Message Recovery Attack of Kyber Based on Information Leakage in Decoding Operation

Mengyao Shi[1,2], Zhu Wang[1,2(✉)], Tingting Peng[1,2], and Fenghua Li[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
wangzhu@iie.ac.cn

[2] University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. In this work, we propose practical side-channel attacks for message recovery in post-quantum key encapsulation mechanisms (KEM). As a target scheme, Kyber is a standardized algorithm in the ongoing NIST standardization process. Notably, this work is the first one that implements message recovery by exploiting the information leaked on computational operations during Kyber decoding. The main findings include 1. analyzing computational operations during decoding by power consumption information to effectively recover message; 2. recovering message by analyzing the time differences existing in decoding single bits; 3. by way of simple power analysis, using incremental storage leakage to recover the message.

Keywords: Lattice-based cryptography · Side-channel attacks · Message decoding · Kyber

1 Introduction

The rapid development of quantum computing technology has posed a severe challenge to the security of traditional modern cryptographic schemes. The unique physical properties of quantum systems, such as superposition and coherence, make quantum Turing machines more computationally efficient than classical Turing machines. In 1994, Peter w. Shor of Bell Labs [1] proposed quantum computing based on the discrete logarithm problem and the significant integer prime factorization problem, which makes it possible to solve a large number of decomposition problems in polynomial time; In 1996, Grover [2] proposed to speed up the key search by continuously doing the Mississippi transformation to increase the likelihood of the desired solution value. These contributions make public-key cryptosystems built on the assumption of computational hardness (e.g., RSA, ECC, and other algorithms) no longer secure, and also raise the issue of the security of protocols based on such algorithms and the security of products based on such protocols. Accordingly, research on post-quantum cryptosystems has become a frontier focus issue in the cryptographic field.

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 630–647, 2023.
https://doi.org/10.1007/978-3-031-25538-0_33

Based on the new cryptosystem, post-quantum algorithms mainly include code-based cryptography, hash-based cryptography, lattice-based cryptography, multivariate cryptography and supersingular isogeny cryptography. At the end of 2016, the National Institute of Standards and Technology (NIST) launched the standard solicitation of post-quantum cryptography algorithm [3]. On July 5, 2022, NIST announced it has completed the third round of the post-quantum Cryptography standardization process. As a result, a total of four candidate algorithms have been selected for standardization. As one of four, Kyber [4] was successfully selected as a standardized algorithm for it strong security and excellent performance [5], and NIST expected it to work well in most applications.

The cryptographic algorithm ensures the algorithm's security through theory, but the mathematical security of algorithm design can not fully guarantee the security of the implementation. Side-channel attack is an effective technology that seriously threatens the safety of cryptographic implementation (cryptographic chip and cryptographic system). Side-channel attacks (refer to Fig. 1) mainly exploits the leakage of side information (such as time, power consumption, etc.) during the operation of the cryptographic algorithm. It achieves the attack by analyzing the dependency between the side-channel information and the secret information. Since timing attack [6] was proposed in 1996, after more than 20 years of development, the international standard algorithms 3DES, AES, Sha-3, RSA, ECC, etc. have been successfully analyzed using side-channel attacks. The mighty power of side-channel attack in analyzing classical cryptographic algorithms seriously threatens the security of cryptographic systems, which poses a severe challenge to the protection of cryptographic algorithms and has attracted extensive attention. Therefore, algorithm security is no longer limited to design safety, and the implementation of security has become an important indicator. Therefore, in the third round of evaluation, NIST pointed out that an important indicator to measure whether the post-quantum cryptography candidate algorithm can become a standard is to resist side-channel attack.

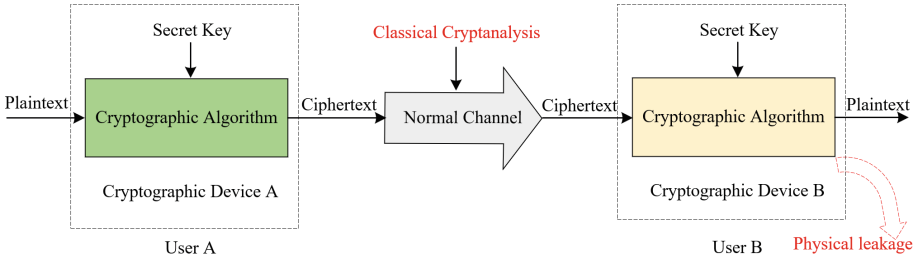

Fig. 1. Pictorial representation of the side-channel attack.

From classical cryptography to post-quantum cryptography, side-channel attack is not a simple transplant. Although assumptions are both based on mathematical difficulties, the post-quantum public key cryptosystem adopts an

entirely different mathematical structure from the traditional public key cryptosystem. The side-channel information leaked in the implementation process and the subsequent side-channel attack methods are further from the conventional one, so the side-channel security can not be obtained by simply transplanting the attack methods. For post-quantum cryptography, the adversary must invest extra effort to find the key-dependent data required for side-channel attack. Therefore, to meet the urgent needs of formulating post-quantum cryptography algorithm standards, it is critical to carry out accurate and efficient side-channel security analysis on post-quantum cryptography.

1.1 Related Work

In the process of encryption and decryption, the cryptographic equipment will inevitably leak signals (power consumption, electromagnetic emanation, time, etc.). Unlike other forms of cryptographic analysis, side-channel attacks use the information leaked from the target device to find information related to the key, including the time or power consumption of the internal operation of the device or the error output generated by the device. Currently, the side-channel attack methods of post-quantum cryptography mainly focus on power consumption/ electromagnetic emanation, time and fault injection. In recent years, side-channel attacks related to Kyber algorithm mainly include: in 2019, Pessl and Primas [7] proposed a single power trace attack scheme for NTT in Kyber; In 2020, Ravi et al. [8] launched a chosen-ciphertext attack against a variety of lattice-based cryptographic algorithms such as Round5, LAC, Kyber, FrodoKEM, NewHope, etc., and its main attack targets are error correcting codes and FO-transformation [9]. In 2019, Ravi and Roy et al. [10] proposed a fault attack scheme against nonce random seeds of Gaussian distribution in NewHope, Kyber, FrodoKEM and other different cryptographic algorithms; In 2018, Albrecht and Deo et al. [11] launched cold boot attacks against NTT in Kyber and NewHope. The above attacks on Kyber and other algorithms mainly focus on core operators such as NTT, error correcting code and FO-transformation. In fact, there are more effective attack points than those mentioned above. Information leakage in the encoding and decoding process can also effectively recover message information.

In the key encapsulation mechanism, the message encoding and message decoding processes involve arithmetic operations on the message. The study of side-channel attacks on these two processes is described below separately.

Message Encoding Process. Amiet *et al.* [12] in PQCrypto 2020 used a single power trace to attack the message encoding operation in the C reference implementation of the NewHope key encapsulation mechanism submitted to NIST for the second round and found that it had a severe side-channel vulnerability. Under the Hamming weight leakage model, the power consumption of the processed sensitive intermediate value under two different values was very different, which leaked information about single bits of the message. When the compiler optimization level was -O0, the bit-by-bit recovery of the message could be realized only by using simple power attack. When the compiler optimization level

reached -O3, 256 templates should be pre-processed. By combining the template attack and the brute-force search, the success rate of message recovery could reach 99%. Based on [12], Sim *et al.* [13] further explored the side-channel security of the message encoding operation in other lattice-based key encapsulation schemes that entered the third round of NIST evaluation. The author used a single power trace to analyze the message encoding phase of the key encapsulation scheme implemented by the C reference. It was found that the side-channel vulnerabilities in [12] commonly exist in Kyber, Saber, and FrodoKEM. In particular, when analyzing Saber and FrodoKEM, the author used a machine learning method to build templates to help recover the encoded message. Xu *et al.* [14] successfully attacked the memory-efficient and high-speed Kyber encoding operation in *pqm4* [15]. By taking specific preprocessing measures to filter POIs and calculate the threshold in the profiling stage, the author could use the two-stage recovery attack on the -O0 and -O3 compiler levels to recover the message with a 100% success rate.

Message Decoding Process. Ngo *et al.* [16] conducted a comprehensive study on the Saber scheme with masking implementation for IND-CCA security. In the reference implementation, the author found the side-channel leakage point of the "incremental storage" vulnerability in the message decoding process mentioned in [17] and the newly found poly_A2A() primitive also contains an exploitable point of "incremental storage" vulnerability. Based on this work, Ngo *et al.* [18] subsequently used the ciphertext malleability proposed by [17] to attack the decoding process of the Saber scheme with shuffling protection at the same time. As a result, the message recovery attack was realized. Ravi *et al.* [19] analyzed the decoding process of multiple lattice-based KEMs and successfully implemented horizontal message recovery attacks.

These attacks show some vulnerabilities in the lattice-based post-quantum cryptography schemes and strongly indicate that more potential vulnerabilities have not been discovered. Therefore, we believe that further research in this field is necessary to give the implementation scheme a complete security check and provide security recommendations for deploying these schemes in the real world.

1.2 Our Contribution

We perform side-channel attacks on the implementation of Kyber obtained from the *pqm4* public library, a testing and benchmarking framework for post-quantum cryptographic schemes on the ARM Cortex-M4 microcontroller. Refer to Fig. 2 for the pictorial description of our attacks (Attack_Decoding) targeting the message recovery. In this work, by focusing on the message decoding operation in the decapsulation phase, we comprehensively analyze the side-channel leakage about time and power consumption that can be used for message recovery in this process. The main contributions of this paper can be summarized as follows.

(1) **Novelty of attack target:** Existing attacks focus more on recovery attacks against the long-term key, while message recovery attacks leading to shared

session key recovery have received little attention. Compared with traditional PKE/KEM based on RSA and ECC, message in LWE/LWR-based PKE/KEM operates uniquely on each bit of message.

(2) **Universality of the attack:** In essence, our work exploits the algorithmic properties inherent in the LWE/LWR-based scheme since the decoding operation is unique to the LWE/LWR scheme. In this paper, we achieve full recovery of the message from the assembly level by analyzing the power leakage and the time leakage generated during the bitwise computation and storage of the message.

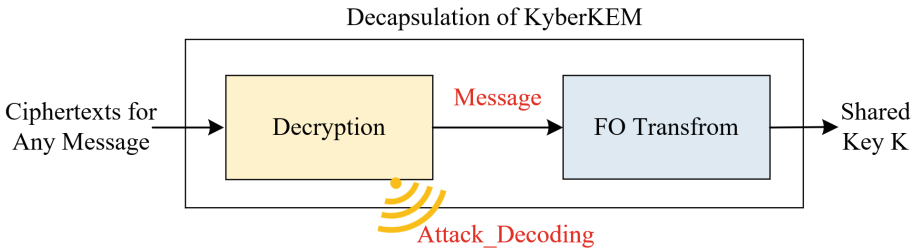

Fig. 2. Pictorial representation of our proposed attacks on the Kyber decapsulation procedure.

1.3 Outline

The rest of the paper is organized as follows. In the next section, we recall the Kyber algorithm and message recovery attacks. Section 3 consists of the detailed experimental environment. Section 4 shows some findings of our side-channel attacks on Kyber's message decoding process. Section 5 makes a complete and profound analysis of our experimental findings in Sect. 4. Finally, the conclusion is discussed in Sect. 6.

2 Background

2.1 Parameter Settings

The basic elements in Kyber are the polynomials in the ring $\mathbb{Z}_q[x]/(x^n + 1)$, denoted by $\mathcal{R}_q$, with $n = 256$ and $q = 3329$ in all variants of Kyber. The parameter k represents the dimension of the matrix of polynomials in $\mathcal{R}_q$. Kyber has three variants aimed at different security levels. In order of increasing security they are Kyber512, Kyber768, and Kyber1024, and their parameters can be found in Table 1. The attack methods proposed in this paper are applicable to the above three security levels.

Table 1. Parameter sets for Kyber.

Algorithm	NIST-level	n	q	k
Kyber512	1(AES-128)	256	3329	2
Kyber768	3(AES-192)	256	3329	3
Kyber1024	5(AES-256)	256	3329	4

2.2 Module Learning with Errors Problem

The learning with errors (LWE) problem proposed by Regev [20] is one of the most well-known hard problems in the average case. It is considered computationally infeasible for both classical and quantum computers. There are two versions of the LWE problem - Search LWE and Decisional LWE. The search variant of the LWE problem requires the attacker to compute the secret value given several LWE samples. The decisional variant of the LWE problem requires the attacker to distinguish uniformly random samples from similar LWE samples.

As we know, several lattice-based NIST candidates are constructed based on LWE or algebraically structured variants of the standard LWE problem known as Ring/Module-LWE (RLWE/MLWE). The security of Kyber is based on the Module-LWE hardness assumption. MLWE differs from LWE in using the polynomial ring instead of the integer ring. The decision and search LWE problem over standard lattices can be extended to the decision and search MLWE problem over module lattices.

2.3 Kyber

Lattice-based IND-CPA schemes can be made secure against CCA by being transformed into IND-CCA schemes with the help of a post-quantum variant of the Fujisaki-Okamoto (FO) transformation [8]. The transformation is also used by Kyber to achieve IND-CCA security. Kyber KEM contains three algorithms: Key Generation, Encapsulation, and Decapsulation. Simplified versions of the three algorithms are described in the corresponding Algorithm 1, Algorithm 2, and Algorithm 3.

Algorithm 1. KYBER.CCAKEM.KEYGEN()

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8+96}$
1: $Z \leftarrow \mathcal{B}^{32}$
2: $(pk, sk') :=$ KYBER.CPAPKE.KEYGEN()
3: $sk := (sk'||pk||\mathrm{H(pk)}||\mathrm{z})$
4: **return** (pk, sk)

Algorithm 2. KYBER.CCAKEM.ENC(pk)

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Output: Shared key $K \in \mathcal{B}^*$
1: $m \leftarrow \mathcal{B}^{32}$
2: $m \leftarrow$ H(m)
3: $(\overline{K}, r) :=$ G(m||H(pk))
4: $c :=$ KYBER.CPAPKE.ENC(pk, m, r)
5: $K :=$ KDF($\overline{K}$||H(c))
6: **return** (c, K)

Algorithm 3. KYBER.CCAKEM.DEC(c, sk)

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8+d_v \cdot n/8}$
Input: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8+96}$
Output: Shared key $K \in \mathcal{B}^*$
1: $pk := sk + 12 \cdot k \cdot n/8$
2: $h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$
3: $z := sk + 24 \cdot k \cdot n/8 + 64$
4: $m' :=$ KYBER.CPAPKE.DEC $(s, (\mathbf{u}, \mathrm{v}))$
5: $(\overline{K}', r') :=$ G(m′||h)
6: $c' :=$KYBER.CPAPKE.ENC (pk, m', r')
7: **if** $c = c'$ **then**
8: **return** $K :=$ KDF($\overline{K}'$||H(c))
9: **else**
10: **return** $K :=$ KDF(z||H(c))
11: **end if**
12: **return** K

2.4 Message Recovery Attack

By analyzing the implementation of Kyber's encapsulation and decapsulation algorithm in the last part, it can be found that if an attacker can try to obtain the message m, he can easily calculate the shared session key through m and known public values, which will seriously threaten the session security of both communication parties. In other words, the leakage of message values will seriously damage the confidentiality of the cryptosystem.

In the part of related work, we summarize the previous message recovery attacks completed in message encoding and decoding process through side-channel attacks. Theoretically, any operation related to the message can be used as a potential attack path to recover useful information about the message. Regarding the implementation process of Kyber, there are attack paths other than message encoding and decoding. In Kyber's encapsulation procedure Algorithm 2, step 1 involves the generation of random message m; step 2 is the hash operation H involving message m; In step 3, as part of the input, m participates in the hash operation G; The message encoding process in step 4 is an effective and commonly used attack path. In Kyber's decapsulation

procedure Algorithm 3, the first desirable attack path is the message encoding process included in step 4; step 5 here corresponds to step 3 of Algorithm 2, which is also desirable; Since the Kyber KEM to which the FO transformation is applied includes the re-encryption operation, there is step 6 similar to step 3 of Algorithm 2.

Considering the realizability of message recovery under above attack paths, this paper mainly focuses on the message decoding operation in the decapsulation procedure. In the following, we will prove that several side-channel vulnerabilities in the message decoding process can be used to recover the complete message.

```
1  void poly_tomsg(unsigned char msg[32],poly *a){
2      unsigned int k;
3      unsigned short t;
4      int i,j;
5      for(i=0;i<32;i++) {
6          msg[i] = 0;
7          for(j=0;j<8;j++) {
8              k = 8*i+j;
9              t = ((a)->coeffs[k] << 1 ) + 1664;
10             /*calculate message bit*/
11             t = ( t / 3329 ) & 1;
12             /*bit update in memory*/
13             msg[i] |= t << j;
14         }
15     }
16 }
```

Listing 1. C code snippet of message decoding operation in Kyber KEM.

3 Experimental Setup

The device under attack we use is an OSR407 development board with a 32bit ARM Cortex-M4 processor. The target message decoding implementation is from the public *pqm4* library, which provides NIST recommended optimized target for embedded software implementations. We use the original implementation it provides (refer to Listing 1 plus a trigger signal to simplify the recording of power traces. The device used to record power traces is the Lecroy 3000z oscilloscope with a sampling rate of 1.0 GSam/sec. Refer to Fig. 3 for our power-based SCA setup used for our experiments.

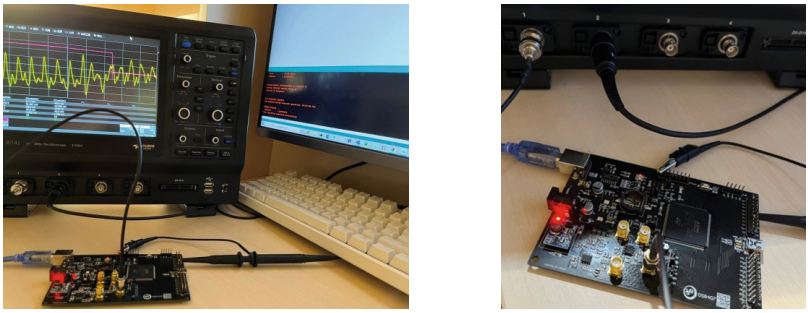

Fig. 3. Experimental setup for SCA.(a) SCA setup. (b) Zoomed-in view of the DUT.

4 SPA of ARM-Specific Implementation

In this section, we attacked the "ARM-Specific" reference implementation specifically optimized for ARM Cortex-M4 processor of Kyber from a side-channel perspective. Accordingly, we found three obvious side-channel leakages on message decoding operation during the decapsulation procedure of Kyber.

4.1 Power Consumption Leakage in Computation

Collecting power traces during Kyber decapsulation, it is found that there are significant differences in the power trace when decoding different message bits. Figure 4 interprets the power trace with the first message byte value of 2. In

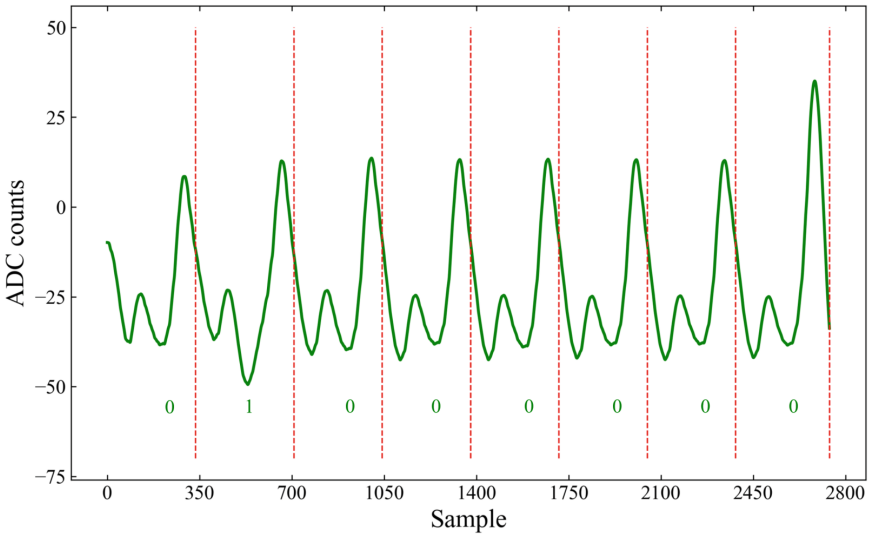

Fig. 4. A single trace measurement of $msg[0] = 2$ (binary 0000 0010).

the process of decoding this byte, the recovered bits are 0,1,0,0,0,0,0,0 orderly; that is, for the calculation results, only the second bit is different. The difference can be clearly seen in Fig. 4, because the power consumption fluctuation is more apparent when the recovered message bit value is 1. Therefore, each bit of a message byte can be directly recovered through simple power attack.

4.2 Timing Leakage

In order to better analyze the time difference in the process of decoding the message bits taking different values, we selected 0 and 255 as experimental objects. When the decoding process of message byte values 255 and 0 is repeated 32 times, we found that the former takes longer than the latter. Additionally, Fig. 5 shows the power traces of the first message byte value of 0 and 255, respectively. It can be clearly seen that when the start position of the two is the same, the sampling end time after decoding is different. Therefore, it can be inferred that processing bit 1 takes much longer than processing bit 0.

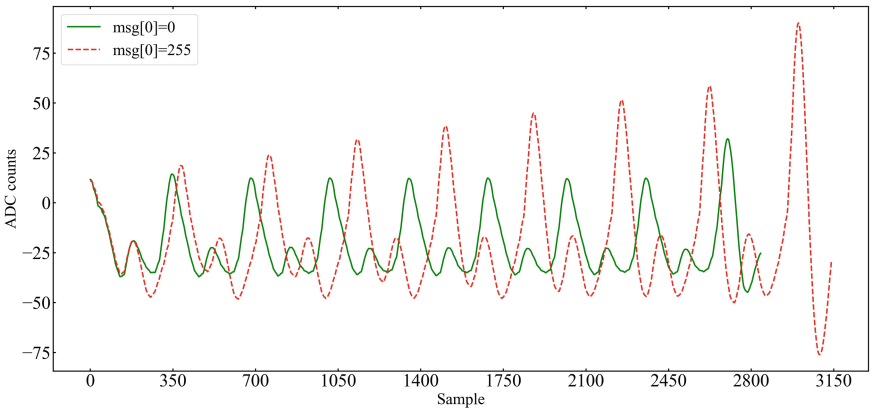

Fig. 5. Traces of $msg[0] = 0$ and $msg[0] = 255$.

4.3 Incremental Leakage

By comparing the power traces of message bytes with values of 0,1,2... 255 during message decoding, it is found that there is a fixed peak interval in the interval processing of adjacent bits, and the maximum power consumption in this interval is always greater than or approximately equal to the power consumption of the corresponding interval of the last bit. Figure 6 indicates the comparison details in the power traces where $msg[0]$ takes 0,1,2,34, and 255, respectively.

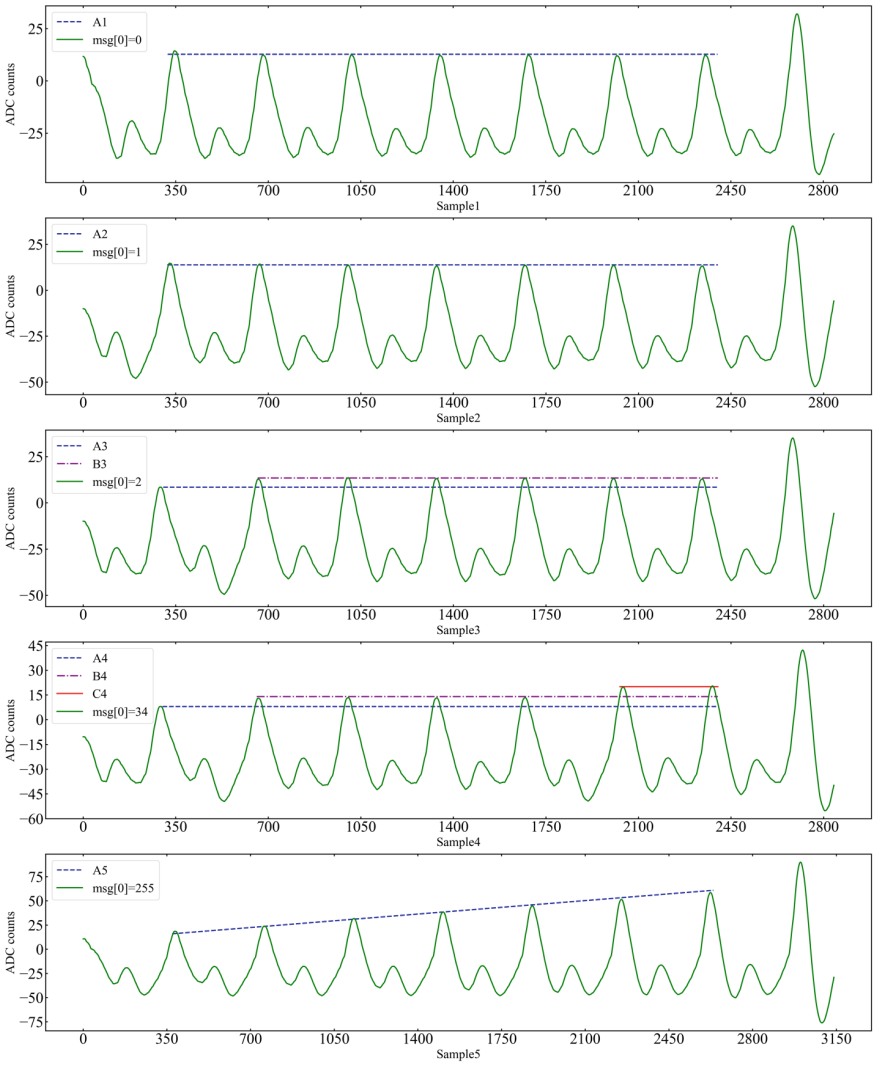

Fig. 6. Traces of $msg[0] = 0, 1, 2, 34, 255$ (top to bottom).

5 Analysis of Experimental Results

To better understand the execution details of the message decoding process on the actual embedded device, we use the arm-none-eabi-gcc compiler toolchain to further obtain the assembly code under -Os (refer to Listing 2) and -O0 compiler options. The analysis of these instructions can help us to complete effective message recovery attacks under the message decoding path.

```
.L3:
 strb r6, [r0, #1]!
 add r5, r1, r2, lsl #1
 movs r4, #0
.L2:
/* t = ((a)->coeffs[k] << 1 ) + 1664; */
 ldrsh r3, [r5], #2
 ldrsh ip, [r0]
 lsls r3, r3, #1
 add r3, r3, #1664
 uxth r3, r3
/* t = ( t / 3329 ) & 1; */
 udiv r3, r3, 3329
 and r3, r3, #1
/* msg[i] |= t << j; */
 lsls r3, r3, r4
 adds r4, r4, #1
 orr r3, r3, ip
 cmp r4, #8
/* store msg[i] in memory*/
 strb r3, [r0]
 bne .L2
 adds r2, r2, #8
 cmp r2, #256
 bne .L3
```

Listing 2. Assembly code snippet (at -Os) of a single iteration of the message decoding function in Kyber.

5.1 Power Consumption Analysis

Power Consumption Analysis at -Os Level. By comparing the power consumption during decoding, two significant differences in power trace segments were found when the resultant bits were 0 and 1, respectively. They are now labeled separately in Fig. 7. According to the execution of instructions, the two apparent differences are caused by the ***udiv*** instruction (integer division instruction) in one-bit processing and the ***strb*** instruction (storage instruction from register to memory) at the end of one-bit processing. The following analyzes the two differences, respectively.

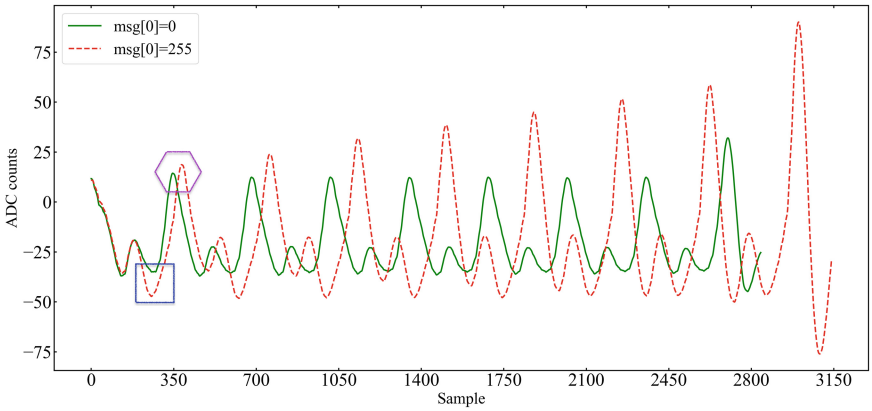

Fig. 7. Traces at -Os of $msg[0] = 0$ and $msg[0] = 255$ (two labels showing differences between message bit 0 and 1)

udiv Instruction. By analyzing the sampling interval of ***udiv*** instruction execution on power traces, it is found that the power fluctuation amplitude when the calculation result is 0 is much smaller than that when the calculation result is 1. This significant difference is marked in Fig. 7 with a blue rectangular box. Further observation shows that this difference exists in the eight times of decoding with $msg[0] = 0$ and 255, which is consistent with the above analysis. To prove the universality of the above analysis, we repeat the above experiment for 256 possible values of one byte, which is verified to be true. Since the difference is so significant that the calculation results can be read directly from the oscilloscope, the value of the message can be recovered from a single power trace through SPA. Figure 8 shows a single power trace. According to the above

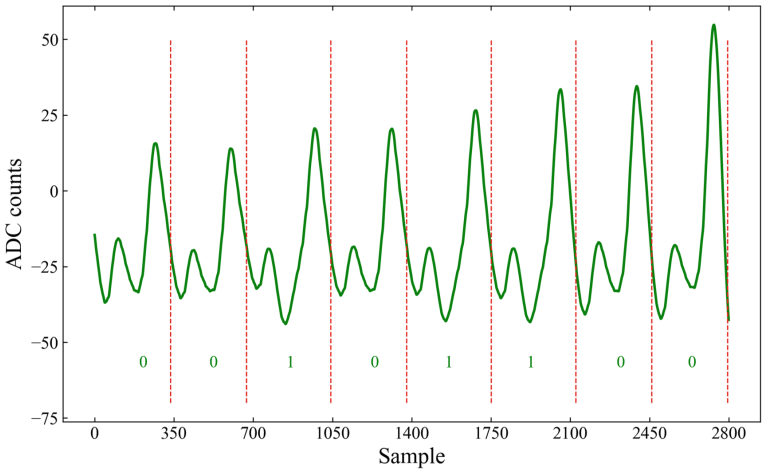

Fig. 8. A single trace measurement where $msg[0] = 52$.

analysis, the decoded message byte can be directly recovered bit by bit horizontally: 52 (binary representation: 00110100).

strb Instruction. It is well known that the storage operation divulges the Hamming weight of the stored value. Therefore, if an attacker can identify the Hamming weight of the median value of each message byte during the decoding operation, he can recover each bit value of the message byte under the leakage feature of incremental storage. According to the analysis of Fig. 6, for $msg[0] = 0$, at the peak position of 1-bit processing, the values of the first seven processes are approximately the same because the Hamming weight of the stored operand is 0 every time, so the power consumption of the corresponding instruction execution is roughly the same; For $msg[0] = 255$, at the peak position of 1-bit processing, the value of the first seven processing times is higher than that of the previous time. This is because each decoded message bit is 1. When it is connected to the median value of the message byte, the Hamming weight of the value increases by 1. Therefore, the power consumption of ***strb*** instruction execution rises with the increase of the Hamming weight of the operand. For the last bit, the decoding result of $msg[0] = 0$ is 0, and the decoding result of $msg[0] = 255$ is 1. At this time, the value of the spike position of 1-bit processing is always greater than the previous one. It can be seen from the assembly code snippet of Listing 2 that the power consumption sampling in this interval will include the subsequent ***add*** and ***cmp*** instructions, and thus the power fluctuation is more significant.

In general, the incremental storage leakage during the decoding operation can be used to recover the message byte except the first bit and the last bit. To prove the universality of the above analysis, we repeat the above experiment for 256 possible values of one byte, which is verified to be true. Since the leakage is so significant that the decoding result can be read directly from the oscilloscope, six bits of one message byte can be recovered from a single power trace through simple power analysis.

The Horizontal Message Recovery Scheme. Using the leakage on the ***udiv*** division instruction, we can recover the value of each message byte bit by bit on a single power trace. In order to make full use of the leaked information, a fault-tolerance scheme can be built with the leaked information on the ***strb*** instruction to assist in verifying that the message bits recovered through the ***udiv*** instruction leakage are correct, except the first bit and the last bit of one byte. For 256 possible values of one message byte, we demonstrate that the above horizontal message recovery scheme can recover the value of message bytes with a 100% success rate.

Power Consumption Analysis at -O0 Level. At the -O0 level, the division operation in the decoding process does not use the ***udiv*** instruction but the ***umull*** series of instructions. Through experimental analysis, it is found that the difference in power consumption is not apparent when the operand takes different values. Therefore, the message value cannot be recovered using the

scheme proposed in the last part. The power consumption leakage found on the ***udiv*** instruction no longer applies to the analysis at -O0 compiler level.

To explore the applicability of the horizontal message recovery attack proposed in the last part at this compiler level, the compiler parameter is modified to -O0, and the experiment under -Os is repeated. To reduce the negative effect of noise, the decoding experimental data is sampled 1000 times and then averaged. Figure 9 shows two power traces during the message decoding process where $msg[0]$ is 0 and 255, respectively.

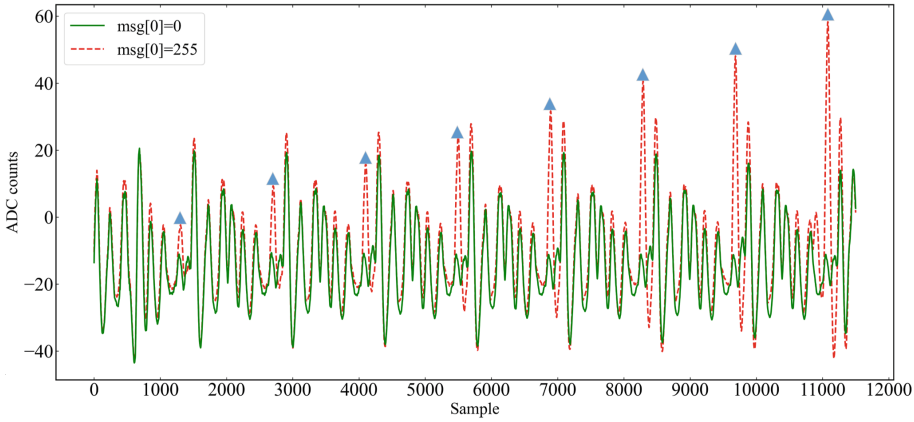

Fig. 9. Traces of $msg[0] = 0$ and $msg[0] = 255$.

For the decoding process with $msg[0] = 0$, the operand of ***strb*** instruction stored each time is 0, and the corresponding Hamming weight is also 0. Therefore, with the Hamming weight model, the power consumption of ***strb*** instruction execution is approximately equal in the eight intervals. For the decoding process with $msg[0] = 255$, the operands of ***strb*** instruction stored each time are 1,3,7,15,31,63,127,255, and the corresponding Hamming weight increases by one from 1 to 8. Therefore, under the Hamming weight model, the power consumption during the execution of ***strb*** instruction is gradually augmented during 8 intervals. Consequently, the average difference in ***strb*** instruction power consumption per bit of decoding processing should be increased. As the result, the interval in which the difference between the two values shown in Fig. 9 increases significantly in turn in 8 processes can represent the sampling interval of ***strb*** instruction execution. We express this interval as $w[i]$, where $i \in [0, 7]$. When a simple power analysis is made for the $w[i]$ interval of a single group of data, the power consumption values of w[i] in the eight processing processes are approximately the same in terms of $msg[0] = 0$; As far as $msg[0] = 255$ is concerned, the power consumption of $w[i]$ in the eight processes is significantly higher than that in the last process. This is consistent with the power consumption characteristics of Hamming weight model.

The Horizontal Message Recovery Scheme. For the attacked message decoding process, when recovering the message of one byte on the obtained power trace, first locate the $w[i]$ interval corresponding to the byte, where $i \in [0, 7]$. According to the analysis in the last section, for $i \in [1, 7]$, compare the power consumption value of $w[i]$ with the power consumption value of $w[i-1]$ corresponding to the previous process, and then recover the message value of the ith bit by bit; For the recovery of the message value of the 0th bit, firstly, build two templates with the corresponding bit value of $w[0]$ being 0 and 1 respectively, then calculate the similarity between $w[0]$ and the two templates respectively. Finally, take the value corresponding to the template with high similarity as the message bit recovery value of $w[0]$. As a result of the attack, the value of the message byte can be completely recovered.

5.2 Timing Leakage Analysis

Usually, the main reason for the leakage of time information is the use of branch statements. Kyber adopts nested for-loop in the decoding process, and its code implementation effectively avoids the timing attack based on branch statements. However, by analyzing the assembly code in the decoding process, we find that the number of clock cycles spent by ***udiv*** instructions depends on the value of the operand, while ***strb***, ***add***, ***movs***, ***ldrsh***, ***lsls***, ***uxth***, ***orr***, ***cmp*** instructions are completed in fixed number of clock cycles. It can be seen that the time difference of different message intermediate byte values calculated depends on the time difference of ***udiv*** instruction execution.

For 256 different values of one message byte, we build 256 groups of ciphertext. Except that the values of the first byte are belonging to [0,255], the message values of other bytes after decoding are random. To obtain a clear and noise-weakened power trace, we repeat the decoding process 100 times and average the obtained power traces.

It is known that the number of sampling points = sampling rate × sampling time. From the power traces in Fig. 5, we can see that for a 1-byte message decoding process, the sampling interval of $msg[0] = 0$ is narrower than that of $msg[0] = 255$. Therefore, when the sampling rates are fixed, the execution time of decoding $msg[0] = 255$ is longer than that of $msg[0] = 0$. According to the clock cycle required for instruction execution, this difference is caused by the difference of the operand when the ***udiv*** instruction is executed. In the process of decoding $msg[0] = 0$, the result of 8 times of ***udiv*** instruction execution is 0; In the process of decoding $msg[0] = 255$, the result of 8 times of ***udiv*** instruction execution is 1. According to the statistics of the sampling points in the eight effective intervals of Fig. 5, there are significant differences in the sampling points processed in these eight segments: for $msg[0] = 0$, there are about 335 sampling points in an effective interval; For $msg[0] = 1$, the number of sampling points in an effective interval is about 373; As explained above, there is a significant difference in clock cycles spent by the ***udiv*** instruction in implementing the above two calculations.

Unlike -Os, which uses the ***udiv*** instruction to implement the division operation t/Q, the other four compiler levels -O0, -O1, -O2, and -O3 use a combination of multiplication ***umull***, subtraction ***subs***, and addition ***add*** instructions to implement the division operation. In particular, in the Kyber's specification, the author mentioned that on the ARM Cortex-M3 processor, the ***umull*** instruction is implemented within a variable number of clock cycles, and its execution time depends on the value of the operands involved in the operation, which is a possible source of timing leakage. However, as far as the ARM Cortex-M4 processor is concerned, this leakage does not exist because the ***umull*** instruction is guaranteed to complete within a single clock cycle. According to the number of clock cycles of instructions execution, the analysis of the remaining assembly instructions shows that when these four compiler optimizations are adopted, the message decoding process does not have side-channel leakage in the time dimension on the Cortex-M4 processor.

6 Conclusion

In this paper, we prove that Kyber's decoding process is vulnerable to side-channel attacks in terms of power consumption and time. Subsequently, We show a new type of horizontal message recovery attack. The proposed attack method can use a single power trace to recover the message value with a 100% success rate. Because of the severe threat to the security system, adding well-designed shuffling and masking schemes to the message decoding process may be an effective protection strategy.

References

1. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)
2. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219 (1996)
3. Chen, L., et al.: Report on post-quantum cryptography, vol. 12. US Department of Commerce, National Institute of Standards and Technology (2016)
4. Bos, J., et al.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 353–367. IEEE (2018)
5. PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4
6. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
7. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 130–149. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_7

8. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMS. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 307–335 (2020)
9. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
10. Ravi, P., Roy, D.B., Bhasin, S., Chattopadhyay, A., Mukhopadhyay, D.: Number "Not Used" once - practical fault attack on *pqm4* implementations of NIST candidates. In: Polian, I., Stöttinger, M. (eds.) COSADE 2019. LNCS, vol. 11421, pp. 232–250. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16350-1_13
11. Albrecht, M.R., Deo, A., Paterson, K.G.: Cold boot attacks on ring and module LWE keys under the NTT. Cryptology ePrint Archive (2018)
12. Amiet, D., Curiger, A., Leuenberger, L., Zbinden, P.: Defeating NewHope with a single trace. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 189–205. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_11
13. Sim, B.Y., et al.: Single-trace attacks on message encoding in lattice-based KEMs. IEEE Access **8**, 183175–183191 (2020)
14. Xu, Z., Pemberton, O.M., Roy, S.S., Oswald, D., Yao, W., Zheng, Z.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: the case study of Kyber. IEEE Trans. Comput. **71**, 2163–2176 (2021)
15. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4
16. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked IND-CCA secure saber KEM implementation. IACR Trans. Cryptographic Hardware Embedded Syst., 676–707 (2021)
17. Ravi, P., Bhasin, S., Roy, S.S., Chattopadhyay, A.: On exploiting message leakage in (few) NIST PQC candidates for practical message recovery and key recovery attacks. Cryptology ePrint Archive (2020)
18. Ngo, K., Dubrova, E., Johansson, T.: Breaking masked and shuffled CCA secure saber KEM by power analysis. In: Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security, pp. 51–61 (2021)
19. Ravi, P., Bhasin, S., Roy, S.S., Chattopadhyay, A.: On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks. IEEE Trans. Inf. Forensics Secur. **17**, 684699 (2021)
20. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM (JACM) **56**(6), 34 (2009)