Cache attack on MISTY1

Haopeng Fan[1*], Wenhao Wang[2], Yongjuan Wang[3]

[1] Information Engineering University, Zhengzhou, 450001, People's Republic of China

[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, People's Republic of China

[*]haopengfan@outlook.com

Abstract: Side-channel attacks exploit information from the physical implementation of cryptographic systems, rather than from theoretical weaknesses. In recent years, cache attacks have made significant progress in their ability to recover secret information by combining observations of the victim's cache access and knowledge of the internal structure of the cipher. So far, cache attacks have been implemented for most Feistel-structured and SPN-structured block cipher algorithms, but the security of algorithms for special structures has little attention.

In this paper, the Flush+Reload attack is performed on the implementation of MISTY1. Unlike Feistel and SPN structures, MISTY1 is a class of the block cipher with a recursive structure. The FL function is performed before the plaintext input S-box and after the ciphertext output S-box, making it difficult to attack the first and last rounds. However, we find that the key scheduling part of MISTY1 leaks many bits of key, which, together with the leakage of partial bits of the round key during encryption, was sufficient to recover the key of the MISTY1 algorithm.

We design the algorithm that only needs to observe one time encryption to recover the MISTY1 128-bit key and use leakage during encryption to reduce the complexity of the algorithm. We experiment on 32-byte cache line and 64-byte cache line environment, respectively. In the 32-byte cache line environment, an adversary only needs to observe five times encryption to recover the all 128-bit key of the MISTY1 in 0.035 seconds; in the 64-byte cache line environment, an adversary needs to observe 10 times encryption to recover the entire 128-bit key in 2.1 hours.

Keywords: Side Channel, Cache attack, Flush+Reload, MISTY1, Key Scheduling Part

1 Introduction

The theoretical security of cryptographic systems does not ensure implementation security. Side channel attacks are a major type of implementation level attack on a cryptographic system. They exploit the leakage of electromagnetic radiation, power consumption, the runtime, or even light, sound and heat leakage during encryption. In recent years, cache attacks based on cache access mechanisms of microprocessors have become an active area of research. Cache attacks exploit the basic principle that cache access is two orders of magnitude faster than memory access. This is a security threat to cryptographic systems because cache access relies on the input of the cryptographic algorithm, i.e., the plaintext and the key. Therefore, the analysis at the execution time of certain operations leaks partial bits of the key.

In the block cipher, the S-box is the only nonlinear structure whose security determines the security of the entire cryptographic algorithm. The nonlinear nature of the S-box dictates that it usually uses look-up tables to deploy. Unfortunately, if the S-box stores in multiple cache lines, the adversary can directly obtain partial bits of the S-box input by observing which cache line is accessed, and then derive the entire key based on the leakage generated by multiple plaintexts encryption.

Related Work: Kocher [1] and Kelsey et al [2] took the lead in mention that cache behavior may pose a security threat. Tsunoo et al [3] gave the first practical results of a time-driven caching attack on the Data Encryption Standard (DES). Various cache attacks against AES were given in [4]-[7], some of which require the ability to detect the first or last round of AES. In addition to this, the security of block cipher algorithms such as SM4 [8], ARIA [9], Camellia [10], and Pilsung [11] had been discussed under cache attacks.

Unlike the above block cipher algorithms, MISTY1 [12] is not a Feistel or SPN structure cipher algorithm, it has a unique *Recursive structure*. MISTY1 run FL function before the first round of plaintext input to the S-box and after the last round of ciphertext output from the S-box, which makes the adversary difficult to attack the first round and the last round of MISTY1, so previous methods in block ciphers can not apply to MISTY1 directly. Tsunoo et al. have discussed cache attacks on MISTY1 in a 32-byte cache line environment[13]. They used the average method, which requires 2^{16} plaintexts to recover the entire key in a practical setting. In fact, the work of [13] did not fully investigate the leakage generated by key scheduling and encryption of MISTY1, and it would be difficult for the adversary to observe such a large number of encryption in a real-world environment.

So what leakage exist in MISTY1, and how to do a cache attack on MISTY1 in a real-world environment?

Leakage of key scheduling part: MISTY1 divides the 128-bit key into eight subkeys. We find that the subkey will be calculated directly through the S-box in the key scheduling part of MISTY1, which leaks some bits of subkeys and the relationship between adjacent subkeys by cache attack on S-box.

Leakage of encryption: Due to the complex structure of MISTY1, we do not need to ensure what exactly the input and output of each round is, but can reduce the round key space by the elimination method. The adversary can observe multiple encryptions to confirm partial bits of the round key.

Attack overview: The attack is divided into two phases: online observation and offline analysis. In the online observation phase, the adversary uses the Flush+Reload attack to collect the leakage generated by key scheduling and encryption of MISTY1. In the offline analysis phase, the adversary gets some bits of the subkey directly based on the leakage of the key scheduling part; then, the adversary needs to set the judgment conditions to reduce the complexity of the attack based on the relationship between adjacent subkeys and the leakage of the round key during encryption. In summary, we only need to observe a very small number of encryption to recover the 128-bit key of the MISTY1.

Attack results: We experiment in 32-byte cache line and 64-byte cache line environments. In the 32-byte cache line environment, the adversary only needs to observe the encryption of one plaintext to recover all the key, and the complexity of the attack algorithm is O(2^7), and the runtime is 0.035 seconds on a personal laptop. In the 64-byte cache line environment, the adversary needs to observe the encryption of 10 plaintexts to recover all the keys, the complexity of the attack is O(2^{38}), and the running time on a personal laptop is 7661.06 seconds(about 2.1 hours).

Our Contributions:

- We show how the key scheduling part and each round of encryption leaks information about the master key and the round key.
- We implement the first cache attack on the block cipher algorithm MISTY1 with the recursive structure. The attack is divided into two phases, online observation and offline analysis.
- We propose algorithms on how to recover the MISTY1 master key using observation of one plaintext in the 32-byte cache line and 64-byte cache line environments.
- We reduce the complexity of the attack algorithm by exploiting the leakage during multiple encryption and perform a practical attack in the 32-byte cache line and 64-byte cache line environments.

Document Outline:

In Section 2 we briefly introduce the theory of cache attacks and the details of the block cipher MISTY1. In Section 3 we describe how to obtain the leakage generated by key scheduling and encryption of MISTY1. In Section 4 we illustrate how to attack MISTY1 on 32-byte cache lines environment. In Section 5 we show how to attack MISTY1 on 64-byte cache lines environment. In Section 6 we give the experimental results, and we summarize the whole paper in Section 7.

2 Background

2.1 Cache attack

The CPU cache is a very fast memory, placed between the main memory and the CPU [14]. It usually ranges in size from some hundred kilobytes to a few megabytes. According to the principle of locality, i.e., the storage units accessed by the CPU tend to be clustered in a small continuous region, CPU cache can effectively avoid high latency. Each cache is stored in cache lines. For now, the most common cache line size is 64 bytes, with a small portion being 32 bytes.

When the CPU attempts to access data, it first looks in the cache, and then there are two cases of cache hit and cache miss.

Cache hit: The CPU finds the requested data in the cache, and the requested data can be supplied to the CPU core with almost no latency.

Cache miss: The CPU does not find the requested data in the cache and must first fetch the data via the front side bus and copy it to the cache, resulting in a latency roughly two orders of magnitude higher than that of a cache hit.

The speed difference between a cache hit and a cache miss may reveal information about the contents of the cache. Moreover, because the contents of the cache depend on previous computations, recovering information about the contents of the cache can disclose secret information about previous computations. Currently, cache attacks are used in breaking symmetric ciphers[3-11], public key ciphers[15-17], digital signature algorithms[18-22], and zero-knowledge proofs[23].

2.2 Flush+Reload

Flush+Reload[24] can evict memory blocks from all levels of cache, which depends on the shared memory pages between the spy process and the crypto process. In Intel processors, user threads can use the clflush instruction to flush readable and executable pages, which make the adversary can use the clflush instruction to frequently refresh the target memory block and measure the time to reload that memory block. If, during execution, the victim accesses the memory block, the block will be cached and the adversary's access will be fast. However, if the victim does not access the block, the adversary will reload the block from memory and the access will take longer. Therefore, the adversary can know if the victim accessed the memory block during execution. The Flush+Reload attack is divided into three steps:

1) The target memory block is flushed from all levels of cache.
2) The spy process waits for the crypto process to access the memory block.
3) The spy process reloads the memory block and measures the load time.

2.3 Flush+Reload on S-box

S-box(Substitution-box) is the basic structure of the block cipher to execute substitution calculation. The S-box is the only nonlinear structure in the block cipher, so the goodness of its S-box metrics directly determines the goodness of the entire block cipher.

When the storage size of the S-box is more than the size of the cache line, it may leak some information about the key. Take S_a as an example, S_a is an S-box with a bits input and output. S_a has 2^a entries of size 2^b bytes, so the table of S_a requires 2^{a+b} bytes of memory. Suppose the size of the cache line is 2^k bytes, then S_a will be stored in 2^{a+b-k} cache lines($a+b>k$). The entries of S_a is stored sequentially in cache lines and the entries are stored in the same cache line only if they have the same first $a+b$-k bits, so if the adversary knows where S_a is stored and can use Flush+Reload to know which cache line was accessed during the S-box operation, then the adversary can recover the first $a+b\text{-}k$ bits of the input of S_a. Since the set of inputs is known, the adversary can also get the set of outputs, which has $2^{k-b}(2^a/2^{a+b-k})$ elements.

2.4 MISTY1

MISTY1 is a block cipher algorithm designed by Japanese scholar M. Matsui[12]. MISTY1 was recommended as a transitional block cipher by NESSIE(New European Schemes for Signatures, Integrity and Encryption), and was also listed as one of the block cipher standards by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). Moreover, the MISTY1 algorithm is one of the selected block ciphers by the Japanese CRYPTERC project in May 2003.

MISTY1 has a 128-bit key, a 64-bit block and a variable number of rounds, which must be a multiple of 4. The MISTY1 with 8 rounds is most often used. Before describing the MISTY1 in detail, the following symbols are given:

Table 1 symbols

X^n	X is a sequence of 0,1with n bits
X_L	the left half of X
X_R	the right half of X
$\|\|$	the connector of the sequence
$X[i:j]$	the i-th bit to the j-th bit of X
$X_{i,j}$	the j-th bit of X_i

2.4.1 Basic module of MISTY1

(1) FL function

FL is a linear transformation function with 32-bit input and output, which divides the 32-bit subkey KL_i^{32} and the input X^{32} into two equal left and right parts, i.e. $KL_i^{32}=KL_{i1}\,\|\,KL_{i2}$ and $X^{32}=X_L\,\|\,X_R$, then executes the following operations:

$$Y_R=(X_L\cap KL_{i1})\oplus X_R\,,Y_L=(X_L\cup KL_{i2})\oplus X_L.$$

(2) FO function

FO function is a nonlinear function with a 3-round Feistel structure, both its input and output are 32 bits. Its input is 32-bit data $X^{32}=(L_0\,\|\,R_0)$ and two 64-bit subkeys $KO_i(KO_{i1},KO_{i2},KO_{i3},KO_{i4})$ and 48-bit keys $KI_i(KI_{i1},KI_{i2},KI_{i3})$. For each round $j(1\le j\le 3)$, the following operation is executed:

$$R_j = FI_{ij}(L_{j-1} \oplus KO_{ij}, KI_{ij}) \oplus R_{j-1}, L_j = R_{j-1}.$$

Then, the output of FO is $Y^{32} = (L_3 \oplus KO_{i4}) \| R_3$.

(3) FI function

FI is a 3-round Feistel structure function with 16-bit input and output. FI consists of two S-boxes, S7 and S9. With the input X^{16} and KI_{ij}, FI_{ij} split them as follows:

$$X^{16} = L_0^9 \| R_0^7, KI_{ij} = KI_{ij,1} \| KI_{ij,2}.$$

where $KI_{ij,1} = KI_{ij}[0:6]$ is a 7 bits key, $KI_{ij,2} = KI_{ij}[7:15]$ is a 9 bit key, and then executes the following operation:

$$L_1^7 = R_0^7, R_1^9 = S9(L_0^9) \oplus ZE(R_0^7).$$

$$L_2^9 = R_1^9 \oplus KI_{ij,2}, R_2^7 = S7(L_1^7) \oplus TR(R_1^9) \oplus KI_{ij,1}.$$

$$L_3^7 = R_2^7, R_3^9 = S9(L_2^9) \oplus ZE(R_2^7).$$

where $ZE(T)$ means to add two zeros before the first bit of T (Zero-Extend), $TR(T)$ means to truncate the first two bits of T (Truncate). At last FI function output $Y^{16} = L_3^7 \| R_3^9$.

(4)S-box

MISTY1 uses two S-boxes, S9 with 9-bit input and output and S7 with 7-bit input and output, both of which can be implemented by look-up table.

S7 has 2^7 entries of size 1 bytes, and thus table of S7 require 128 bytes of memory. S9 has 2^9 entries of size 2 bytes, and thus table of S9 require 1KB of memory. This is the essential reason of the leakage on MISTY1.

2.4.2 Encryption of MISTY1

Taking the 8-round MISTY1 as an example, 64-bit plaintext P^{64} split into equal lengths and two parts L_0 and R_0, then the encryption process executes the following operations:

(1) For odd rounds $i(i = 1,3,5,7)$, do:

$$R_i = FL_i(L_{i-1}, KL_i), L_i = FL_{i+1}(R_{i-1}, KL_{i+1}) \oplus FO(R_i, KO_i, KI_i).$$

(2) For even rounds $i(i = 2,4,6,8)$, do:

$$R_i = L_{i-1}, L_i = R_{i-1} \oplus FO(R_i, KO_i, KI_i).$$

(3) At the end of 8 rounds of encryption, do:

$$R_9 = FL_9(L_8, KL_9), L_9 = FL_{10}(R_8, KL_{10}).$$

(4)Finally, output 64 bit ciphertext:

$$C^{64} = L_9 \parallel R_9$$

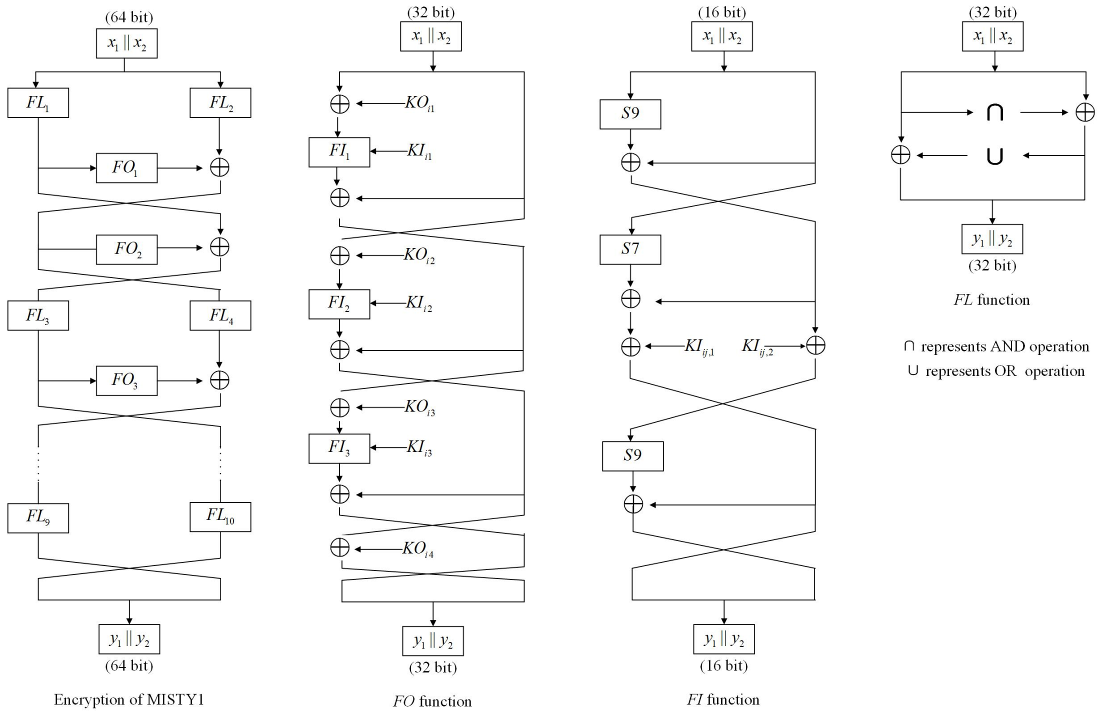

Figure 1 flow chart of MISTY1

2.4.3 Key scheduling part of MISTY1

The key scheduling part of MISTY1 first divides 128-bit master key K into eight 16-bit subkeys:

$$K = K_0 \parallel K_1 \parallel K_2 \parallel K_3 \parallel K_4 \parallel K_5 \parallel K_6 \parallel K_7$$

Then, according to $K_i' = FI_{K_{i+1}}(K_i)$, eight 16-bits subkeys could be generated:

$$K_0', K_1', K_2', K_3', K_4', K_5', K_6', K_7'$$

The correspondence between the round subkeys $KO_{ij}, KI_{ij}, KL_{ij}$ and the actual subkeys KO_i, KI_i, KL_i is as follows, where i is identified with i-8 when $i > 8$:

Table 2 Round key generation of MISTY1

Round key	KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	$KL_{i1} \parallel KL_{i2}$
Value	K_i	K_{i+2}	K_{i+7}	K_{i+4}	K_{i+5}'	K_{i+1}'	K_{i+3}'	$K_{(i+1)/2} \parallel K_{(i+1)/2+6}'$ odd i

								$K'_{i/2+2} \parallel K_{i/2+4}$ even i

Each round of the FO function uses seven 16-bit round subkeys and the FL function uses two 16-bit round subkeys. When the subkeys computed by $K'_i = FI_{K_{i+1}}(K_i)$, K_i is the input of S7 and S9, which leads to the leakage many bits of key and makes it easier for an adversary to recover the 128-bit key, details are given in Section 3.

3 Leakage in key scheduling and encryption of MISTY1

3.1 Leakage in key scheduling of MISTY1

The key scheduling part of the MISTY1 is related to K_i and K_{i+1}. Let $j = i+1$, K_{i1} and K_{i2} denote the first 9 bits and the last 7 bits of K_i, K_{j1} and K_{j2} denote the first 7 bits and the last 9 bits of K_j. See Figure 2 for the procedure of the key scheduling part.

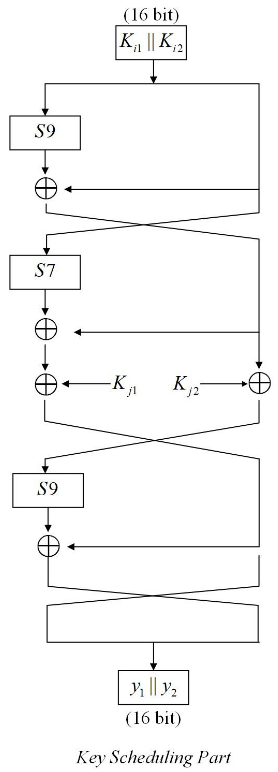

Figure 2 Key Scheduling Part

In the software implementation of the MISTY1, S9 has 2^9 entries of size 2 bytes, thus the table of S9 require 1KB of memory. S7 has 2^7 entries of size 1 bytes, thus the table of S7 require 128 bytes of memory. Assuming that the size of the cache line is 2^k bytes and the adversary can make the cache attack on S9 and S7. As Section 2.3 describes, the adversary can recover the first $10-k$ bits of K_{i1} and the first $7-k$ bits of K_{i2} when the victim accessing the first S9 and S7. Moreover, the victim accessing the second S9 , the adversary can obtain the first $10-k$ bits of $y = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2}$.

After the whole key scheduling part completed, the adversary can recover the following three parts of leakage:

(1) $K_i[0:9-k](0 \le i \le 7)$

(2) $K_i[9:15-k](0 \le i \le 7)$

(3) $y_i[0:9-k](0 \le i \le 7)$

3.2 Leakage in encryption of MISTY1

We focus on the calls to the FI function during encryption. For each encryption, the order of the round keys used by the FI function is determined, as shown in Table 3.

Table 3 Round keys used by FI function

Round	Round keys		
1	K'_5	K'_1	K'_3
2	K'_6	K'_2	K'_4
3	K'_7	K'_3	K'_5
4	K'_0	K'_4	K'_6
5	K'_1	K'_5	K'_7
6	K'_2	K'_6	K'_0
7	K'_3	K'_7	K'_1
8	K'_4	K'_0	K'_2

If the adversary can make the cache attack on S9 and S7, he can record the cache trace of every access to S9 and S7 during encryption. In the online phase, the adversary needs to log 48 S9-accesses and 24 S7-accesses of each encryption and mark the corresponding K'_i.

In the offline phase, the adversary doesn't need to know the specific input and output of each FI function. Let the input of the i-th call of FI function be $x_i(= x_{i1} \| x_{i2})$, the input of the second S9 be z_i, and the round key be K'_{n_i}, the i-th call of FI function is shown in Figure 3.

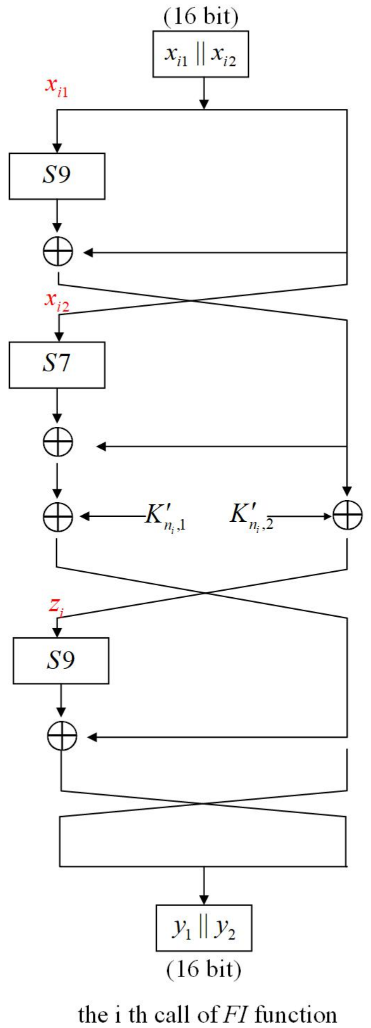

Figure 3 the i-th call of FI function

The relationship between $K'_{n_i,2}$, x_{i1} , x_{i2} and z_i is as follows:

$$K'_{n_i,2} = S9(x_{i1}) \oplus ZE(x_{i2}) \oplus z_i$$

The adversary can obtain the first $10-k$ bits of x_{i1} by the cache attack . Let Y be the set of the first S9 outputs, that is, $Y = \{S9(t) \mid t[0:9-k] = x_{i1}[0:9-k]\}$. By making cache attacks on S9 and S7, we get $z_i[0:9-k]$, $x_{i2}[0:6-k]$ and $x_{i1}[0:9-k]$. So we can use the elimination method to determine $K'_{n_i,2}[0:8-k]$. That is, if there is n, for any $y \in Y$, satisfies $n[0:8-k] \neq y[0:8-k]$, then we can know:

$$K'_{n_i,2}[0:8-k] \neq n[0:8-k] \oplus ZE(x_{i2}) \oplus z_i$$

The range size of $K'_{n_i,2}[0:8-k]$ ($= K'_{n_i}[7:15-k]$) is 2^{9-k} . Each cache attack on encryption of MISTY1 can eliminate some wrong values. We can determine $K'_{n_i,2}[0:8-k]$ accurately after observing 5-10 encryption. Since the generation of K'_{n_i} is related to the K_{n_i} and K_{n_i+1}, it will leakage the information of them, which can help the adversary recover the key of MISTY1.

4 Attack scheme for 32-byte cache line

4.1 Information Leakage

According to 2.3, for each cache attack on S9, the adversary can recover the first 5 bits of the input. And for each cache attack on S7, the adversary can obtain the first 2 bits of the input.

According to 3.1, in the leakage of the key scheduling part, the adversary can recover three parts of information of the key: (1) $K_i[0:4](0 \le i \le 7)$ (2) $K_i[9:10](0 \le i \le 7)$ (3) $y_i[0:4](0 \le i \le 7, y = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2})$

According to 3.2, the adversary can determine $K'_{i2}[0:3](0 \le i \le 7)$ accurately after observing some encryption.

4.2 Baseline attack in 32 byte cache line

Assuming that the adversary know a pair of plaintext and ciphertext (m,c) encrypted by the correct key of MISTY1. According to the leakage of key scheduling part, we propose a baseline attack on MISTY1 for 32-byte cache line, and the algorithm is divided into the following steps:

(0) By the cache attack, the adversary already know $K_i[0:4](0 \le i \le 7)$, $K_i[9:10](0 \le i \le 7)$ and $y_i[0:4](0 \le i \le 7, y = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2})$

(1) Exhaust the unknown 4 bits of $K_0[5:8]$ to compute $S9(K_0[0:8])$. According to the known information, if $K_0[5:8]$ is not satisfied the equation $y_0[2:3] = S9(K_0[0:8])[2:3] \oplus K_0[9:10] \oplus K_1[9:10]$, continue the loop; if it is satisfied, proceed to (2)

(2) Compute $K_1[7:8] = S9(K_0[0:8])[0:1] \oplus y_0[0:1]$ and $K_{0,11} \oplus K_{1,11} = S9(K_0[0:8])_4 \oplus y_{0,4}$. Recursively exhaust $K_i[5:6](1 \le i \le 7)$, and determine whether $K_i[0:8]$ satisfies equation

$y_i[2:3] = S9(K_i[0:8])[2:3] \oplus K_i[9:10] \oplus K_{i+1}[9:10]$. If not, continue the loop; if it satisfies, calculate $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$ and $K_{i,11} \oplus K_{i+1,11} = S9(K_i[0:8])_4 \oplus y_{i,4}$, then proceed to (3)

(3) Determine the sum of $K_{i,11} \oplus K_{i+1,11} (0 \le i \le 7)$ is 0 or not, if not, then return (2), if it is satisfied, then let $K_{0,11} = 0$ or 1, and calculate $K_{i,11} (1 \le i \le 7)$, proceed to (4)

(4) Exhaust $K_i[12:15] (0 \le i \le 7)$, determine if $\text{MISTY1}(m, K) = c$ holds or not. If it is satisfied, output the correct key K ; if not, continue the loop.

See Algorithm 1 for the Baseline attack for 32-byte cache line.

Algorithm 1 Baseline attack on MISTY1 in 32 byte cache line

```
input: Ki[0:4], Ki[9:10], yi[0:4], 0≤i≤7, encryption algorithm MISTY1
output: K = K0|| K1|| K2|| K3|| K4|| K5|| K6|| K7
for K0[5:8] = 2^4-1 downto 0 do
    if S9(K0[0:8])[2:3]⊕K0[9:10]⊕K1[9:10]≠y0[2:3] then
        continue
    else
        K1[7:8]=S9(K0[0:8])[0:1]⊕y0[0:1]
        K0,11⊕K1,11=S9(K0[0:8])4⊕y0,4
        for i = 1 upto 7 do
            for Ki[5:6]=2^2-1 downto 0 do
                if S9(Ki[0:8])[2:3]⊕Ki[9:10]⊕Ki+1[9:10]≠yi[2:3] then
                    continue
                else
                    Ki+1[7:9]=S9(Ki[0:8])[0:1]⊕yi[0:1]
                    Ki,11⊕Ki+1,11=S9(Ki[0:8])4⊕yi,4
                end if
           if sum(K0,11⊕K1,11, K1,11⊕K2,11, ..., K7,11⊕K0,11) ≠ 0 then
                continue
           else
                for j = 0 upto 7 do
                    for Kj[12:15] = 2^4-1 downto 0
                        if MISTY1(m, K) = c then
                            return K
                        else
                            continue
                        end if
                end for
         end for
  end if
```

28:end for

Complexity estimation about Algorithm 1: In step (1), we should first exhaust 4 bits of $K_0[5:8]$, and the equation (i) contains two bits of known information, making only 1/4 go to the next loop. So the complexity of step (1) is $O(2^{4-2})$.

In step (2), we should exhaust 2 bits of $K_i[5:6](1 \le i \le 7)$, and the equation (ii) contains two bits of known information, making only 1/4 go to the next loop. So the complexity of step (2) is $O(2^{7\times(2-2)})$.

In step (3), we should exhaust 1 bit of $K_{0,11}$, so the complexity of step (3) is $O(2)$.

In step (4), we should exhaust 4 bit of $K_i[12:15](0 \le i \le 7)$, so the complexity of step (4) is $O(2^{4\times 8})$.

Therefore, the total complexity of Algorithm 1 is $O(2^{4-2+7\times(2-2)+1+4\times 8}) = O(2^{35})$. Baseline attack can be used to recover the key when the adversary observes only one time encryption. If the adversary can observe multiple encryption, he can use the leakage of $K'_{n_i,2}[0:3]$ to reduce the complexity of the attack algorithm.

4.3 Improved attack on MISTY1 in 32 byte cache line

The improved attack reduces the complexity of step (4) in Algorithm 1, noted as Step (4*).

Step (4*) First, exhaust 4 bits of $K_0[12:15]$. Then recursively exhaust $K_i[12:15](1 \le i \le 7)$, and determines whether $FI(K_{i-1}, K_i)[8:11] = K'_{n_i,2}[0:3]$ holds at the same time, if not, continue the loop, if it holds, then determine whether K satisfies $MISTY1(m, K) = c$. If not, continue the loop; if it satisfies, output the correct key K and end the algorithm.

Algorithm 2 Improved attack on MISTY1 in 32 byte cache line

input: Ki[0:4], Ki[9:10], $y_i[0:4]$, $0 \le i \le 7$, encryption algorithm MISTY1

output: $K = K_0 \| K_1 \| K_2 \| K_3 \| K_4 \| K_5 \| K_6 \| K_7$

1: for $K_0[5:8] = 2^4$-1 downto 0 do

2: if $S9(K_0[0:8])[2:3] \oplus K_0[9:10] \oplus K_1[9:10] \ne y_0[2:3]$ then

```
        continue
    else
        K_1[7:8]=S9(K_0[0:8])[0:1]⊕y_0[0:1]
        K_{0,11}⊕K_{1,11}=S9(K_0[0:8])_4⊕y_{0,4}
        for i = 1 upto 7 do
            for K_i[5:6]=2^2-1 downto 0 do
                if S9(K_i[0:8])[2:3]⊕K_i[9:10]⊕K_{i+1}[9:10]≠y_i[2:3] then
                    continue
                else
                    K_{i+1}[7:9]=S9(K_i[0:8])[0:1]⊕y_i[0:1]
                    K_{i,11}⊕K_{i+1,11}=S9(K_i[0:8])_4⊕y_{i,4}
                end if
            if sum(K_{0,11}⊕K_{1,11}, K_{1,11}⊕K_{2,11}, ..., K_{7,11}⊕K_{0,11}) ≠ 0 then
                continue
            else
                for K_0[12:15] = 2^4-1 downto 0
                    for j = 1 upto 7 do
                        for K_j[12:15] = 2^4-1 downto 0
                            if FI(K_{j-1}, K_j)[8:11] ≠K'_j[0:3] then
                                continue
                            else if MISTY1(m, K) ≠ c then
                                continue
                            else
                                return K
                        end if
                end for
        end for
    end if
end for
```

Complexity estimation about Algorithm 2: In step (4*), we should first exhaust 4 bits of $K_0[12:15]$. Then recursively exhaust 4 bits of $K_i[12:15](1 \le i \le 7)$, and and the equation (iii) contains four bits of known information, making about 1/16 go to the next loop. so the complexity of step (4*) is $O(2^{4+7\times(4-4)})$.

Therefore, the total complexity of Algorithm 2 is $O(2^{4-2+7\times(2-2)+1+4+7\times(4-4)}) = O(2^7)$.

5 Attack scheme for 64-byte cache line

5.1 Information Leakage

According to 2.3, for each cache attack on S9, the adversary can recover the first 4 bits of the input. And for each cache attack on S7, the adversary can obtain the first 1 bit of the input.

According to 3.1, in the leakage of the key scheduling part, the adversary can recover three parts of information of the key: (1) $K_i[0:3](0 \le i \le 7)$ (2) $K_{i,9}(0 \le i \le 7)$ (3) $y_i[0:3](0 \le i \le 7, y = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2})$

According to 3.2, the adversary can determine $K'_{i2}[0:2](0 \le i \le 7)$ accurately after observing some encryption.

5.2 Baseline attack in 64 byte cache line

Assuming that the adversary know a pair of plaintext and ciphertext (m,c) encrypted by the correct key of MISTY1. According to the leakage of key scheduling part, we propose a baseline attack on MISTY1 for 64-byte cache line, and the algorithm is divided into the following steps:

(0) By the cache attack, the adversary already know $K_i[0:3](0 \le i \le 7)$, $K_{i,9}(0 \le i \le 7)$ and $y_i[0:3](0 \le i \le 7, y = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2})$

(1) Exhaust the unknown 5 bits of $K_0[4:8]$ to compute $S9(K_0[0:8])$. According to the known information, if $K_0[4:8]$ is not satisfied the equation $y_{0,2} = S9(K_0[0:8])_2 \oplus K_{0,9} \oplus K_{1,9}$, continue the loop; if it is satisfied, proceed to (2)

(2) Compute $K_1[7:8] = S9(K_0[0:8])[0:1] \oplus y_0[0:1]$ and $K_{0,10} \oplus K_{1,10} = S9(K_0[0:8])_3 \oplus y_{0,3}$. Recursively exhaust $K_i[4:6](0 \le i \le 7)$, and determine whether $K_i[0:8]$ satisfies equation $y_{i,2} = S9(K_i[0:8])_2 \oplus K_{i,9} \oplus K_{i+1,9}$. If not, continue the loop; if it satisfies, calculate $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$ and $K_{i,10} \oplus K_{i+1,10} = S9(K_i[0:8])_3 \oplus y_{i,3}$, then proceed to (3)

(3) Determine the sum of $K_{i,10} \oplus K_{i+1,10}(0 \le i \le 7)$ is 0 or not, if not, then return (2), if it is satisfied, then let $K_{0,11} = 0$ or 1, and calculate $K_{i,11}(1 \le i \le 7)$, proceed to (4)

(4) Exhaust $K_i[11:15](0 \le i \le 7)$, determine if $\text{MISTY1}(m,K) = c$ holds or not. If

it is satisfied, output the correct key K; if not, continue the loop.

See Algorithm 3 for the Baseline attack for 64-byte cache line.

Algorithm 3 Baseline attack on MISTY1 in 64 byte cache line

```
input: Ki[0:3], Ki9:, y_i[0:3], 0≤i≤7, encryption algorithm MISTY1
output: K = K_0|| K_1|| K_2|| K_3|| K_4|| K_5|| K_6|| K_7
for K_0[4:8] = 2^5-1 downto 0 do
    if S9(K_0[0:8])_3 ⊕ K_{0,9} ⊕ K_{1,9} ≠ y_{0,2} then
        continue
    else
        K_1[7:8]=S9(K_0[0:8])[0:1] ⊕ y_0[0:1]
        K_{0,10} ⊕ K_{1,10}=S9(K_0[0:8])_3 ⊕ y_{0,3}
        for i = 1 upto 7 do
            for K_i[4:6]=2^3-1 downto 0 do
                if S9(K_i[0:8])_2 ⊕ K_{i9} ⊕ K_{i+1,9} ≠ y_{i,2} then
                    continue
                else
                    K_{i+1}[7:9]=S9(K_i[0:8])[0:1] ⊕ y_i[0:1]
                    K_{i,10} ⊕ K_{i+1,10}=S9(K_i[0:8])_3 ⊕ y_{i,3}
                end if
            if sum(K_{0,10} ⊕ K_{1,10}, K_{1,10} ⊕ K_{2,10}, ..., K_{7,10} ⊕ K_{0,10}) ≠ 0 then
                continue
            else
                for j = 0 upto 7 do
                    for K_j[11:15] = 2^5-1 downto 0
                        if MISTY1(m, K) = c then
                            return K
                        else
                            continue
                        end if
                end for
        end for
    end if
end for
```

Complexity estimation about Algorithm 3: In step (1), we should first exhaust 5 bits of $K_0[4:8]$, and the equation (iv) contains one bit of known information, making only 1/2 go to the next loop. So the complexity of step (1) is $O(2^{5-1})$.

In step (2), we should exhaust 3 bits of $K_i[4:6](1 \le i \le 7)$, and the equation (vi) contains one bit of known information, making only 1/2 go to the next loop. So the complexity of step (2) is $O(2^{7\times(3-1)})$.

In step (3), we should exhaust 1 bit of $K_{0,11}$, so the complexity of step (3) is $O(2)$.

In step (4), we should exhaust 4 bit of $K_i[11:15](0 \le i \le 7)$, so the complexity of step (4) is $O(2^{5\times8})$.

Therefore, the total complexity of Algorithm 1 is $O(2^{5-1+7\times(3-1)+1+5\times8}) = O(2^{59})$. Baseline attack can be used to recover the key when the adversary observes only one time encryption. If the adversary can observe multiple encryption, he can use the leakage of $K'_{n_i,2}[0:2]$ to reduce the complexity of the attack algorithm.

5.3 Improved attack on MISTY1 in 64 byte cache line

The improved attack reduces the complexity of step (4) in Algorithm 3, noted as Step (4*).

Step (4*) First, exhaust 5 bits of $K_0[11:15]$. Then recursively exhaust $K_i[11:15](1 \le i \le 7)$, and determines whether $FI(K_{i-1}, K_i)[8:10] = K'_{n_i,2}[0:2]$ holds at the same time, if not, continue the loop, if it holds, then determine whether K satisfies $MISTY1(m, K) = c$. Continue the loop if not and output the correct key K if it satisfies.

Algorithm 4 Improved attack on MISTY1 in 64 byte cache line

```
input: Ki[0:3], Ki9:, yi[0:3], 0≤i≤7, encryption algorithm MISTY1
output: K = K0|| K1|| K2|| K3|| K4|| K5|| K6|| K7
for K0[4:8] = 2^5-1 downto 0 do
    if S9(K0[0:8])3 ⊕ K0,9 ⊕ K1,9≠y0,2then
        continue
    else
        K1[7:8]=S9(K0[0:8])[0:1] ⊕ y0[0:1]
        K0,10 ⊕ K1,10=S9(K0[0:8])3 ⊕ y0,3
        for i = 1 upto 7 do
            for Ki[4:6]=2^3-1 downto 0 do
                if S9(Ki[0:8])2 ⊕ Ki9 ⊕ Ki+1,9≠yi,2 then
                    continue
                else
                    Ki+1[7:9]=S9(Ki[0:8])[0:1] ⊕ yi[0:1]
                    Ki,10 ⊕ Ki+1,10=S9(Ki[0:8])3 ⊕ yi,3
                end if
```

```
            if sum(K0,10⊕K1,10, K1,10⊕K2,10, ..., K7,10⊕K0,10) ≠ 0 then
                continue
            else
                for K0[12:15] = 2^4-1 downto 0
                    for j = 1 upto 7 do
                        for Kj[12:15] = 2^4-1 downto 0
                            if FI(Kj-1, Kj)[8:11] ≠K'j[0:3] then
                                continue
                            else if MISTY1(m, K) ≠ c then
                                continue
                            else
                                return K
                        end if
                end for
        end for
    end if
end for
```

Complexity estimation about Algorithm 2: In step (4*), we should first exhaust 5 bits of $K_0[11:15]$. Then recursively exhaust 4 bits of $K_i[12:15](1 \le i \le 7)$, and $K'_{n_i,2}[0:2]$ making about 1/8 go to the next loop. so the complexity of step (4*) is $O(2^{5+7\times(5-3)})$.

Therefore, the total complexity of Algorithm 4 is $O(2^{5-1+7\times(3-1)+1+5+7\times(5-3)}) = O(2^{38})$.

6 Experimental Results

We build 32-byte cache lines experiment in gem5 simulation configuration [30] with architecture of x86-64 at 2.0 GHz , 32 KB L1-I/L1-D Cache, 2 MB L2 Cache and 32B cache line. We conduct 64-byte cache line experiment on Huawei Matebook 14 with Intel(R) Core(TM) i5-8265U CPU, 1.60 GHz, 256 KB L1-I/L1-D Cache, 1 MB L2 Cache and 64B cache line. The offline analysis program was written in Python and run on a Huawei Matebook 14 laptop.

We assume that the adversary has get a pair of plaintext and ciphertext encrypted by MISTY1. To obtain cache usage information, we use the Flush+Reload attack in Mastik toolkit[25]. In this section, we use the key 0x00112233445566778899aabbccddeeff for our experiments.

6.1 Leakage of key scheduling part

6.1.1 32-byte cache line

Recall Section 3, in 32-byte cache line environment S9 needs 32(=1KB/32B) cache lines to storage. From the start to the end of S9, we perform Flush+Reload on the 32 cache lines in turn and record the latency. The shortest time corresponding to the first 5 bits of the S9 input. Figure 4 shows the latency during Flush+Reload.

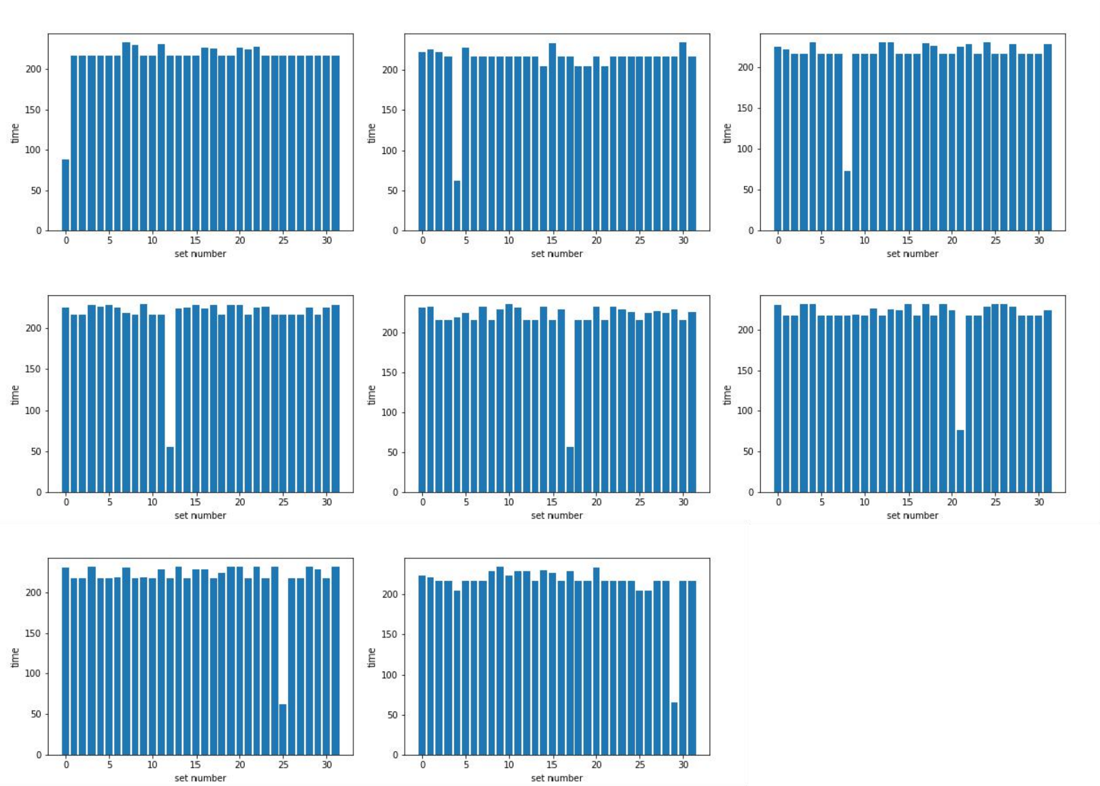

Figure 4 Leakage of $K_i[0:4]$

Recall Section 4, it leaks the first 5 bits of K_i, they are 0x0, 0x4, 0x8, 0xc, 0x11, 0x15, 0x19, 0x1d.

Recall Section 3.1, in 32-byte cache line environment S7 needs 4(=128B/32B) cache lines to storage. From the start to the end of S7, we perform Flush+Reload on the 4 cache lines in turn and record the latency. The shortest time corresponding to the first 2 bits of the S7 input. Figure 5 shows the latency during Flush+Reload.

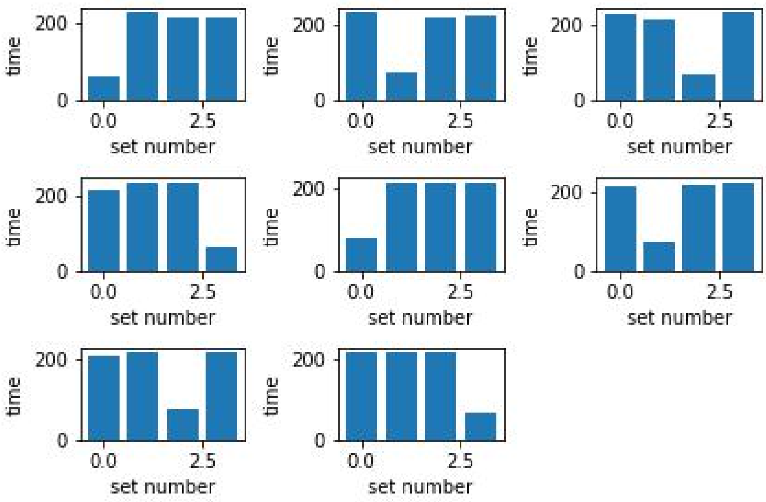

Figure 5 Leakage of $K_i[9:10]$

From the Figure 5, we can obtain $K_i[9:10]$, they are 0,1,2,3,0,1,2,3.

For the second S9 access, we can similarly obtain the first 5 bits the input, which corresponds to $y_i[0:4](0 \le i \le 7, y_i = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2})$

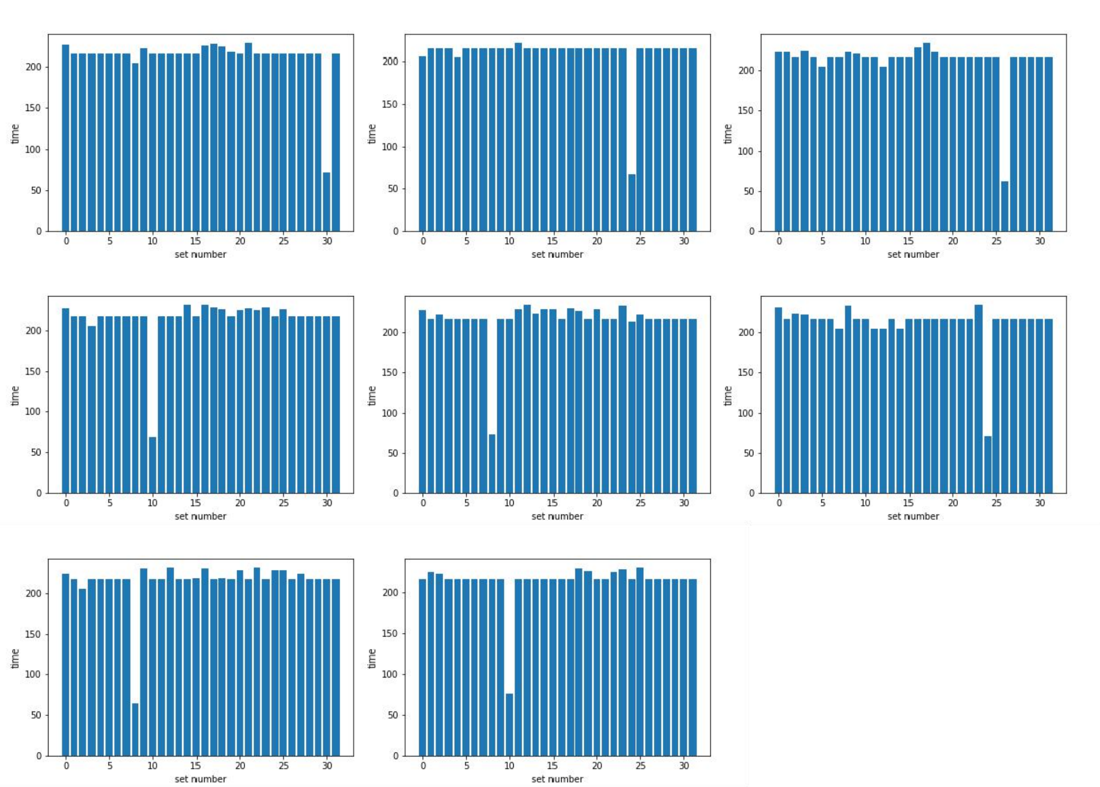

Figure 6 Leakage of $y_i[0:4]$

From the Figure 6, we can obtain $y_i[0:4]$, they are 0x1e, 0x18, 0x1a, 0xa, 0x8, 0x18, 0xa, 0xc.

6.1.2 64-byte cache line

In 64-byte cache line environment S9 needs 16(=1KB/64B) cache lines to storage. From the start to the end of S9, we perform Flush+Reload on the 16 cache lines in turn and record the latency. The shortest time corresponding to the first 4 bits of the S9 input, which showed in Figure 7.

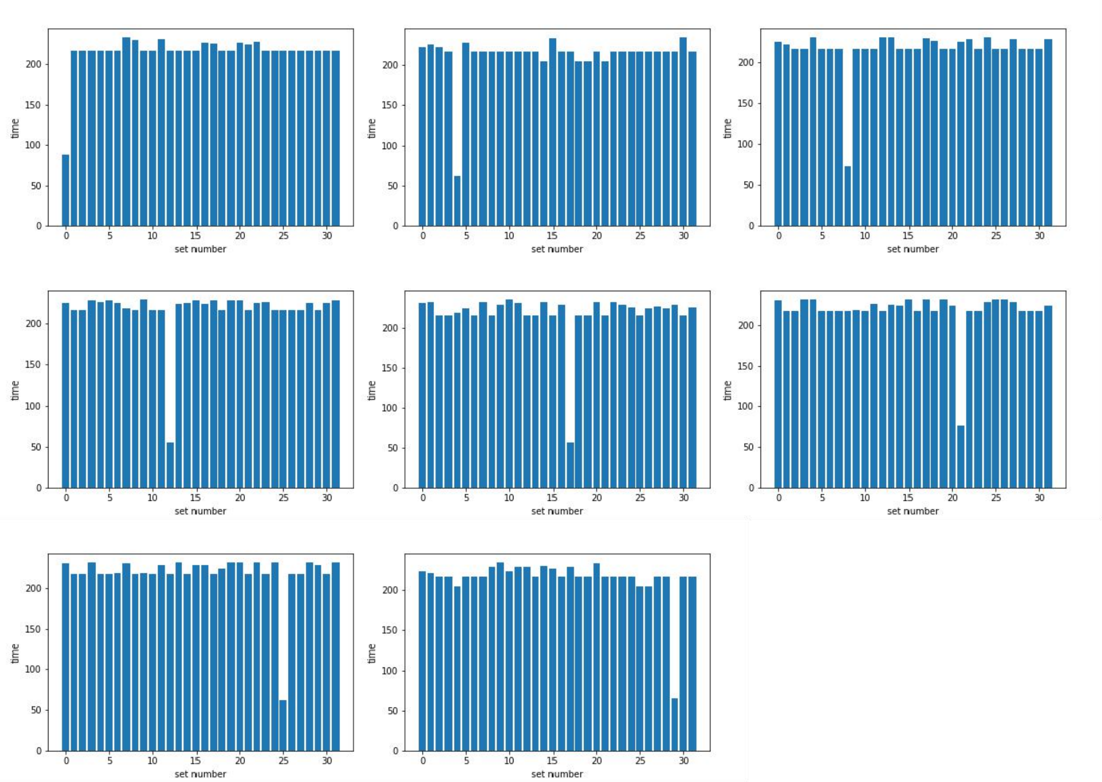

Figure 7 Leakage of $K_i[0:3]$

Recall Section 5, it leaks the first 4 bits of K_i, they are 0x0, 0x2, 0x4, 0x6, 0x8, 0xa, 0xc, 0xe.

In 64-byte cache line environment S7 needs 2(=128B/64B) cache lines to storage. From the start to the end of S7, we perform Flush+Reload on the 2 cache lines in turn and record the latency. The shortest time corresponding to the first 1 bit of the S7 input, which showed in Figure 8.

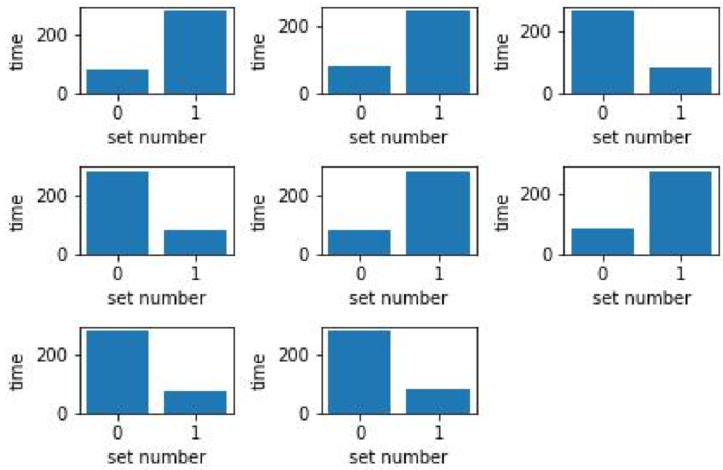

Figure 8 Leakage of $K_{i,9}$

From the Figure 5, we can obtain $K_{i,9}$, they are 0,0,1,1,0,0,1,1.

For the second S9 access, we can similarly obtain the first 4 bits the input, which corresponds to $y_i[0:3](0 \le i \le 7, y_i = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2})$

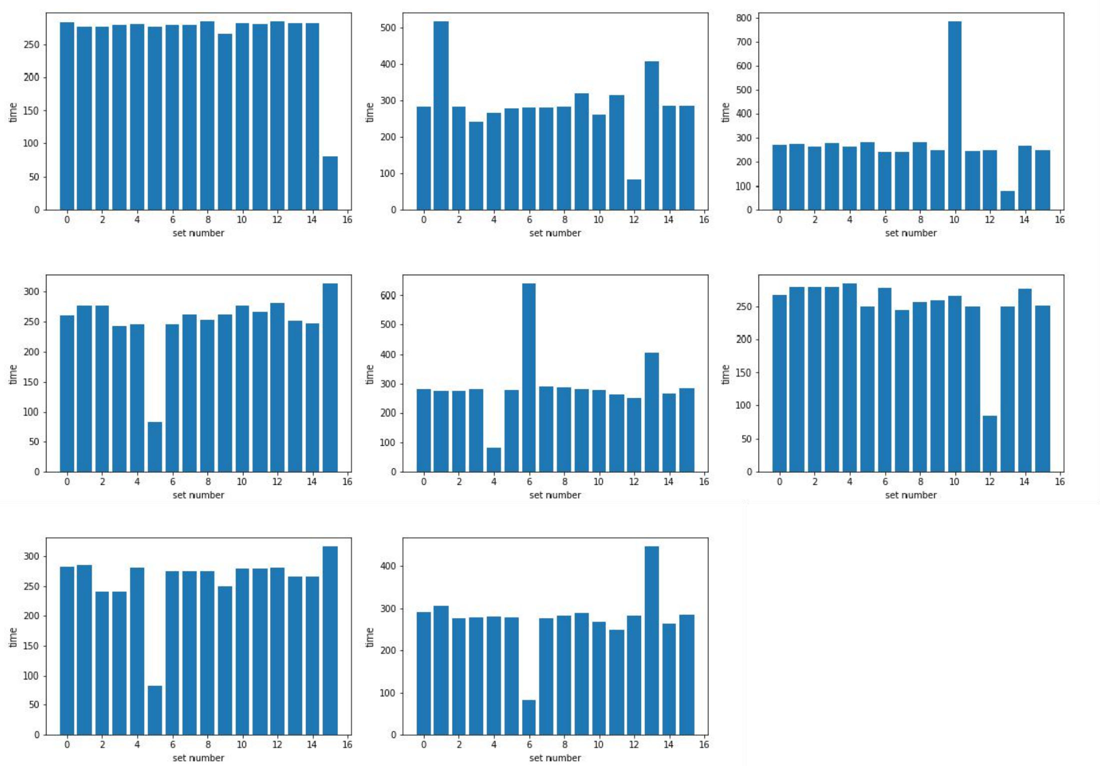

Figure 9 Leakage of $y_i[0:3]$

From the Figure 9, we can obtain $y_i[0:3]$, they are 0xf, 0xc, 0xd, 0x5, 0x4, 0xc, 0x5, 0x6.

6.2 Confirm the leakage of K_i'

Recall to Section 3.2, We can determine $K_i'[0:8-k]$ accurately after observing some encryption. And it is vital for Algorithm 2 and 4.

6.2.1 32-byte cache line

Each $K_i'[0:3](0 \le i \le 7)$ has $2^4 = 16$ possibilities. After observing one encryption, the adversary gets the space size as Figure 10 shows:

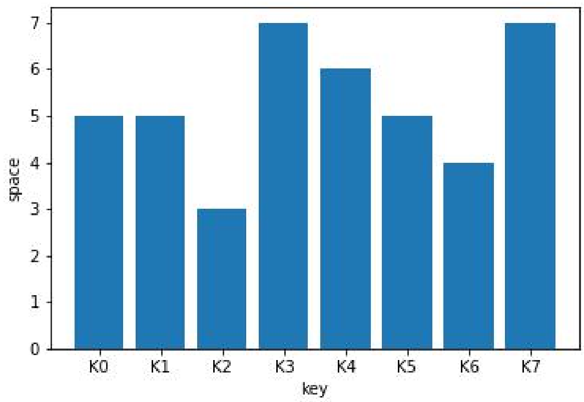

Figure 10 Space of $K_i'[0:3]$ after observing one encryption

Observing one encryption makes the adversary drop the fetch space from 2^{32} to $2^{18.75} \approx (5\times5\times3\times7\times6\times5\times4\times7)$.

After observing 5 times encryption, the adversary can determine the value of $K_{n_i,2}'[0:3]$, as Figure 11 shows:

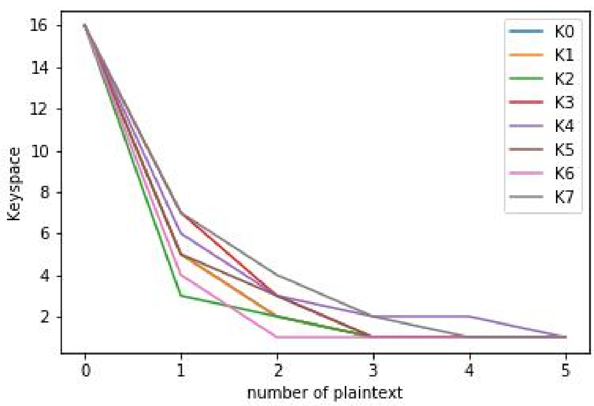

Figure 11 Determine $K_i'[0:3]$ after observing 5 times encryption

6.2.2 64-byte cache line

Each $K_i'[0:2](0 \le i \le 7)$ has $2^3 = 8$ possibilities. After observing one encryption, the adversary gets the space size as Figure 12 shows:

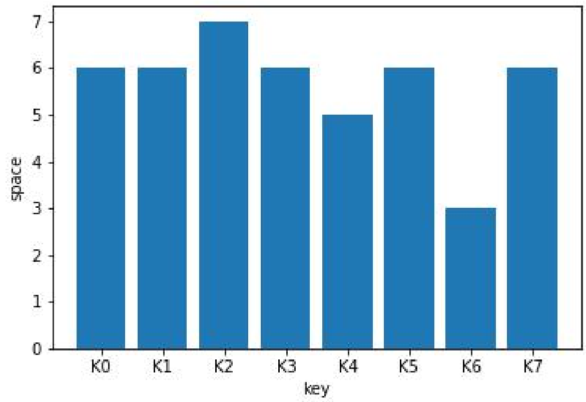

Figure 12 Space of $K_i'[0:2]$ after observing one encryption

Observing one encryption makes the adversary drop the fetch space from 2^{24} to $2^{19.64} \approx (6 \times 6 \times 7 \times 6 \times 5 \times 6 \times 3 \times 6)$.

After observing 10 times encryption, the adversary can determine the value of $K'_{n_i,2}[0:2]$, as Figure 13 shows:

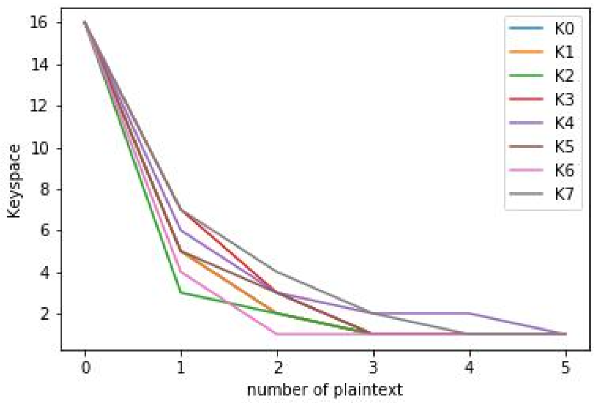

Figure 11 Determine $K_i'[0:2]$ after observing 10 times encryption

6.3 Recover the whole 128-bit key

6.3.1 32-byte cache line

When more than 5 times of cache leakage from encryption are available, the adversary first needs to confirm $K_i'[0:3](0 \le i \le 7)$, and then use Algorithm 2 to recover the key of MISTY1. We find the correct key in only 0.035s.

Let us consider a worse case, assuming that the adversary only observes only one time encryption, then the adversary can exhaust the $K_i'[0:3]$ space and then use Algorithm 4 for the attack, then the attacker needs $O(2^{25.75})$ time complexity at worst for the attack, which takes 5145.33 seconds(about 1.4 h) on a personal laptop to complete the attack.

6.3.2 64-byte cache line

When more than 10 times of cache leakage from encryption are available, the adversary first needs to confirm $K_i'[0:2](0 \le i \le 7)$, and then use Algorithm 4 to recover the key of MISTY1. We find the correct key in 7661.06 seconds(about 2.1 hours).

Let us consider a worse case, assuming that the adversary only observes only one time encryption, then the adversary can exhaust the $K_i'[0:2]$ space and then use Algorithm 4 for the attack, then the attacker needs $O(2^{57.64})$ time complexity at worst for the attack. See Table 4 for complexity under different conditions.

Table 4 Complexity of attack algorithm under different conditions

		32-byte cache line	64-byte cache line
Observe one encryption	Algorithm 1/3	$O(2^{35})$	$O(2^{59})$
	Algorithm 2/4	$O(2^{25.75})$	$O(2^{57.64})$
Observe multiple encryption		$O(2^{7})$	$O(2^{38})$

7 Discussion and Conclusion

Contermeasures. There are two basic ideas for defenses: one is to completely eliminate cache leakage; the other is to eliminate the correlation between cache leakage and secret information.

In the first idea, using constant-time program to prevent secret-related memory accesses [26] and tuning the operating system so that it can preload certain data every time a process is activated [7] are both effective approaches. In our case, if S9 and S7 are loaded into the cache in advance, the adversary will not be able to obtain information by cache hit and cache miss.

In the second idea, the randomization-based approach [27] separates the memory line from the cache set, which may prevent the adversary from finding the cache set corresponding to the memory location. In addition to this, masking schemes [6] are also effective, applying masks (implemented at the hardware level) to the offset fields based on some unset addressing bits in the physical address. At this point the user no longer has control over the offset field, so he cannot initialize the particular set he wants to use as a target in the tertiary cache, nor can he determine if the set is used by the victim.

The use of S9. The MISTY1 uses S9 to increase the security of the algorithm. However, since the output length of S9 is larger than 8 bit, it has to use 2 byte of memory to store it, which results in the entire S9 requiring 1K of memory. This means the cache line size has to be larger than 1K to not generate leakage, which is difficult to achieve. To avoid cache attacks, block cipher should use small S-boxes as much as possible.

Next work. Our attack can be extended to the cloud environment, but most cloud platforms nowadays disable the clflush command and use non-inclusive cache. Fortunately, many advanced cache attack techniques are proposed to support it [28, 29]. The adversary does not need to share any virtual memory with the victim, nor does he need to share the same processor core. The cache attack can succeed even when the victim and adversary are run on different processor cores and do not share virtual memory by exploiting hardware characteristics of the last-level cache (LLC), which is shared across cores [29]. When the adversary can observe the accesses to S9 and S7, he can run our attack algorithm to recover the whole key of MISTY1.

Conclusion. In this work we implemented the Flush+Reload attack on the block cipher MISTY1, and we recover the 128-bit key in 5145.33 seconds (about 1.4hours) and 0.035 seconds in the 32-byte cache line environment by observing 1 and 5 times encryption. Moreover, in the 64-byte environment we observe 10 times encryption to recover the 128-bit full key in 7661.06 seconds (about 2.1 hours). Our work demonstrates that the application of S9 and the design of the key scheduling part make MISTY1 more vulnerable to cache attacks.

REFERENCE

[1] P. Kocher, “Timing attacks on implementations of Diffifie Hellman, RSA, DSS, and other systems,” in CRYPTO ’96, ser. LNCS, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113.

[2] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, “Side channel cryptanalysis of product ciphers,” Journal of Computer Security, vol. 8, no. 2/3, pp. 141–158, 2000.

[3] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, “Cryptanalysis of DES implemented on computers with cache,” in CHES ’03, ser. LNCS, C. D. Walter, C¸ . Koc¸, and C. Paar, Eds., vol. 2779. Springer, 2003, pp. 62–76.

[4] Bonneau J, Mironov I. Cache-collision timing attacks against AES[C]//International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2006: 201-215.

[5] Neve M, Seifert J P. Advances on access-driven cache attacks on AES[C]//International Workshop on Selected Areas in Cryptography. Springer, Berlin, Heidelberg, 2006: 147-162.

[6] Irazoqui G, Eisenbarth T, Sunar B. S $ A: A shared cache attack that works across cores and defies VM sandboxing--and its application to AES[C]//2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 591-604.

[7] Gullasch D, Bangerter E, Krenn S. Cache games--bringing access-based cache attacks on AES to practice[C]//2011 IEEE Symposium on Security and Privacy. IEEE, 2011: 490-505.

[8] Lou X, Zhang F, Xu G, et al. Enhanced Differential Cache Attacks on SM4 with Algebraic Analysis and Error-Tolerance[C]//International Conference on Information Security and Cryptology. Springer, Cham, 2019: 480-496.

[9] Bae D, Hwang J, Ha J. Flush+ Reload Cache Side-Channel Attack on Block Cipher ARIA[J]. Journal of the Korea Institute of Information Security & Cryptology, 2020, 30(6): 1207-1216.

[10] Poddar R, Datta A, Rebeiro C. A cache trace attack on CAMELLIA[C]//International Conference on Security Aspects in Information Technology. Springer, Berlin, Heidelberg, 2011: 144-156.

[11] Genkin D, Poussier R, Sim R Q, et al. Cache vs. key-dependency: Side channeling an implementation of Pilsung[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020: 231-255.

[12] Matsui M. New block encryption algorithm MISTY[C]//International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, 1997: 54-68.

[13] Tsunoo Y, Tsujihara E, Shigeri M, et al. Improving cache attacks by considering cipher structure[J]. International Journal of Information Security, 2006, 5(3): 166-176.

[14] “Intel 64 and IA-32 architectures optimization reference manual,” http://www.intel.com/Assets/PDF/manual/248966.pdf, 2010, Intel Corporation.

[15] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations”. In: IEEE SP. 2019, pp. 435–452.

[16] Daniel J. Bernstein, Joachim Breitner, Daniel Genkin, Leon Groot Bruin derink, Nadia Heninger, Tanja Lange, Christine van Vredendaal, and Yuval Yarom. “Sliding Right into Disaster: Left-to-Right Sliding Windows Leak”. In: CHES. 2017, pp. 555–576.

[17] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. “Last-Level Cache Side-Channel Attacks are Practical”. In: IEEE SP. 2015, pp. 605–622.

[18] Jancar J, Sedlacek V, Svenda P, et al. Minerva: The curse of ECDSA nonces[J]. IACR

Transactions on Cryptographic Hardware and Embedded Systems, 2020: 281-308.
[19] Aranha D F, Novaes F R, Takahashi A, et al. Ladderleak: Breaking ecdsa with less than one bit of nonce leakage[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 225-242.
[20] Genkin D, Pachmanov L, Pipman I, et al. ECDSA key extraction from mobile devices via nonintrusive physical side channels[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 1626-1638.
[21] Pereida García C, Brumley B B, Yarom Y. Make sure DSA signing exponentiations really are constant-time[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 1639-1650.
[22] Aldaya A C, García C P, Brumley B B. From A to Z: Projective coordinates leakage in the wild[J]. IACR Cryptol. ePrint Arch., 2020, 2020: 432.
[23] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. “CacheQuote: Effiffifficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks”. In: IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018.2 (2018), pp. 171–191.
[24] Yuval Yarom and Katrina Falkner. Flush+Reload: a high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, Proceedings of the 23rd USENIX Security Symposium, pages 719–732. USENIX Association, 2014.
[25] Yuval Yarom. Mastik: A Micro-Architectural Side-Channel Toolkit. http://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf. Sept. 2016.
[26] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. “The Security Impact of a New Cryptographic Library”. In: LATINCRYPT. 2012, pp. 159–176.
[27] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. “ScatterCache: Thwarting Cache Attacks via Cache Set Randomization”. In: USENIX Security. 2019.
[28] Liu F, Yarom Y, Ge Q, et al. Last-level cache side-channel attacks are practical[C]//2015 IEEE symposium on security and privacy. IEEE, 2015: 605-622.
[29] Yan M, Sprabery R, Gopireddy B, et al. Attack directories, not caches: Side channel attacks in a non-inclusive world[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 888-904.
[30] Binkert N, Beckmann B, Black G, et al. The gem5 simulator[J]. ACM SIGARCH computer architecture news, 2011, 39(2): 1-7.