM Cortex-M4 microcontroller. The Kyber768 parameter set with $\eta_1 = 2$ and $k = 3$, and compile the implementation using **arm-none-eabi-gcc** with the optimization flag "-O3", which is generally the most challenging setting through side-channel analysis [31].

The target board's program execution frequency is set to 24MHz, and we set the sampling frequency is selected to be four times the execution frequency (x4).

3.2. Attack points

To comprehensively evaluate and compare the leakage gap between no mask and first-order mask, we choose the decoding and encoding functions that are widely used in **pqm4** and **mkm4**.

3.2.1. Decoding functions

The decoding function in **pqm4** is $poly\_frommsg$ (Listing 1), which we denote as Decode. It is the process of converting the message m into a polynomial r. It assigns values through $r[i] = \beta \cdot m_i$, $i \in [0, 255]$, where m_i is message bit and β is the center of the integer ring $\mathbb{Z}_q$. Regarding the leakage point in the decoding operation, opinions vary. In [31,35], they focus on the variable $mask$. When the message bit is 0 or 1, the $mask$ value is assigned the corresponding value 0×0000 or 0xFFFF. Since the difference in Hamming weight between these two possible values has reached the extreme value of the current number of bits, a strong leakage can be generated. Using this leakage model, the value of the message bit can be derived. But in the research of other teams, such as in [36], they are more concerned about the code of the variable r in the subsequent sentence, which converts the message share from the Boolean domain to the polynomial domain. This is also referred to as *determiner-leakage* [31,34,41]. These researches collectively demonstrate that the leakage from the decode function is relatively easy to exploit in side-channel attacks.

```
void poly_frommsg(poly *r,const unsigned char msg[32])
{
    int i,j;
    uint16_t mask;
    for (i=0;i<32;i++) {
        for (j=0;j<8;j++) {
            mask= - ((msg[i] >> j)&1);
            r->coeffs[8*i+j]=mask&((q+1)/2);
        }
    }
}
```

Listing 1: Decode: poly_frommsg

In the implementation process of the first-order mask **mkm4**, the implementation function of the decoding operation is changed into $masked\_poly\_frommsg$, which we call Masked_Decode. This function splits message m into m' and $m\varepsilon$, and performs two cyclic operations respectively. While this scheme prevents us from directly obtaining the value of m, since $m = m' \oplus m''$, we can deduce the values of m' and m'' separately. Then, we can proceed with the second-order attack to recover the message m.

```
masked_poly_frommsg:
    ...
    ldrsh lr, [r0]
    sbfx r3, r3, #0, #1  @ mask <- msg.share[0]
    and r3, ip, r3
    adds r2, r2, 1
    add r3, r3, lr
    cmp r2, 8
    strh r3, [r0], #2    @ store r->polys[0]
    bne .L4
    ...
    sbfx r3, r3, #0, #1  @ mask <- msg.share[1]
    ...
    strh r3, [r1], #2    @ store r->polys[1]
    ...
```

Listing 2: Masked_Decode: masked_poly_frommsg

3.2.2. Encoding functions

The encoding function of **pqm4** is *poly_tomsg* (Listing 3), which can encode each polynomial separately and concatenate the output byte arrays to form message m. We denote it as Encode. Upon entering the function, m is initialized to 0, and as the loop progresses, it is incrementally updated bitwise by $r[i]$. Analyzing the assembler code shows that the updated bits are stored using the STRB command at the end of each inner loop. This presents an obvious side-channel leakage for message recovery attacks.

The implementation of *masked_poly_tomsg* in **mkm4** is similar to this, and we refer to this function as Masked_Encode. After generating its assembly code (Listing 4), we observed that its storage operations are essentially the same as those of the unmasked *poly_tomsg* implementation. However, instead of storing the message m only once, it stores m' and m'' separately. This allows the leakage analysis of m' and m'' to share the same loop traces, rather than collecting them separately.

```
void poly_tomsg(unsigned char msg[32], poly *r)
{
    uint16_t t;
    int i,j;
    for (i=0;i<32;i++) {
        msg[i]=0;
        for (j=0;j<8;j++) {
            t=(((r->coeffs[8*i+j]<<1)+q/2)/q)&1;
            msg[i] |= t << j;          }
    }
}
```

Listing 3: Encode: poly_tomsg

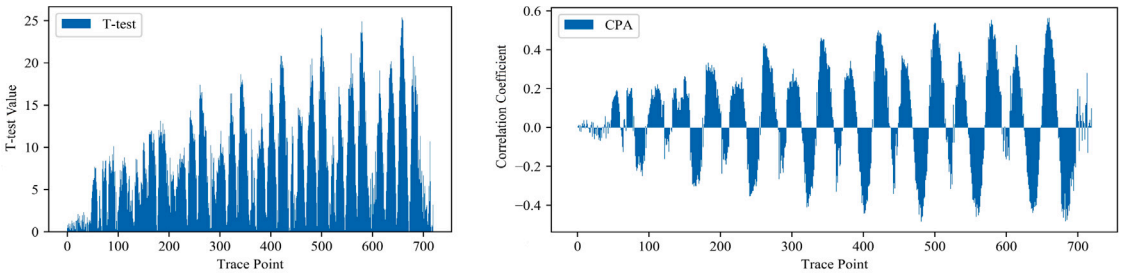
```
masked_poly_tomsg:
    ...
    add r6, r6, r1
    cmp r7, #8
    strb r0, [r4]       @ store msg->share[0]
    strb r6, [r4, #32]      @ store msg->share[1]
    bne .L21
```

Listing 4: Masked_Encode: masked_poly_tomsg

Let $m[i,j]$ denote the jth bit of the ith byte of m. After each update of $m[i,j]$, the STRB instruction updates the entire $m[i]$ to storage. This allows the HW of the updated bits in $m[i,j]$ to be recoverable [34]. Through the sequential recovery of HW, we can reconstruct the entire $m[i]$, and consequently, the entire message m can be recovered.

The specific inference rules are as follows:

$$m[i,j] = \begin{cases} 0, & \text{if } \mathrm{HW}(m[i,j]) = \mathrm{HW}(m[i,j-1]) \\ 1, & \text{if } \mathrm{HW}(m[i,j]) = \mathrm{HW}(m[i,j-1]) + 1 \end{cases}$$



(a) T-test value (b) Pearson correlation coefficient

Fig. 2. The leakage of Encode.

Table 2
Leakage assessment results for the initial datasets of the four attack points.

Operation	Results		
	V_m	S_m	C_m
Decode	18.81	1.27	0.38
Encode	25.39	2.07	0.56
Masked_Decode	25.16	2.05	0.34
Masked_Encode	66.44	70.84	0.89

4. Leakage assessment results

In this section, we simulate various countermeasures against side-channel attacks at four different attack points. Then, we conduct leakage assessments and comparisons under different implementation conditions.

Firstly, we apply evaluation techniques such as T-test, SNR, and CPA to assess and analyze the leakage of the collected Kyber (unmasked/first-order masked) original dataset. Due to the influence of noise environment and countermeasures such as misalignment implemented by manufacturers during the practical application, we also evaluate the leakage under different noise environments, different degrees of clock jitter and random delays, and even at low sampling rates. This evaluation aims to assess the leakage characteristics of the attack points and lay the foundation for the subsequent analysis of attack and defense costs.

4.1. The initial dataset of Kyber

We collected the initial data sets for the four attack points separately: Decode, Encode, Masked_Decode, and Masked_Encode. Each dataset consists of 1600 traces and chooses HW as its label. The leakage assessment process reveals a gradual increase in the leakage value of a bit during the encoding operation (Fig. 2). This can be attributed to the initial value of msg being set as 0 in the encoding process, and subsequent bitwise assignments within the loop leading to an incremental correlation with the leakage value. The decision was made to conduct detection and analysis on a byte-by-byte basis.

To enhance the quantification of evaluation results for attack points in each case, we opt to utilize the maximum value within each interval as a representation. Let V_m be the maximum T-test value, S_m the highest SNR, and C_m the maximum Pearson correlation coefficient within the range. For encoding operations, these values are usually located in the last bit. Table 2 summarizes the obtained results. For each dataset, we see eight distinct spikes in the leak assessment.

As we can see from Table 2, the leakage and correlation of the power traces of Masked_Encode are the highest, and its V_m reaches 66.44. Even the lowest value of the V_m is well above the 4.5 threshold. This shows that they all have a significant leakage and the corresponding message m can be easily extracted. Moreover, we found that C_m is a more stable indicator than V_m in our experiments. So, in the following experiments, the importance order of evaluation metrics is arranged as C_m, V_m, S_m. We set the threshold of 0.1 for C_m to correspond to the threshold of 4.5 for V_m, indicating significant leakage in the dataset.

4.2. Datasets with added noise

Noise is the most frequently occurring influencing factor in our daily acquisition, and Gaussian noise is the most common one. we simulate different levels of noise environment for four datasets by adding the effect of Gaussian noise to the original datasets to evaluate the resistance of the traces of the attack point to noise.

We control the degree of added noise by controlling the numerical value of SNR, denoted by N_{SNR}. The SNR is inversely proportional to the degree of noise influence. We apply SNR $\in \{5, 3, 1, 0.1\}$ for leakage assessment.

In Table 3, the decoding function is significantly more affected by noise than the encoding function. When SNR $= 0.1$, both the V_m and the C_m of Decode are reduced by about 80%, and the C_m of Masked_Decode is reduced by 70.6%. Among them, Encode is

Table 3
Leakage assessment results of four attack points under different noise levels (represented by SNR in N_{SNR}).

		N_5	N_3	N_1	$N_{0.1}$
Decode	V_m	7.24	5.79	5.90	3.40
	S_m	0.36	0.27	0.21	0.18
	C_m	0.19	0.16	0.18	0.08
Encode	V_m	23.59	22.52	20.29	19.67
	S_m	1.75	1.62	1.33	1.15
	C_m	0.55	0.52	0.40	0.46
Masked_Decode	V_m	6.26	4.11	4.22	4.16
	S_m	0.50	0.39	0.37	0.36
	C_m	0.12	0.12	0.11	0.10
Masked_Encode	V_m	32.13	25.24	23.64	21.05
	S_m	4.12	2.57	1.34	1.30
	C_m	0.52	0.45	0.39	0.37

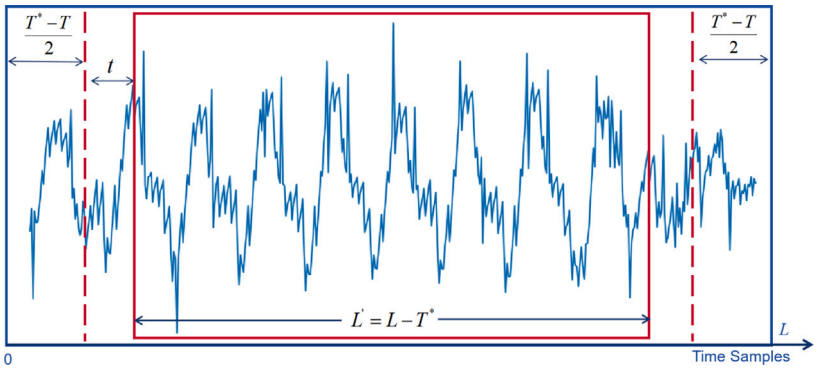


Fig. 3. Illustration of shifting technique principle.

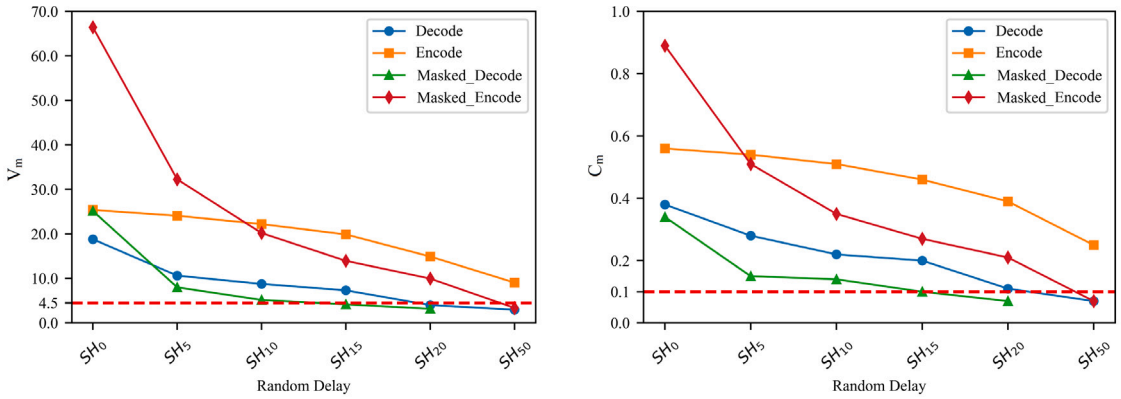
the least affected with only a slight reduction, while Masked_Encode, as the attack point with the highest correlation coefficient, also reduces the leakage by more than half.

This result may be related to the overall flatter leakage of the decoding traces, which is more susceptible to the noise of small vibrations than the rapidly rising leakage waveform of the encoding traces. As for Encode and Masked_Encode, the leakage points in Masked_Encode account for a smaller proportion of the entire trace. Despite having higher leakage values, they also require higher classification precision and are thus more susceptible to noise interference. Therefore, in practical environments with noisy conditions, side-channel attacks on decoding operation are more difficult than that of encoding operation. On the other side, we can reduce the SNR of decoding operation by introducing randomization technology or adding components around the chip, thus enhancing protection.

4.3. Random delay processing

Introducing a random delay in the execution of the algorithm may break the synchronization and increase the complexity of the attack [42], which can effectively prevent side-channel attacks. In the experiment, the random delay is simulated by SH_{T^*}. we set $T^* = 50$ and $T \in \{5, 10, 15, 20, T^*\}$, which means that the length of all traces is reduced by T^* and delayed by at least $\frac{T^*-T}{2}$ time points (Fig. 3). Its processing steps include: (1) Choose an appropriate T^*, which represents the maximum accepted amplitude and also determines the dimension of the processed output traces to be $L' = L - T^*$, where L is the original length of the traces. (2) Choose a different $T \in [0, T^*]$. Discard the length $\frac{T^*-T}{2}$ of the head and tail of the trace. This interval remains unused. (3) In the rest of the range, generate a uniformly random $t \in [0, T]$, which is the starting point of the shifting window. We start from this point, select a trace of length L', and add it to our generated random delay dataset. In this process, the trace size in our generated data set is always constant, as T^* is fixed.

In our original datasets, Masked_Encode has the most data points in the power trace, with 8000 points, and the least is Decode with only 480 points. However, their correlations are all within the range of $T \leq 50$, reduced to below 0.1, as shown in Table 4. Among them, the attack point where the C_m decreases to below 0.1 the fastest is Masked_Decode, consisting of 500 points. The only attack point that is difficult to protect against using random delay countermeasures is Encode, with an original trace length of 720 points. In this case, it shows that the resistance of the data shifting is independent of the proportion of the delay range to the whole trace. From Fig. 4, it can be observed that Masked_Encode exhibits the fastest rate of decline, indicating its weakest resistance to shifting. Although it has the longest acquisition trace and the highest initial leakage and correlation, when $T = 5$, these values decrease by nearly half.



(a) The change in the maximum T-test value (V_m) over the range (b) The highest Pearson correlation coefficient (C_m) change

Fig. 4. Variation of leakage results with changing T when using SH_T to simulate random delays with fixed $T^* = 50$.

Table 4
Leakage assessment results at various T with Fixed $T^* = 50$.

		SH_5	SH_{10}	SH_{15}	SH_{20}	SH_{50}
Decode	V_m	10.62	8.76	7.32	4.00	2.97
	S_m	0.38	0.30	0.31	0.26	0.22
	C_m	0.28	0.22	0.20	0.11	0.07
Encode	V_m	24.08	22.21	19.89	14.92	9.05
	S_m	1.72	1.45	0.94	0.87	0.46
	C_m	0.54	0.51	0.46	0.39	0.25
Masked_Decode	V_m	8.00	5.16	4.16	3.18	
	S_m	0.44	0.41	0.39	0.29	
	C_m	0.15	0.14	0.10	0.07	
Masked_Encode	V_m	32.26	20.20	13.95	9.95	3.37
	S_m	3.79	1.58	0.50	0.89	0.77
	C_m	0.51	0.35	0.27	0.21	0.07

Table 5
The extended experiment: Increasing the range of random delays simulated by $SH_{T^*}(T = T^*)$ to identify values of T that yield leakage results of Encode below the threshold.

		SH_{60}	SH_{70}	SH_{75}	SH_{80}
Encode	V_m	6.72	4.75	4.73	2.88
	S_m	0.34	0.11	0.33	0.15
	C_m	0.16	0.12	0.10	0.08

Based on the results, we can conclude that random delay offers significant protection for the Decode, Masked_Decode, and Masked_Encode. An effective shifting range of fewer than 50 points is sufficient. For Encode, this approach also has a certain protective effect. However, to noticeably reduce the leakage correlation, the power traces of this attack point require a larger range of random delays. As we can see in Table 5, when selecting the remaining attack points, reducing the C_m below 0.1 may require random delays covering less than 10% of the entire trace length. Therefore, we can conclude that as long as the random delay range T reaches 10% of the time samples in the traces, effective protection and counteraction against attacks on these operations can be achieved.

4.4. Simulation of clock jitter

Clock jitter is a side-channel countermeasure that is usually implemented in physical devices. We chose to simulate the clock jitter effect by using the AR_R method. To simulate the effect of clock jitter, we deform each trace by adding and removing R points separately, where $R \sim \mathcal{N}(0, \sigma^2)$. We first treat the base jitter effect dataset as high jitter and low jitter, where we took $\sigma^2 = 4$ for low jitter and $\sigma^2 = 36$ for high jitter. Then, we perform the same Add-Remove operation in both datasets and compare the differences. We construct a loop that repeats R times, producing the insertion position $p \sim \mathcal{U}(1, len - 1)$ as the arithmetic average of the current value and the previous value. After the insertion is done, R points are deleted by uniform random selection again in the same way. In Fig. 5, we can observe the differences among the original dataset, the dataset with high jitter effects, and the traces processed based on the high jitter effects with AR_{200}.

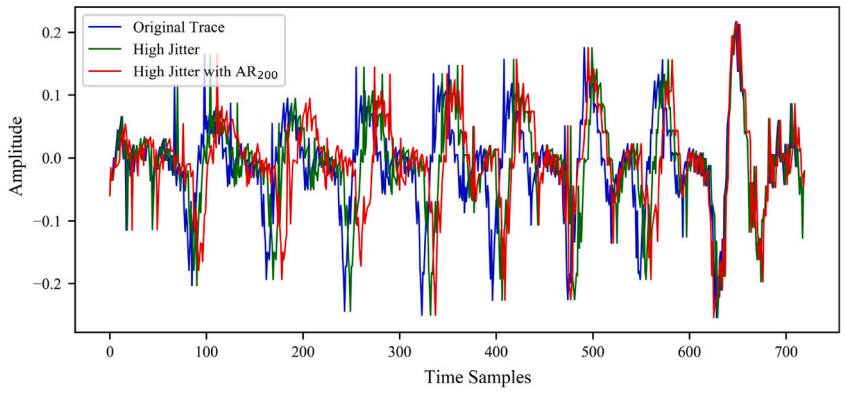


Fig. 5. Comparison of traces between the original Encode's dataset, dataset with high jitter effects, and traces processed based on high jitter effects with AR_{200}.

Table 6
Leakage assessment results of artificially generated low-jitter countermeasure with different R.

		AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}	AR_{3000}
Decode	V_m	11.74	7.82	5.77	5.60	4.53	
	S_m	0.52	0.73	0.35	0.27	0.25	
	C_m	0.29	0.21	0.16	0.14	0.08	
Encode	V_m	25.61	24.58	22.86	19.97	18.09	
	S_m	1.92	1.76	1.64	1.30	0.92	
	C_m	0.57	0.54	0.51	0.47	0.43	
Masked_Decode	V_m	11.08	4.83	3.76			
	S_m	0.95	0.47	0.25			
	C_m	0.15	0.12	0.09			
Masked_Encode	V_m	59.11	38.46	30.49	27.97	19.66	5.35
	S_m	33.52	4.85	3.14	2.08	1.72	0.54
	C_m	0.80	0.56	0.50	0.45	0.40	0.16

Table 7
Leakage assessment results for artificially generated high-jitter countermeasures with different R.

		AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}	AR_{3000}
Decode	V_m	7.53	6.19	5.31	4.81	3.82	
	S_m	0.33	0.18	0.30	0.34	0.23	
	C_m	0.19	0.16	0.15	0.11	0.10	
Encode	V_m	24.54	22.04	21.24	19.15	16.53	
	S_m	1.84	1.57	1.31	1.38	1.16	
	C_m	0.54	0.50	0.49	0.46	0.40	
Masked_Decode	V_m	4.57	3.86	4.22			
	S_m	0.42	0.35	0.42			
	C_m	0.14	0.11	0.10			
Masked_Encode	V_m	41.12	34.71	28.91	25.08	20.67	5.62
	S_m	5.40	3.04	2.47	2.40	1.00	0.68
	C_m	0.59	0.53	0.47	0.40	0.35	0.16

In the leakage assessment, we set $R \in \{0, 20, 50, 100, 200, 3000\}$. We chose these values based on the dimension of the dataset and the effect of jitter. From Tables 6 and 7, we can see the corresponding evaluation results. In terms of AR_0, the most difference in correlation affected by random high and low jitter is Decode. However, as the number of subsequent jitter points R increases, the distinction between high and low jitter effects starts to diminish. In the low jitter datasets, except that the initial increase of jitter points from AR_0 to AR_{20}, which leads to a relatively sharp reduction of the leakage of Decode and Masked_Decode, the subsequent increase in R leading to jitter effects had a relatively mild impact. The C_m for these two functions also drops to below 0.1 the fastest.

We can observe that in both the low-jitter datasets and the high-jitter datasets, the values of R at which they reach the threshold are similar. For Decode, its critical R value is around 200, and the number of jitter points accounts for 40% of the time samples. For Masked_Decode, this value is around 50, which is 10% of the whole trace and is the attack point most susceptible to clock jitter effects. However, for encoding operations, they can withstand extremely high jitter effects and it is difficult to provide effective protection. For Masked_Encode, the parameter corresponding to the limit state falls between AR_{3000} and AR_{4000}, which

Table 8
Leakage assessment based on the combination of low jitter and shifting countermeasures: For each combination, four resulting values are given (a) the V_m for Encode, (b) the C_m for Encode, (c) the V_m for Masked_Encode, and (d) the C_m for Masked_Encode.

a c / b d	SH_{10}		SH_{20}		SH_{50}		SH_{70}
AR_{50}	20.20	16.38	14.86	9.27	7.83	4.23	4.00
	0.47	0.31	0.37	0.23	0.20	0.09	0.09
AR_{100}	18.29	17.62	12.77	8.00	6.92		
	0.42	0.30	0.33	0.22	0.17		
AR_{200}	16.48	14.35	13.24	8.40	6.89		
	0.40	0.29	0.31	0.23	0.17		
AR_{3000}		4.27		4.63			
		0.14		0.12			

Table 9
Leakage assessment based on the combination of high jitter and shifting countermeasures.

a c / b d	SH_{10}		SH_{20}		SH_{50}		SH_{70}
AR_{50}	19.49	15.10	12.63	9.98	7.98	3.40	4.04
	0.46	0.32	0.33	0.22	0.21	0.08	0.11
AR_{100}	16.25	17.25	12.95	8.92	6.78		
	0.41	0.29	0.32	0.20	0.18		
AR_{200}	16.03	12.89	10.51	8.18	5.00		
	0.39	0.27	0.27	0.19	0.12		
AR_{3000}		5.15		4.58			
		0.15		0.11			

represents more than 40% of the trace points. While the highest value remains above the threshold at AR_{4000}, the correlation result plot has already become scattered. Nonetheless, when considering only whether it falls below the threshold, applying the AR_{8000} countermeasure, which corresponds to jitter 100% points, reduces C_m to 0.11. Meanwhile, We applied the AR_{700} countermeasure with high jitter to Encode, with a total length of 720 points, but the V_m and C_m results remained at 13.22 and 0.32 respectively, indicating a high level of leakage. Therefore, we attempt to combine random delay and clock jitter countermeasures ($SH_T AR_R$) to conduct additional leakage assessments.

For Tables 8 and 9, we conduct further experimental parameter settings, *i.e.* $SH_{70}AR_{50}$, To explore the effects of stacking countermeasures. As expected, the stacked countermeasure accelerates the reduction of leakage value and correlation, but AR_R has a much slower impact than SH_T. For example, in the case of Encode, the C_m of $SH_{50}AR_{200}$ under the high jitter and SH_{70} without jitter reach the same value. It can be observed that Encode approaches the threshold at SH_{70} and $SH_{50}AR_{200}$, while Masked_Encode approaches the threshold at $SH_{20}AR_{3000}$ and SH_{50}. The 200 and 3000 represent 27.8% and 37.5% of their trace points, respectively.

4.5. Low sampling rate datasets

In practice scenarios, another possible difficulty in acquiring traces from IIoT devices is that you will not be able to achieve a sufficient sampling rate. Thus, we use a sampling rate that is consistent with the running frequency (x1), to perform a simple low sampling rate simulation for its leakage detection.

By comparing the leakage assessment results of fourfold (x4) sampling rate in Table 2 and single (x1) sampling rate in Table 10, it is the Decode and Encode without the mask that affects more in terms of correlation, but this is within the acceptable fluctuation range and cannot represent anything. These data precisely illustrate that the leakage value and correlation are not greatly affected as the sampling rate is reduced, even though our sampling rate is reduced by four times. There is still a high risk of being attacked in low sampling rate mode, the only change is that their traces are shorter and harder to detect.

5. Deep learning SCA results

In this section, we use the neural network to carry out side-channel attacks on the four original datasets respectively and test the classification accuracy of the model trained on the initial state under the influence of different countermeasures.

Table 10
Leakage assessment results for x1 sampling rate datasets of four attack points.

Operation	Results		
	V_m	S_m	C_m
Decode	18.89	1.12	0.32
Encode	22.98	2.15	0.52
Masked_Decode	14.52	0.84	0.35
Masked_Encode	65.90	68.12	0.89

Table 11
The MLP architecture is used uniformly in the message recovery attacks at the four attack points. The numerical value of input_size depends on the length of traces at different points.

Layer type	(Input, output) shape
Batch normalization 1	(input_size, input_size)
Dense 1	(input_size, 256)
Batch normalization 2	(256, 256)
Dropout 1	(256, 256)
Dense 2	(256, 512)
Batch normalization 3	(512, 512)
Dropout 2	(512, 512)
Dense 3	(512, 1024)
Batch normalization 4	(1024, 1024)
Dropout 3	(1024, 1024)
Dense 4	(1024, 512)
Batch normalization 5	(512, 512)
Dropout 4	(512, 512)
Dense 5	(512, 256)
Softmax	(256, 256)

5.1. Model training

We use MLP to conduct message recovery attacks and set the label to the message value, the detailed MLP architecture in Table 11. Specifically, the model consists of an input layer, five dense layers, and a softmax output layer. The output shapes of the dense layer are 256, 512, 1024, 512, and 256, respectively, and use the ReLU activation function. In addition, the model contains batch normalization and dropout operations for regularization and optimization. During training, we selected the categorical cross-entropy loss function to measure the difference between the probability distribution of byte values predicted by the model and the actual byte value distribution. The optimizer used was the Adam algorithm, which combines momentum and adaptive gradient methods, with a learning rate set at 0.05 (Masked_Encode's MLP use 0.0001). The evaluation metric is accuracy, as it is an intuitive measure of the proportion of samples correctly predicted by the model. The batch size is 32. We used 1K traces of the first byte for input and finally predicted the probabilities of the byte value between 0 and 255 through the softmax layer. 80% of the dataset is used for training and 20% for validation.

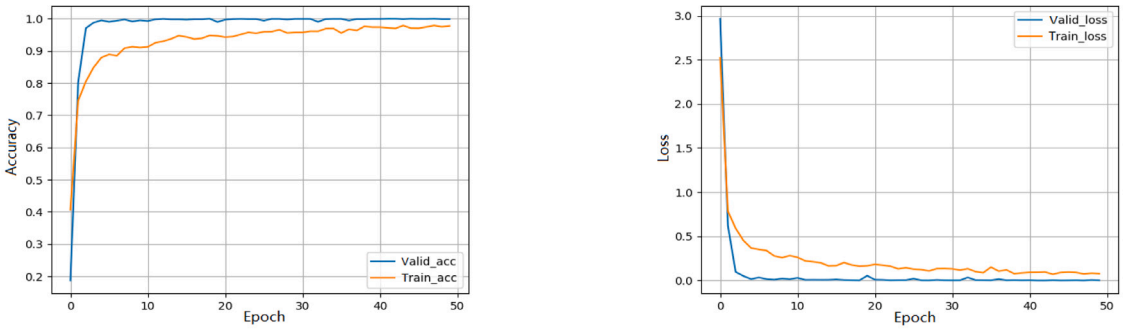
Different from the bitwise network training of [43], each byte is divided into 8 bits. We directly use the trace data of bytes for training. In order to better evaluate the impact of the various countermeasures, we did not choose to take the whole trace of the loop and then crop it to 32 bytes as in [13,43,44]. This allows our classification results close to the theoretical highest on the initial dataset. It is important to note that since Masked_Decode and Masked_Encode process m' and m'' in the same way, our experiments only target the first byte of m'. Therefore, the accuracy of attacking these two functions to recover the message m should be the square of the accuracy we provide.

5.2. Decode

In the process of training the attack model against the Decode, we set a total of 50 epochs. In Fig. 6, within 10 epochs, the classification accuracy on the validation set is already close to 100%. We tested the model on an initial dataset of 1600 traces, achieving an accuracy of 99.94%. In Table 12, the trained MLP model is first tested on datasets in environments with varying levels of SNR. The model's accuracy was observed to be affected by these different SNR levels. Unlike previous leakage assessments, the C_m at N_1 showed a slight upward fluctuation due to randomness, while the accuracy consistently decreased, more precisely reflecting the extent of the impact. Additionally, when SNR = 0.1, the accuracy dropped below 1%, aligning with the C_m.

Even though the accuracy of $N_{0.1}$has dropped to 0.37%, it has learned some features so that the accuracy does not drop to zero. Meanwhile, under these conditions, we can still obtain the correct message within up to 105 enumeration attempts.

Unlike the impact of noise, although random delays have a significant effect on the results of V_m and C_m, this effect does not extend to the classification results of deep-learning models. When handling datasets affected by random delay effects, we take the midpoint of the traces and pad the front and back of the trace with zeros of equal length (adding an extra zero at the end, if the gap is odd) until it meets the input size requirements of the MLP. In Table 13, we can observe that random delay, which does not



(a) Training and validation accuracy (b) Training and validation loss

Fig. 6. Performance in model training against Decode.

Table 12
The message recovery accuracy of MLP for Decode with different SNR of noise.

	N_5	N_3	N_1	$N_{0.1}$
Accuracy (%)	19.81	11.63	6.37	0.37

Table 13
The message recovery accuracy of MLP for Decode with random delay for different T.

	SH_5	SH_{10}	SH_{15}	SH_{20}	SH_{50}
Accuracy (%)	36.94	19.81	13.56	9.56	4.19

Table 14
The message recovery accuracy of Decode's MLP in low jitter with artificially generated different R.

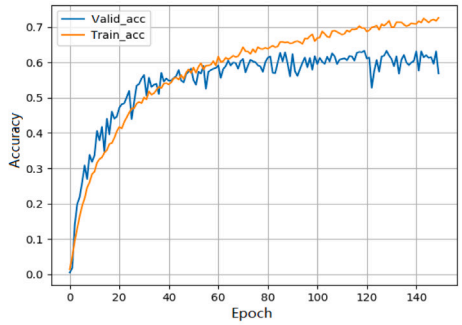
Low_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}
Accuracy (%)	50.32	6.62	2.13	1.81	1.69

Table 15
The message recovery accuracy of Decode's MLP in high jitter with artificially generated different R.

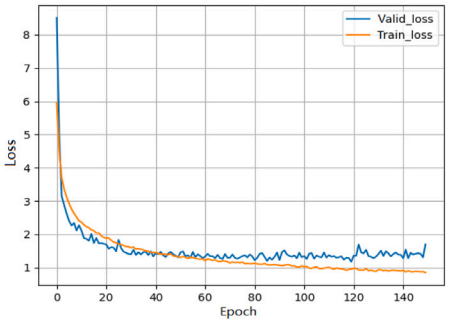
High_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}
Accuracy (%)	10.37	4.00	1.94	1.69	1.13

cause some distortion to the original traces, is not enough to combat the deep learning-based attack. When $T = 50$, even though its V_m is far below the threshold and very messy, we can still see relatively aggregated peaks from the correlation coefficient, and its trace characteristics are preserved. These features are very convenient for feature extraction in deep learning, so it still retains a relatively high accuracy of 4.19% here. From the previous comparison, it is clear that the countermeasure of adding deformation to the power traces of decoding operations is more effective against deep learning-based attacks, but for T-test value, a small delay can quickly degrade its results.

Clock jitter is also a common countermeasure to deform traces. We can observe that the high and low jitter effects make a large difference at the beginning. In Tables 14 and 15, at AR_0, the high-jitter countermeasure reduced the model's accuracy to 10.37%, compared to 50.32% for the corresponding low-jitter countermeasure. This gap quickly narrowed after the implementation of Add-Remove Deformation, and by AR_{100}, the accuracy of the two had become very close. It can be observed that the trend in accuracy changes is generally consistent with the trend in C_m. However, at AR_{200}, while the impact of different variances between high and low jitter on correlation can be disregarded and may even be reversed, there is still an effect on the message recovery accuracy. When the C_m is similar, there is a certain gap between the accuracy of $N_{0.1}$ and AR_{200}, which we speculate is related to the fact that the distribution of correlation coefficient in the dataset with SNR = 0.1 is more disrupted than that with AR_{200}. The influence and deformation caused by high noise have a wider coverage and larger deformation than the deformation caused by add and remove time samples. This result indicates that adding noise to decoding operations is a more effective countermeasure against deep learning-based attacks than other countermeasures.



(a) Training and validation accuracy



(b) Training and validation loss

Fig. 7. Performance in model training against Encode.

Table 16
The message recovery accuracy of MLP for Encode's dataset with different levels of noise.

	N_5	N_3	N_1	$N_{0.1}$
Accuracy (%)	7.81	5.69	3.69	3.31

Table 17
Message recovery accuracy of model attacks on Encode's dataset with random delays for different T with $T^* = 50$.

	SH_5	SH_{10}	SH_{15}	SH_{20}	SH_{50}
Accuracy (%)	25.87	21.13	17.56	15.50	6.44

Table 18
The message recovery accuracy of Encode's MLP in low jitter with artificially generated different R.

Low_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}
Accuracy (%)	46.50	19.56	13.63	6.69	4.06

5.3. Encode

The accuracy of Encode's MLP attack model is lower than that of the decoding function. In Fig. 7, we set 150 epochs and gradually obtained an accuracy higher than 60% after 65 epochs, and we used this model against 1600 traces of the original dataset and got an accuracy of 66.12%. It aligns with the calculated single-trace message recovery accuracy upper limit of 62% in [44], and the training limit has been reached. Due to the high initial C_m of the Encode and its strong resistance to noise in the leakage evaluation, the model's accuracy remains 3.31% even when noise is added to achieve an SNR $= 0.1$, which is significantly higher than that of the Decode. When the noise level gradually increases, the accuracy of the Encode's attack model shows a trend of steady and low-speed reduction (Table 16). At the same time, we observe that at N_5, the reduction in this model's accuracy is higher than that of the attack model for the decoding operation, which contradicts our leakage assessment results. This may be related to its stepped correlation, as the encoding operation only leaks complete 8-bit data operations at the final bit. The smaller and more concentrated the learning interval, the higher the model's sensitivity to interference.

For random delays, we achieved acceptable accuracy consistent with the test results against the attack model targeting Decode. When T^* is fixed to 50, the relationship between parameter T and accuracy is shown in Table 17. In addition, we conducted tests on the accuracy of the countermeasure with $T \in \{60, 70, 75, 80\}$ for Encode in Table 20. The SH_{75} is the operation that reduces C_m to a threshold of 0.10, and it maintained an accuracy of 4.16%. We can observe that when T is between 70 and 80, the accuracy fluctuates and remains around 4% without a significant drop. This is sufficient to demonstrate the limitations of standalone random delay countermeasures in addressing deep learning-based side-channel attacks. It is better suited for use in conjunction with other countermeasures.

In Tables 18 and 19, the high leakage value of the initial traces and the lengthening of the traces leading to a reduced proportion of deformation points (27.78%) resulted in an attack accuracy higher than the model attacking Decode. Due to the different degree of deformation, despite the similar C_m values observed when applying AR_{200} combined with high/low jitter and $N_{0.1}$, the AR_{200} provides less protection against trace vulnerabilities compared to $N_{0.1}$. We attempted to combine random delays and high clock jitter countermeasures and measured the accuracy of the attack model under these conditions (Table 21).

Table 19
The message recovery accuracy of Encode's MLP in high jitter with artificially generated different R.

$High\_Jitter$	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}
Accuracy (%)	22.69	9.19	9.75	5.25	3.50

Table 20
The extended experiments: Accuracy of model attacks on Encode's dataset with random delays for different T ($T^* = T$).

	SH_{60}	SH_{70}	SH_{75}	SH_{80}
Accuracy (%)	5.94	4.06	4.16	3.94

Table 21
Effectiveness of combined high-jitter and random delay countermeasures against Encode's attack models, measured by message recovery accuracy.

Accuracy (%)	SH_{10}	SH_{20}	SH_{50}	SH_{70}
AR_{50}	6.87	6.06	3.06	2.00
AR_{100}	4.87	3.62	2.37	1.94
AR_{200}	3.25	2.56	2.19	1.31

Table 22
The message recovery accuracy of Masked_Decode's MLP with different levels of noise.

	N_5	N_3	N_1	$N_{0.1}$
Accuracy (%)	2.50	2.00	1.00	0.88

Table 23
The message recovery accuracy of Masked_Decode's MLP with random delay for different T.

	SH_5	SH_{10}	SH_{15}	SH_{20}	SH_{50}
Accuracy (%)	27.50	15.00	9.56	7.06	3.38

We observe that when more influence is applied to traces, the accuracy reduction of adding the same influence factor is smaller due to the lower initial accuracy. This resistance makes it relatively easy to reduce the attack model's accuracy to single-digit percentages, but achieving an accuracy of 1% or lower requires further intensifying the effect of jitter countermeasures or switching to countermeasures, which have a greater impact. Nevertheless, given the initial leakage in the Encode, the combined countermeasure has already achieved a certain level of effectiveness.

5.4. Masked_Decode

The execution processes of Masked_Decode and Decode are similar, which makes the model training procedures the same. Also, due to m' and m'' are processed in the same way, we train a single model specifically for m', which can also be applied to recover m''. The accuracy of this attack point in the following is the message recovery attack accuracy against m'. If the accuracy of the attack on m is involved, the result should be squared. The model tested on the initial dataset achieved an accuracy of 99.06% in recovering m'.

Consistent with Decode, the countermeasure of adding noise in Table 22 still has a significant impact on the attack model targeting Masked_Decode. At $N_{0.1}$, the model accuracy decreased to 0.88% < 1%. Although this value is slightly higher than that of Decode, when comparing the accuracy results under the complete set of four levels of noise, Masked_Decode is actually more susceptible to influence than Decode. When adding a slight noise, such as SNR = 5, the model accuracy of Decode still retains 19.81%, but the accuracy of the Masked_Decode model has already plummeted to 2.50%. In addition, the model of Masked_Decode maintained an expected minimum accuracy of 3.38% under the countermeasure of random delay (Table 23) and showed extremely low resistance under the clock jitter countermeasure.

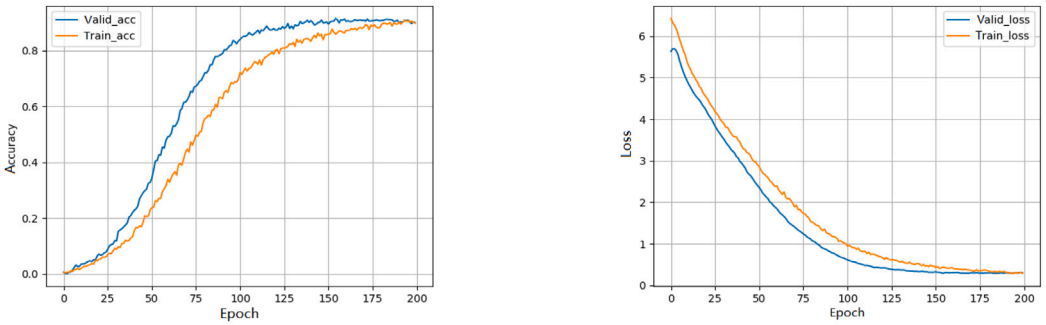
Based on the accuracy data of incorporating noise and clock jitter, we can conclude that this attack model targeting Masked_Decode exhibits low resistance to countermeasures causing distortion. This conclusion is consistent with the leakage assessment results. In Tables 24 and 25, we observe a notably drastic decrease in accuracy from AR_0 to AR_{20} compared to other models. This decline continues as distortion intensifies, eventually dropping to below 1%, representing the lowest accuracy achieved by this countermeasure to date. This indicates that both noise and clock jitter can provide effective protection against this vulnerability.

Table 24
The accuracy of Masked_Decode's MLP in low jitter countermeasures with artificially generated different R.

Low_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}
Accuracy (%)	50.89	3.75	2.37	1.44	0.81

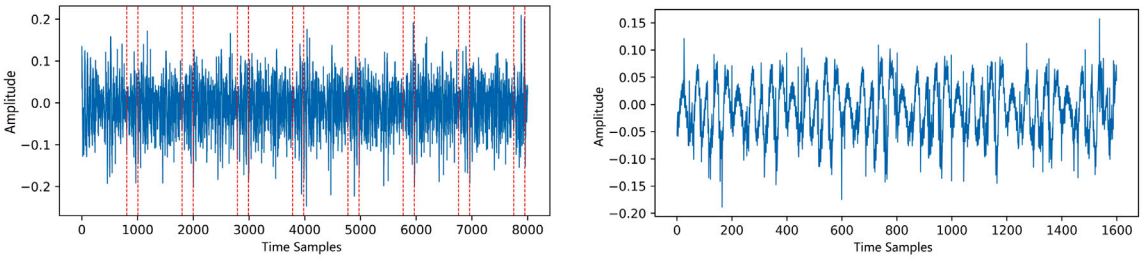
Table 25
The accuracy of Masked_Decode's MLP in high jitter countermeasures with artificially generated different R.

High_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}
Accuracy (%)	10.88	2.94	1.00	1.31	0.75



(a) Training and validation accuracy (b) Training and validation loss

Fig. 8. Performance in model training against Masked_Encode.



(a) Original trace (b) Trace composed of clipped segments

Fig. 9. Preprocessing the trace. Select the intervals of 100 points before and after the highest T-test values of each of the 8 bits in a byte, clip and concatenate these segments to form a new preprocessed dataset with 1600 (8*200) points.

5.5. Masked_Encode

Similar to attacking Masked_Decode, at Masked_Encode, we still choose to attack m'. Since the trace length of the initial dataset of Masked_Encode is much higher than that of other attack points, the proportion of points directly related to the m' is low. Therefore, when training the attack model, it is very prone to overfitting. The accuracy is very difficult to improve, remaining far below the expected accuracy in high-leakage scenarios. So, we perform a preprocessing on the Masked_Encode dataset, clipping out parts of traces that are not directly related to the message. We analyze the T-test results of the initial dataset, selecting the highest T-test value point from each bit operation in the byte. Then, we extract the interval of 100 points before and after this point, such as Fig. 9. By concatenating these segments of 200 points for each of the 8 bits, we form the processed dataset. At the same time, the model learning rate is reduced to 0.0001 and the number of epochs is set to 200.

The preprocessing operation effectively reduces the difficulty of training the attack model. In Fig. 8, we can see its training effect, and the accuracy of attacking the processed dataset reaches 88.13%, which is higher than the accuracy of attacking Encode. We apply the same preprocessing to the datasets affected by noise, random delays, and clock jitter countermeasures to adapt them to the input size of the MLP model. Among them, the dataset with random delays has already been processed with padding.

From the attack accuracy shown in Table 26, for the noise countermeasure, both Masked_Encode and Encode exhibit consistent performance, demonstrating resistance to noise interference. Although the model attack accuracy drops significantly with a small

Table 26
The message recovery accuracy of MLP for Masked_Encode with different SNR of noise.

	N_5	N_3	N_1	$N_{0.1}$
Accuracy (%)	14.12	8.06	5.94	3.88

Table 27
The message recovery accuracy of MLP for Masked_Encode with random delay for different T.

	SH_5	SH_{10}	SH_{15}	SH_{20}	SH_{50}
Accuracy (%)	3.68	1.68	1.27	1.01	4.20

Table 28
The message recovery accuracy of Masked_Encode's MLP in low jitter countermeasures with artificially generated different R.

Low_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}	AR_{3000}
Accuracy (%)	38.56	4.25	1.81	1.50	1.19	0.63

Table 29
The message recovery accuracy of Masked_Encode's MLP in high jitter countermeasures with artificially generated different R.

High_Jitter	AR_0	AR_{20}	AR_{50}	AR_{100}	AR_{200}	AR_{3000}
Accuracy (%)	8.62	2.44	1.75	0.75	1.06	0.25

Table 30
Accuracy results of the message recovery attack after applying the combined countermeasure of low jitter and random delay at Masked_Encode. Due to the low-accuracy and randomness, the results do not follow the expected pattern closely and provide a very limited reference value.

Accuracy (%)	SH_{10}	SH_{20}	SH_{50}
AR_{50}	0.19	0.18	0.31
AR_{100}	0.25	0.31	0.60

amount of noise, the rate of decline slows as the SNR decreases. At $N_{0.1}$, the model accuracy remains at 3.88%, which is the highest observed under this countermeasure.

However, on the dataset using random delay countermeasures, the model shows unexpectedly low accuracy in Table 27. This is related to our preprocessing using clipping. When our clipping intervals are less accurate due to random delays, although we ensure that the highest correlation intervals are within the clipped range, each trace may include segments unfamiliar to the model. If we adopt a more refined alignment strategy, the accuracy could likely be improved. This approach makes the results appear inconsistent with the other classification results. In practical attack scenarios, random delay countermeasures may produce unexpected effects if combined with other countermeasures that prevent alignment.

For the dataset implementing clock jitter countermeasures as shown in Tables 28 and 29, the classification accuracy drops to <1%, which also deviates from the leakage assessment results. Through comparison analysis with the Encode dataset, we believe this discrepancy is related to leakage from the first few bits of the byte. In the encoding operation, the highest correlation coefficient comes from the last bit of the byte, but during the model training process, all 8 bits of the trace need to be profiled. In the leakage evaluation of Masked_Encode, although its correlation for the last bit is significantly higher than that of Encode, the leakage from its initial bits is less pronounced in comparison. Consequently, Masked_Encode exhibits a higher maximum correlation coefficient than Encode, yet its overall attack effectiveness is inferior to that of Encode.

Despite the significant drop in attack accuracy under the random delay and clock jitter countermeasures, we further conduct experiments combining both countermeasures. In Tables 30 and 31, the attack accuracy after using combined countermeasures for protection is shown. We can observe that the results obtained in both tables already do not conform to the inherent laws in the horizontal and vertical directions and the accuracy values in both tables are below 1%. Due to the high degree of jitter, the accuracy results in Table 31 appear more scattered than those in Table 30. Nevertheless, the results in either table provide only a very limited reference due to low-accuracy and randomness. In the future, we would try to improve the model architecture and training strategies against the Masked_Encode process, which may lead to performance gains.

6. Countermeasures analysis

In Sections 4 and 5, we test the leakage and attack difficulty of the Kyber implementation under different countermeasures. In this section, We conduct a comprehensive analysis of the results for each attack point and provide recommendations for suitable countermeasures during implementation for each point.

Table 31
Accuracy results of the message recovery attack after applying the combined countermeasure of high jitter and random delay at Masked_Encode.

Accuracy (%)	SH_{10}	SH_{20}	SH_{50}
AR_{50}	0.25	0.56	0.44
AR_{100}	0.31	0.31	0.19

For the Decode, the leakage assessment indicates that countermeasures such as $N_{0.1}$, SH_{50}, and high/low jitter with AR_{200} can suppress the highest correlation coefficients between the power traces and message can be suppressed, which are 0.08, 0.07 and 0.08, respectively. However, when considering the results of deep learning-based SCAs, $N_{0.1}$ proves to be the most effective countermeasure. While the model attack accuracy remains above 1% under other countermeasures, adding noise to achieve SNR = 0.1 reduces the attack accuracy to 0.37%. Therefore, when protecting the Decode, introducing random noise is an appropriate countermeasure, and the interference effect can be further enhanced through the physical design of the IIoT system.

For the Encode, the leakage assessment results indicate that these countermeasures are less effective than for Decode. The largest reduction in correlation is achieved with the random delay. However, when training the attack model, we can find that this countermeasure alone provides relatively weak protection against deep learning-based SCAs. Therefore a combination of random delay and clock jitter countermeasures should be considered. Based on these results, we estimate that a random delay range greater than 10% of the total, combined with high jitter and the proportion of deformation points accounting for more than 30% of the entire trace, can provide better protection.

As shown in our results, Masked_Decode emerges as the most defensible attack point. Regardless of the employed countermeasures, the level of trace leakage swiftly diminishes, falling beneath the critical threshold. Concurrently, whether countermeasures such as noise or clock jitter are employed, any deformation of the traces substantially reduces the efficacy of the attack model targeting this point. In our experiments, the model's attack accuracy drops below 1% at $N_{0.1}$ and under conditions of high/low jitter with AR_{200}. This degree of reduction is unattainable at other attack points with the same countermeasures. Thus, in the protection of this vulnerability, the use of countermeasures that induce deformation can produce significant effects.

Masked_Encode is a unique attack point. Without preprocessing the dataset, these traces are significantly larger compared to other traces, a feature that provides a natural defense against deep learning-based SCAs. However, it also exhibits an extremely high leakage level in both T-test and correlation analysis. We are attempting to trim and concatenate traces to achieve a simple preprocessing effect. This operation helps boost the validation accuracy of our model from around 30% to 88%. Subsequent testing is conducted without sliding windows, which reduces the model's attack accuracy on datasets employing random delay countermeasures, as the model cannot learn a complete waveform. When combined with other countermeasures to prevent accurate classification and achieve alignment purposes, the application of random delay countermeasures in Masked_Encode should achieve outstanding results. Meanwhile, the clock jitter countermeasure has shown excellent effectiveness in protecting against this attack point. In practice, the combination of clock jitter and random delay countermeasures applied to this attack point provides robust protection against this vulnerability.

Although Kyber still has certain vulnerabilities in the experiment, compared with traditional encryption algorithms, it can provide long-term and strong security to protect critical facilities and sensitive data in IIoT systems. As long as appropriate countermeasures are applied to vulnerabilities, the majority of attacks can be defended against.

7. Conclusion

This paper provides a comprehensive assessment of side-channel leakage and resistance in a Kyber implementation, focusing on four critical attack points within an STM32F405 device featuring an ARM Cortex-M4 chip. By employing a variety of countermeasures, including masking, noise addition, random delays, and clock jitter, we assess the effectiveness of these techniques in mitigating side-channel vulnerabilities. Our use of deep-learning models to attack implementations with different countermeasures has verified the leakage and resistance assessment results, highlighting the varying degrees of susceptibility across different algorithmic intermediate values of Kyber implementations. This research not only underscores the importance of targeted countermeasures but also advances the understanding of side-channel vulnerabilities in post-quantum cryptographic implementations. The insights gained from this study offer valuable guidance for the development of more robust and secure cryptographic systems that can be used in IIoT against deep learning-based side-channel attacks. Future work includes exploring more sophisticated countermeasures and further mounting similar side-channel security assessments on other IIoT implementations of post-quantum cryptography.

CRediT authorship contribution statement

Zitian Huang: Writing – original draft, Validation, Software. **Huanyu Wang:** Writing – review & editing, Project administration, Methodology. **Bijia Cao:** Validation, Software. **Dalin He:** Resources. **Junnian Wang:** Project administration, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] E. Fujisaki, T. Okamoto, Secure integration of asymmetric and symmetric encryption schemes, J. Cryptology 26 (2013) 80–101.
[2] H. Tan, W. Zheng, P. Vijayakumar, Secure and efficient authenticated key management scheme for UAV-assisted infrastructure-less IoVs, IEEE Trans. Intell. Transp. Syst. 24 (6) (2023) 6389–6400, http://dx.doi.org/10.1109/TITS.2023.3252082.
[3] H. Tan, W. Zheng, Y. Guan, R. Lu, A privacy-preserving attribute-based authenticated key management scheme for accountable vehicular communications, IEEE Trans. Veh. Technol. 72 (3) (2023) 3622–3635, http://dx.doi.org/10.1109/TVT.2022.3220410.
[4] S. Chari, C.S. Jutla, J.R. Rao, P. Rohatgi, Towards sound approaches to counteract power-analysis attacks, in: Advances in Cryptology—CRYPTO, Springer, 1999, pp. 398–412.
[5] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, F.-X. Standaert, Shuffling against side-channel attacks: A comprehensive study with cautionary note, in: Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18, Springer, 2012, pp. 740–757.
[6] J.-S. Coron, I. Kizhvatov, An efficient method for random delay generation in embedded software, in: International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2009, pp. 156–170.
[7] E. Cagli, C. Dumas, E. Prouff, Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without pre-processing, in: Cryptographic Hardware and Embedded Systems–CHES 2017, Springer, 2017, pp. 45–68.
[8] P.C. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in: Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16, Springer, 1996, pp. 104–113.
[9] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Advances in Cryptology—CRYPTO, Springer, 1999, pp. 388–397.
[10] D. Agrawal, B. Archambeault, J.R. Rao, P. Rohatgi, The EM side—channel (s), in: Cryptographic Hardware and Embedded Systems-CHES 2002, Springer, 2003, pp. 29–45.
[11] D. Heinz, M.J. Kannwischer, G. Land, T. Pöppelmann, P. Schwabe, A. Sprenkels, First-order masked kyber on ARM Cortex-M4, Cryptol. ePrint Arch. (2022) Paper 2022/058. URL https://eprint.iacr.org/2022/058.
[12] M. Brisfors, M. Moraitis, E. Dubrova, Side-channel attack countermeasures based on clock randomization have a fundamental flaw, Cryptol. ePrint Arch. (2022).
[13] L. Backlund, K. Ngo, J. Gärtner, E. Dubrova, Secret key recovery attack on masked and shuffled implementations of CRYSTALS-Kyber and Saber, in: International Conference on Applied Cryptography and Network Security, Springer, 2023, pp. 159–177.
[14] C. Hoffmann, B. Libert, C. Momin, T. Peters, F.-X. Standaert, Towards leakage-resistant post-quantum CCA-secure public key encryption, IACR Cryptol. ePrint Arch. 2022 (2022) 873.
[15] M. Azouaoui, Y. Kuzovkova, T. Schneider, C. van Vredendaal, Post-quantum authenticated encryption against chosen-ciphertext side-channel attacks, Cryptol. ePrint Arch. (2022).
[16] J.-P. D'Anvers, M. Van Beirendonck, I. Verbauwhede, Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations, IEEE Trans. Comput. 72 (2) (2022) 321–332.
[17] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J.M. Schanck, P. Schwabe, G. Seiler, D. Stehle, CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM, in: 2018 IEEE European Symposium on Security and Privacy, EuroS&P, 2018, pp. 353–367, http://dx.doi.org/10.1109/EuroSP.2018.00032.
[18] E. Alkim, Y.A. Bilgin, M. Cenk, F. Gérard, Cortex-M4 optimizations for {R, M} LWE schemes, IACR Trans. Cryptogr. Hardw. Embed. Syst. (2020) 336–357.
[19] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J.M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber algorithm specifications and supporting documentation, NIST PQC Round 2 (4) (2019) 1–43.
[20] B.J. Gilbert Goodwill, J. Jaffe, P. Rohatgi, et al., A testing methodology for side-channel resistance validation, in: NIST Non-Invasive Attack Testing Workshop, Vol. 7, 2011, pp. 115–136.
[21] B.L. Welch, The generalization of 'STUDENT'S'problem when several different population varlances are involved, Biometrika 34 (1–2) (1947) 28–35.
[22] S. Mangard, Hardware countermeasures against DPA–a statistical analysis of their effectiveness, in: Topics in Cryptology–CT-RSA 2004: The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings, Springer, 2004, pp. 222–235.
[23] A. Duc, S. Faust, F.-X. Standaert, Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version, J. Cryptology 32 (4) (2019) 1263–1297.
[24] E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model, in: Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6, Springer, 2004, pp. 16–29.
[25] C. Mujdei, L. Wouters, A. Karmakar, A. Beckers, J.M. Bermudo Mera, I. Verbauwhede, Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication, ACM Trans. Embed. Comput. Syst. 23 (2) (2024) 1–23.
[26] Q. Pan, J. Wu, A.K. Bashir, J. Li, J. Wu, Side-channel fuzzy analysis-based AI model extraction attack with information-theoretic perspective in intelligent IoT, IEEE Trans. Fuzzy Syst. 30 (11) (2022) 4642–4656, http://dx.doi.org/10.1109/TFUZZ.2022.3172991.
[27] Z. Martinasek, J. Hajny, L. Malina, Optimization of power analysis using neural network, in: Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Springer, 2014, pp. 94–107.
[28] L. Weissbart, Performance analysis of multilayer perceptron in profiling side-channel analysis, in: Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18, Springer, 2020, pp. 198–216.
[29] H. Maghrebi, T. Portigliatti, E. Prouff, Breaking cryptographic implementations using deep learning techniques, in: Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6, Springer, 2016, pp. 3–26.
[30] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[31] B.-Y. Sim, J. Kwon, J. Lee, I.-J. Kim, T.-H. Lee, J. Han, H. Yoon, J. Cho, D.-G. Han, Single-trace attacks on message encoding in lattice-based KEMs, IEEE Access 8 (2020) 183175–183191.
[32] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, N. Homma, Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs, IACR Trans. Cryptogr. Hardw. Embed. Syst. (2022) 296–322.
[33] Z. Xu, O. Pemberton, S.S. Roy, D. Oswald, W. Yao, Z. Zheng, Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber, IEEE Trans. Comput. 71 (9) (2021) 2163–2176.
[34] P. Ravi, S. Bhasin, S.S. Roy, A. Chattopadhyay, On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks, IEEE Trans. Inf. Forensics Secur. 17 (2021) 684–699.

[35] J. Wang, W. Cao, H. Chen, H. Li, Practical side-channel attack on message encoding in masked kyber, in: 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom, IEEE, 2022, pp. 882–889.
[36] E. Dubrova, K. Ngo, J. Gärtner, R. Wang, Breaking a fifth-order masked implementation of CRYSTALS-Kyber by copy-paste, in: Proceedings of the 10th ACM Asia Public-Key Cryptography Workshop, 2023, pp. 10–20.
[37] L. Wu, S. Picek, Remove some noise: On pre-processing of side-channel measurements with autoencoders, IACR Trans. Cryptogr. Hardw. Embed. Syst. (2020) 389–415.
[38] M. Bucci, R. Luzzi, M. Guglielmo, A. Trifiletti, A countermeasure against differential power analysis based on random delay insertion, in: 2005 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2005, pp. 3547–3550.
[39] NewAE Technology Inc., ChipWhisperer Documentation, 2024, URL https://chipwhisperer.readthedocs.io/en/latest/getting-started.html.
[40] M.J. Kannwischer, J. Rijneveld, P. Schwabe, K. Stoffelen, PQM4: Post-quantum crypto library for the ARM cortex-M4, 2019.
[41] D. Amiet, A. Curiger, L. Leuenberger, P. Zbinden, Defeating NewHope with a single trace, in: Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11, Springer, 2020, pp. 189–205.
[42] Z. He, X. Deng, B. Yang, K. Dai, X. Zou, A SCA-resistant processor architecture based on random delay insertion, in: 2015 International Conference on Computing and Communications Technologies, ICCCT, IEEE, 2015, pp. 278–281.
[43] K. Ngo, E. Dubrova, T. Johansson, Breaking masked and shuffled CCA secure Saber KEM by power analysis, in: Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security, 2021, pp. 51–61.
[44] Y. Ji, E. Dubrova, A side-channel attack on a masked hardware implementation of CRYSTALS-Kyber, in: Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security, 2023, pp. 27–37.