# Cache attack on MISTY1

Haopeng Fan[1*], Wenhao Wang[2], Yongjuan Wang[1], Wenyu Zhang[1], Qingjun Yuan[1]

[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 physical implementations of cryptographic systems. Cache attacks have improved at recovering information by combining observations of the victim's cache access and knowledge of the cipher's structure. Cache attacks have been implemented for most Feistel- and SPN-structured block cipher algorithms, but the security of algorithms for special structures has seen little attention.

We perform a Flush+Reload attack on MISTY1, a class of 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, the key scheduling part of MISTY1 leaks many bits of the key, which, together with the leakage of partial bits of the round key during encryption, is sufficient to recover it.

We design an algorithm that can recover the MISTY1 128-bit key after observing encryption one time, and then use leakage during encryption to reduce its complexity. We experiment on 32- and 64-byte cache line environments. An adversary need observe as little as 5 encryptions to recover the 128-bit key in 0.035 second in the first case, and 10 encryptions to recover the key in 2.1 hours in the second case.

Keywords: Side channel, Cache attack, Flush+Reload, MISTY1, key scheduling part

## 1 Introduction

The theoretical security of cryptographic systems does not ensure implementation security. A side-channel attack is a major type of implementation-level attack that exploits the leakage of electromagnetic radiation, power consumption, runtime or even light, sound and heat during encryption. Cache attacks based on cache access mechanisms of microprocessors have become an active research area. 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 during the execution of certain operations leaks partial bits of the key.

In the block cipher, 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, an adversary can directly obtain partial bits of the S-box input by observing which cache line is accessed, and derive the entire key based on the leakage generated by the encryption of multiple plaintexts.

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

Unlike the above block cipher algorithms, MISTY1 [12] is not a Feistel or SPN structure cipher algorithm; it has a unique *recursive structure*. MISTY1 runs the $FL$ 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 it difficult for the adversary to attack the first and last rounds, so previous methods in block ciphers cannot be applied directly to MISTY1. Tsunoo et al. discussed cache attacks on MISTY1 in a 32-byte cache line environment [13], using the average method, which requires $2^{16}$ plaintexts to recover the entire key in a practical setting. In fact, Tsunoo et al. [13] did not fully investigate the leakage generated by the key scheduling part and encryption of MISTY1, and it would be difficult for the adversary to observe such a large number of encryptions in a real-world environment.

So, what leakage exists in MISTY1, and how can one perform a cache attack on it 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 in a cache attack.

**Leakage of encryption:** Owing to the complex structure of MISTY1, we do not need to ensure the exact input and output of each round, but can reduce the round key space by elimination. 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. During online observation, the adversary uses a Flush+Reload attack to collect the leakage generated by key scheduling and encryption. In the offline analysis phase, the adversary gets some bits of the subkey directly, based on leakage of the key scheduling part. The adversary must then set the judgment conditions to reduce the attack's complexity, based on the relationship between adjacent subkeys and the leakage of the round key during encryption. In summary, we need only to observe a very small number of encryptions to recover the 128-bit key.

**Attack results:** We experiment in 32- and 64-byte cache line environments. In the first case, the adversary only needs to observe 5 encryptions to recover the entire key; the complexity of the attack algorithm is O( $2^7$ ), and the runtime is 0.035 s on a personal laptop. In the second case, the adversary must observe 10 encryptions to recover the key, the complexity of the attack is O( $2^{38}$ ), and the running time on a personal laptop is 7661.06 s (about 2.1 h).

**Our Contributions:**

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

**Document Outline:**

Section 2 briefly introduces 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 part and encryption of MISTY1. Sections 4 and 5 show how to attack MISTY1 in 32- and 64-byte cache line environments, respectively. Based on Sections 4 and 5, Section 6 experiment in the real world and provides the results. We summarise the article in Section 7.

## 2 Background

### 2.1 Cache attack

CPU cache is very fast memory placed between a computer's main memory and CPU [14], whose size ranges from hundreds of kilobytes to a few megabytes. According to the principle of locality, which states that storage units accessed by the CPU tend to be clustered in a small, continuous region, CPU cache can effectively avoid high latency. Cache is stored in cache lines that are usually 64 bytes, while a few are 32 bytes.

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

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

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

The speed difference between a hit and miss may reveal information about cache contents. Moreover, because the contents of the cache depend on previous computations, to recover information about its contents can disclose information about previous computations. Cache attacks have been used to break 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, and it depends on the shared memory pages between the spy and crypto processes. In Intel processors, user threads can use the clflush instruction to flush readable and executable pages, enabling the adversary to use this instruction to frequently refresh the target memory block and measure the time to reload it. If the victim accesses the memory block during execution, 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 access will take longer. Therefore, the adversary can know if the victim accessed the memory block during execution. The Flush+Reload attack has 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

The substitution-box (S-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 S-box metrics directly determines the goodness of the entire block cipher.

When the S-box storage size is more than that of the cache line, it may leak some information about the key. For example, take $S_a$, an S-box with $a$ bits input and output. $S_a$ has $2^a$ entries of size $2^b$ bytes, so its table requires $2^{a+b}$ bytes of memory. If 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 $S_a$ are stored sequentially in cache lines. Two entries are stored in the same cache line only if they have the same first $a+b\text{-}k$ bits. So, an adversary who knows where $S_a$ is stored and can use Flush+Reload to know which cache line was accessed during the S-box operation 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]. It has been recommended as a transitional block cipher by the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) and listed as a block cipher standard by the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). Moreover, the Japanese CRYPTERC project selected it as a block cipher 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. Eight rounds are most often used. Table 1 provides some symbols used to describe MISTY1.

Table 1 Symbols

| | |
|---|---|
| $X^n$ | $X$ is a sequence of 0,1 with $n$ bits |
| $X_L$ | left half of $X$ |
| $X_R$ | right half of $X$ |
| $\|\|$ | connector of sequence |
| $X[i:j]$ | $i$-th to $j$-th bits of $X$ |
| $X_{i,j}$ | $j$-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 input $X^{32}$ into equal left and right parts, $KL_i^{32} = KL_{i1} \| KL_{i2}$ and $X^{32} = X_L \| X_R$, and 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* is a nonlinear function with a three-round Feistel structure, whose 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:

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

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

(3) *FI* function

*FI* is a three-round Feistel structure function with 16-bit input and output. *FI* consists of two S-boxes, S7 and S9. $FI_{ij}$ splits inputs $X^{16}$ and $KI_{ij}$ as

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

where $KI_{ij,1} = KI_{ij}[0:6]$ is a seven-bit key and $KI_{ij,2} = KI_{ij}[7:15]$ is a nine-bit key, and executes the following operations:

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

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

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

where $ZE(T)$ denotes the addition of two zeros before the first bit of $T$ (zero-extend), and $TR(T)$ means to truncate the first two bits of $T$ (truncate). $FI$ outputs $Y^{16} = L_3^7 \| R_3^9$.

(4) S-box

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

S7 has $2^7$ entries of size 1 byte; hence, its table requires 128 bytes of memory. S9 has $2^9$ entries of size 2 bytes, and its table requires 1 KB of memory. This is the essential reason for leakage of MISTY1.

2.4.2 Encryption of MISTY1

Taking the eight-round MISTY1 as an example, 64-bit plaintext $P^{64}$ is split into equal lengths and two parts $L_0$ and $R_0$, and 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 eight rounds of encryption, do

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

(4) Output 64-bit ciphertext:

$$C^{64} = L_9 \| R_9.$$

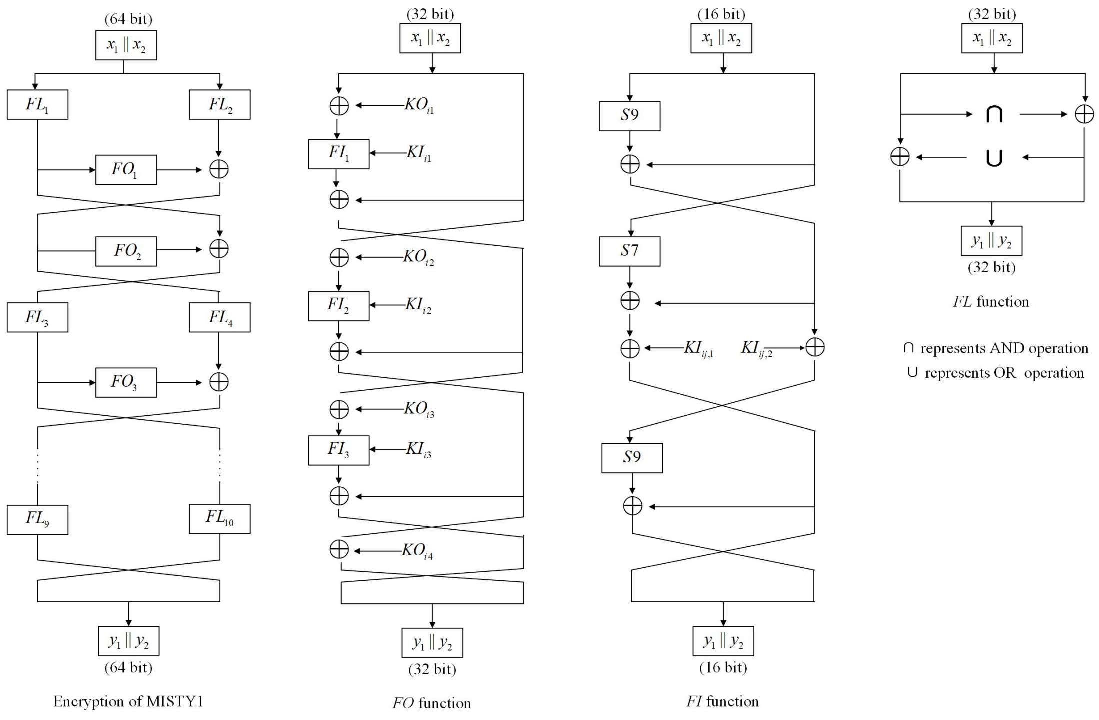

Figure 1 Flowchart of MISTY1

### 2.4.3 Key scheduling part of MISTY1

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

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

According to $K'_i = FI_{K_{i+1}}(K_i)$, eight 16-bit subkeys can 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 actual subkeys $KO_i, KI_i, KL_i$ is shown in Table 2, where $i$ is identified with $i$-8 when $i > 8$.

Table 2 Round key generation of MISTY1

| Round key | $KO_{i1}$ | $KO_{i2}$ | $KO_{i3}$ | $KO_{i4}$ | $KI_{i1}$ | $KI_{i2}$ | $KI_{i3}$ | $KL_{i1} \| KL_{i2}$ |
|---|---|---|---|---|---|---|---|---|
| Value | $K_i$ | $K_{i+2}$ | $K_{i+7}$ | $K_{i+4}$ | $K'_{i+5}$ | $K'_{i+1}$ | $K'_{i+3}$ | $K_{(i+1)/2} \| K'_{(i+1)/2+6}$ odd $i$<br>$K'_{i/2+2} \| K_{i/2+4}$ even $i$ |

Each round of the $FO$ function uses seven 16-bit round subkeys, and the $FL$ function uses two 16-bit round subkeys.

## 3 Leakage in key scheduling and encryption of MISTY1

## 3.1 Leakage in key scheduling of MISTY1

The key scheduling part of MISTY1 is related to $K_i$ and $K_{i+1}$. Let $j=i+1$, $K_i = K_{i1} \| K_{i2}$, $K_j = K_{j1} \| K_{j2}$ $K_{i1}$ and $K_{j2}$ are nine bits while $K_{i2}$ and $K_{j1}$ are seven bits. Figure 2 represents the key scheduling part.

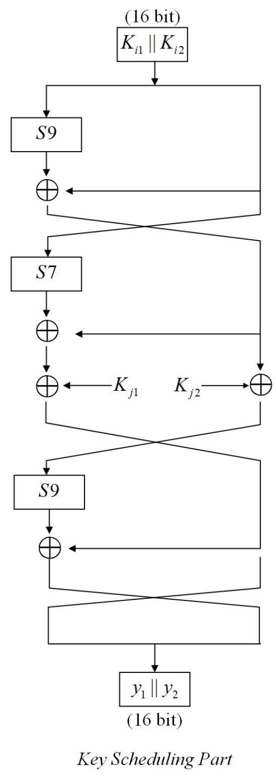

Figure 2 Key scheduling part

In the software implementation of MISTY1, S9 has $2^9$ entries of size 2 bytes, so its table requires 1 KB of memory. S7 has $2^7$ entries of size 1 byte, so its table requires 128 bytes of memory. Assume that the size of the cache line is $2^k$ bytes and the adversary can execute a 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 accesses the first S9 and S7. Moreover, the victim accesses the second S9, so the adversary can obtain the first $10-k$ bits of $y = S9(K_{i1}) \oplus ZE(K_{i2}) \oplus K_{j2}$.

After key scheduling is 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 execute the cache attack on S9 and S7, the cache trace of every access of S9 and S7 can be recorded 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 does not need to know the specific input and output of each *FI* function. Let the input of the *i*-th call of the *FI* function be $x_i(=x_{i1} \| x_{i2})$, the input of the second S9 $z_i$, and the round key $K'_{n_i}$. Then the *i*-th call of the *FI* function is shown in Figure 3.

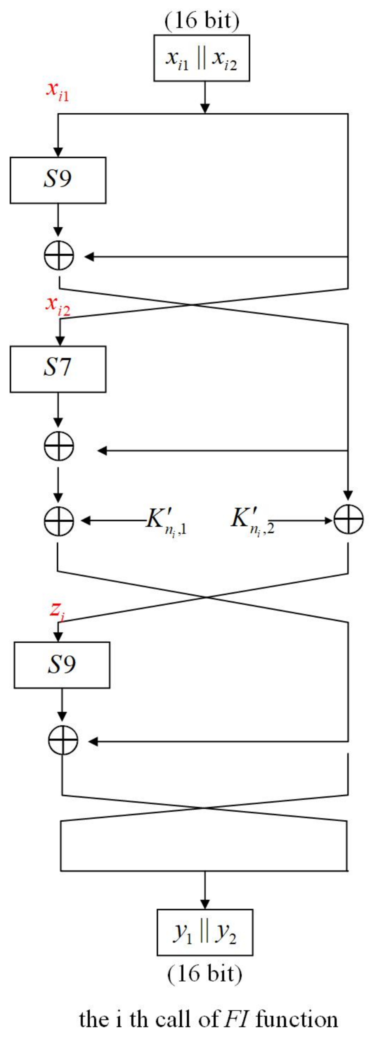

Figure 3 $i$-th call of $FI$ function

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

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

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, i.e., $Y = \{S9(t) \mid t[0:9-k] = x_{i1}[0:9-k]\}$. From 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,

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

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 the encryption of MISTY1 can eliminate some wrong values. We can accurately determine $K'_{n_i,2}[0:8-k]$ after observing 5–10 encryptions. Since the generation of $K'_{n_i}$ is related to $K_{n_i}$ and $K_{n_i+1}$, it will leak their information, which can help the adversary recover the key.

## 4 Attack scheme for 32-byte cache line

### 4.1 Information Leakage

According to Section 2.3, for each cache attack on S9, the adversary can recover the first five bits of the input, and for each cache attack on S7, the first two bits of the input.

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

### 4.2 Baseline attack in 32-byte cache line

Assume that the adversary knows a plaintext and ciphertext $(m,c)$ encrypted by the correct key of MISTY1. According to the leakage of the key scheduling part, we propose the following algorithm for a baseline attack on MISTY1 for a 32-byte cache line:

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

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

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

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

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

Algorithm 1 describes the baseline attack for a 32-byte cache line.

**Algorithm 1** Baseline attack on MISTY1 in 32-byte cache line

```
input: $K_i[0:4]$, $K_i[9:10]$, $y_i[0:4]$, $0 \le i \le 7$, 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
3:       continue
4:   else
5:       $K_1[7:8] = S9(K_0[0:8])[0:1] \oplus y_0[0:1]$
6:       $K_{0,11} \oplus K_{1,11} = S9(K_0[0:8])_4 \oplus y_{0,4}$
7:       for $i = 1$ upto 7 do
8:           for $K_i[5:6] = 2^2 - 1$ downto 0 do
9:               If $S9(K_i[0:8])[2:3] \oplus K_i[9:10] \oplus K_{i+1}[9:10] \ne y_i[2:3]$ then
10:                  continue
11:              else
12:                  $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$
13:                  $K_{i,11} \oplus K_{i+1,11} = S9(K_i[0:8])_4 \oplus y_{i,4}$
14:              end if
15:          If sum($K_{0,11} \oplus K_{1,11}$, $K_{1,11} \oplus K_{2,11}, \cdots, K_{7,11} \oplus K_{0,11}$) $\ne 0$ then
16:              continue
17:          else
18:              for $j = 0$ upto 7 do
19:                  for $K_i[12:15] = 2^4$ -1 downto 0
20:                      If $\text{MISTY1}(m, K) = c$ then
```

```
21:                 return $K$
22:             else
23:                 continue
24:             end if
25:         end for
26:     end for
27:   end if
28: end for
```

**Complexity of Algorithm 1:** S-box is a one-to-one mapping. If we know the two bits of S9 output, there are 1/4 of all S9 inputs making the output equate with the known two bits.

In step 1, we should first exhaust four bits of $K_0[5:8]$. We already know $y_0[2:3]$, $K_0[9:10]$ and $K_1[9:10]$. Then, we can calculate $S9(K_0[0:8])[2:3] = y_0[2:3] \oplus K_0[9:10] \oplus K_1[9:10]$. These two bits of $S9(K_0[0:8])$ making only 1/4 values go to the next loop. So, the complexity of step 1 is $O(2^{4-2})$.

In step 2, we should exhaust two bits of $K_i[5:6](1 \le i \le 7)$, and $S9(K_i[0:8])[2:3] = y_i[2:3] \oplus K_i[9:10] \oplus K_j[9:10]$ 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 one bit of $K_{0,11}$, so its complexity is $O(2)$.

In step 4, we should exhaust four bits of $K_i[12:15](0 \le i \le 7)$, so its complexity is $O(2^{4\times8})$.

Therefore, the total complexity of Algorithm 1 is $O(2^{4-2+7\times(2-2)+1+4\times8}) = O(2^{35})$. A baseline attack can be used to recover the key when the adversary observes only one encryption. If the adversary can observe multiple encryptions, the leakage of $K'_{n_i,2}[0:3]$ can be used 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*: Exhaust four bits of $K_0[12:15]$. Then recursively exhaust

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

**Algorithm 2** Improved attack on MISTY1 in 32-byte cache line

```
input: $K_i[0:4]$, $K_i[9:10]$, $y_i[0:4]$, $0 \le i \le 7$, 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
3:       continue
4:   else
5:       $K_1[7:8] = S9(K_0[0:8])[0:1] \oplus y_0[0:1]$
6:       $K_{0,11} \oplus K_{1,11} = S9(K_0[0:8])_4 \oplus y_{0,4}$
7:       for $i = 1$ upto 7 do
8:           for $K_i[5:6] = 2^2 - 1$ downto 0 do
9:               If $S9(K_i[0:8])[2:3] \oplus K_i[9:10] \oplus K_{i+1}[9:10] \ne y_i[2:3]$ then
10:                  continue
11:              else
12:                  $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$
13:                  $K_{i,11} \oplus K_{i+1,11} = S9(K_i[0:8])_4 \oplus y_{i,4}$
14:              end if
15:          If sum$(K_{0,11} \oplus K_{1,11}, K_{1,11} \oplus K_{2,11}, \cdots, K_{7,11} \oplus K_{0,11}) \ne 0$ then
16:              continue
17:          else
18:              for $K_0[12:15] = 2^4$ -1 downto 0
19:              for $j = 1$ upto 7 do
20:                  for $K_i[12:15] = 2^4$ -1 downto 0
21:                      If $FI(K_{j-1}, K_j)[8:11] \ne K'_j[0:3]$ then
22:                          continue
23:                      else if MISTY1$(m, K) = c$ then
24:                          continue
25:                      else
```

```
26:                         return $K$
27:                     end if
28:             end for
29:     end for
30:   end if
31:end for
```

**Complexity of Algorithm 2:** In step 4*, we should first exhaust four bits of $K_0[12:15]$, then recursively exhaust four bits of $K_i[12:15](1 \le i \le 7)$. We have known $FI(K_i, K_{i+1})[8:11]$, which contains four bits of known information. By program verification, we find that if $K_i$ is a fixed value, $FI_{K_i}(K_{i+1})$ is a one-to-one mapping about $K_{i+1}$. If we know the four bits of $FI(K_i, K_{i+1})$, there are $1/2^4 = 1/16$ of $K_{i+1}$ making the output equate with the known four bits. So, the complexity of step 4* is $O(2^{4+7\times(4\text{-}4)})$.

Hence 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 Section 2.3, for each cache attack on S9, the adversary can recover the first four bits of the input, and for each cache attack on S7, the first one bit.

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

### 5.2 Baseline attack in 64-byte cache line

Assume that the adversary knows plaintext and ciphertext $(m, c)$ encrypted by the correct key of MISTY1. According to the leakage of the key scheduling part, we propose a baseline attack on MISTY1 for a 64-byte cache line:

(0) By the cache attack, the adversary already knows $K_i[0:3](0 \le i \le 7)$,

$K_{i,9}(0 \le i \le 7)$ 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 five 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, then $y_{0,2} = S9(K_0[0:8])_2 \oplus K_{0,9} \oplus K_{1,9}$, and we continue the loop; if it is satisfied, then 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 $y_{i,2} = S9(K_i[0:8])_2 \oplus K_{i,9} \oplus K_{i+1,9}$. If not, then continue the loop; if it is satisfied, then calculate $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$ and $K_{i,10} \oplus K_{i+1,10} = S9(K_i[0:8])_3 \oplus y_{i,3}$, and proceed to (3);

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

(4) Exhaust $K_i[11:15](0 \le i \le 7)$ and determine whether $\text{MISTY1}(m, K) = c$ holds. If it does, then output the correct key $K$; if not, then continue the loop.

Algorithm 3 represents the baseline attack for a 64-byte cache line.

**Algorithm 3** Baseline attack on MISTY1 in 64-byte cache line

```
input: $K_i[0:3]$, $K_{i,9}$, $y_i[0:3]$, $0 \le i \le 7$, encryption algorithm MISTY1
output: $K = K_0 \| K_1 \| K_2 \| K_3 \| K_4 \| K_5 \| K_6 \| K_7$
1: for $K_0[4:8] = 2^5 - 1$ downto 0 do
2:   If $S9(K_0[0:8])_3 \oplus K_{0,9} \oplus K_{1,9} \ne y_{0,2}$ then
3:       continue
4:   else
5:       $K_1[7:8] = S9(K_0[0:8])[0:1] \oplus y_0[0:1]$
6:       $K_{0,10} \oplus K_{1,10} = S9(K_0[0:8])_3 \oplus y_{0,3}$
7:       for $i = 1$ upto 7 do
8:           for $K_i[4:6] = 2^3 - 1$ downto 0 do
9:               If $S9(K_i[0:8])_2 \oplus K_{i,9} \oplus K_{i+1,9} \ne y_{i,2}$ then
```

```
10:                     continue
11:                 else
12:                     $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$
13:                     $K_{i,10} \oplus K_{i+1,10} = S9(K_i[0:8])_3 \oplus y_{i,3}$
14:                 end if
15:             If $\mathrm{sum}(K_{0,10} \oplus K_{1,10},\ K_{1,10} \oplus K_{2,10}, \cdots, K_{7,10} \oplus K_{0,10}) \neq 0$ then
16:                 continue
17:             else
18:                 for $j = 0$ upto 7 do
19:                     for $K_i[11:15] = 2^5$ -1 downto 0
20:                         If $\mathrm{MISTY1}(m, K) = c$ then
21:                             return $K$
22:                         else
23:                             continue
24:                         end if
25:                 end for
26:         end for
27:     end if
28: end for
```

**Complexity for Algorithm 3:** In step 1, we should first exhaust five bits of $K_0[4:8]$. We already know $y_{0,2}$, $K_{0,9}$ and $K_{1,9}$. Then, we can calculate $S9(K_0[0:8])_2 = y_{0,2} \oplus K_{0,9} \oplus K_{1,9}$. S-box is a one-to-one mapping, these one bits of $S9(K_0[0:8])$ making only 1/2 values go to the next loop.. So, the complexity of step 1 is $O(2^{5-1})$.

In step 2, we should exhaust three bits of $K_i[4:6] (1 \le i \le 7)$. $S9(K_i[0:8])_2 = y_{i,2} \oplus K_{i,9} \oplus K_{i+1,9}$ contains one bit of known information, making only 1/2 values go to the next loop. So, the complexity of step 2 is $O(2^{7\times(3-1)})$.

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

In step 4, we should exhaust five bits of $K_i[11:15] (0 \le i \le 7)$, so its complexity is $O(2^{5\times 8})$.

Therefore, the total complexity of Algorithm 1 is $O(2^{5-1+7\times(3-1)+1+5\times 8}) = O(2^{59})$. A baseline attack can be used to recover the key when the adversary observes only one-time encryption. If the adversary can observe multiple encryptions, the leakage of $K'_{n_i,2}[0:2]$ can be used 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 five bits of $K_0[11:15]$. Then recursively exhaust $K_i[11:15](1 \le i \le 7)$ and determine whether $FI(K_{i-1}, K_i)[8:10] = K'_{n_i,2}[0:2]$ holds at the same time. If not, then 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 32-byte cache line

```
input: $K_i[0:3]$, $K_{i,9}$, $y_i[0:3]$, $0 \le i \le 7$, encryption algorithm MISTY1
output: $K = K_0 \| K_1 \| K_2 \| K_3 \| K_4 \| K_5 \| K_6 \| K_7$
1: for $K_0[4:8] = 2^5 - 1$ downto 0 do
2:   If $S9(K_0[0:8])_3 \oplus K_{0,9} \oplus K_{1,9} \ne y_{0,2}$ then
3:       continue
4:   else
5:       $K_1[7:8] = S9(K_0[0:8])[0:1] \oplus y_0[0:1]$
6:       $K_{0,10} \oplus K_{1,10} = S9(K_0[0:8])_3 \oplus y_{0,3}$
7:       for $i = 1$ upto 7 do
8:           for $K_i[4:6] = 2^3 - 1$ downto 0 do
9:               If $S9(K_i[0:8])_2 \oplus K_{i,9} \oplus K_{i+1,9} \ne y_{i,2}$ then
10:                  continue
11:              else
12:                  $K_{i+1}[7:8] = S9(K_i[0:8])[0:1] \oplus y_i[0:1]$
13:                  $K_{i,10} \oplus K_{i+1,10} = S9(K_i[0:8])_3 \oplus y_{i,3}$
14:              end if
15:          If sum($K_{0,10} \oplus K_{1,10}$, $K_{1,10} \oplus K_{2,10}, \cdots, K_{7,10} \oplus K_{0,10}) \ne 0$ then
16:              continue
17:          else
```

```
18:            for $K_0[11:15] = 2^5 - 1$ downto 0
19:             for $j = 1$ upto 7 do
20:                 for $K_i[11:15] = 2^5 - 1$ downto 0
21:                      If $FI(K_{j-1}, K_j)[8:10] \neq K'_j[0:2]$ then
22:                           continue
23:                      else if MISTY1$(m, K) = c$ then
24:                           continue
25:                      else
26:                           return $K$
27:                    end if
28:             end for
29:       end for
30:   end if
31:end for
```

**Complexity of Algorithm 4:** In step 4*, we should first exhaust four bits of $K_0[11:15]$, then recursively exhaust four bits of $K_i[11:15](1 \le i \le 7)$. We have known $FI(K_i, K_{i+1})[8:10]$, which contains three bits of known information. Given $FI_{K_i}(K_{i+1})$ is a one-to-one mapping about $K_{i+1}$, if we know the three bits of $FI(K_i, K_{i+1})$, there are $1/2^3 = 1/8$ of $K_{i+1}$ making the output equate with the known three bits. So, the complexity of step 4* is $O(2^{5+7\times(5\text{-}3)})$ .So, the complexity of step 4* is $O(2^{5+7\times(5\text{-}3)})$ .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 built a 32-byte cache line experiment in a gem5 simulation configuration [25] with architecture of x86-64 at 2.0 GHz, 32 KB L1-I/L1-D cache, 2 MB L2 cache and a 32-B cache line. We conducted 64-byte a cache line experiment on a Huawei Matebook 14 with Intel® Core™ i5-8265U CPU, 1.60 GHz, 256 KB L1-I/L1-D cache, 1 MB L2 cache and a 64-B cache line. The offline analysis program was written in Python and ran on a Huawei Matebook 14 laptop.

We get the code of MISTY1 from github[26]. Assume that the adversary has gotten a pair of plaintext and ciphertext encrypted by MISTY1. To obtain cache usage information, we use the Flush+Reload attack in Mastik Toolkit [27]. We used the key

0x00112233445566778899aabbccddeeff for our experiments.

## 6.1 Leakage of key scheduling part

### 6.1.1 32-byte cache line

Recall from Section 3 that in the 32-byte cache line environment, the size of S9 is 1 KB, so S9 occupies 32 cache lines. 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 corresponds to the first five bits of S9 input. Figure 4 shows the latency during Flush+Reload.

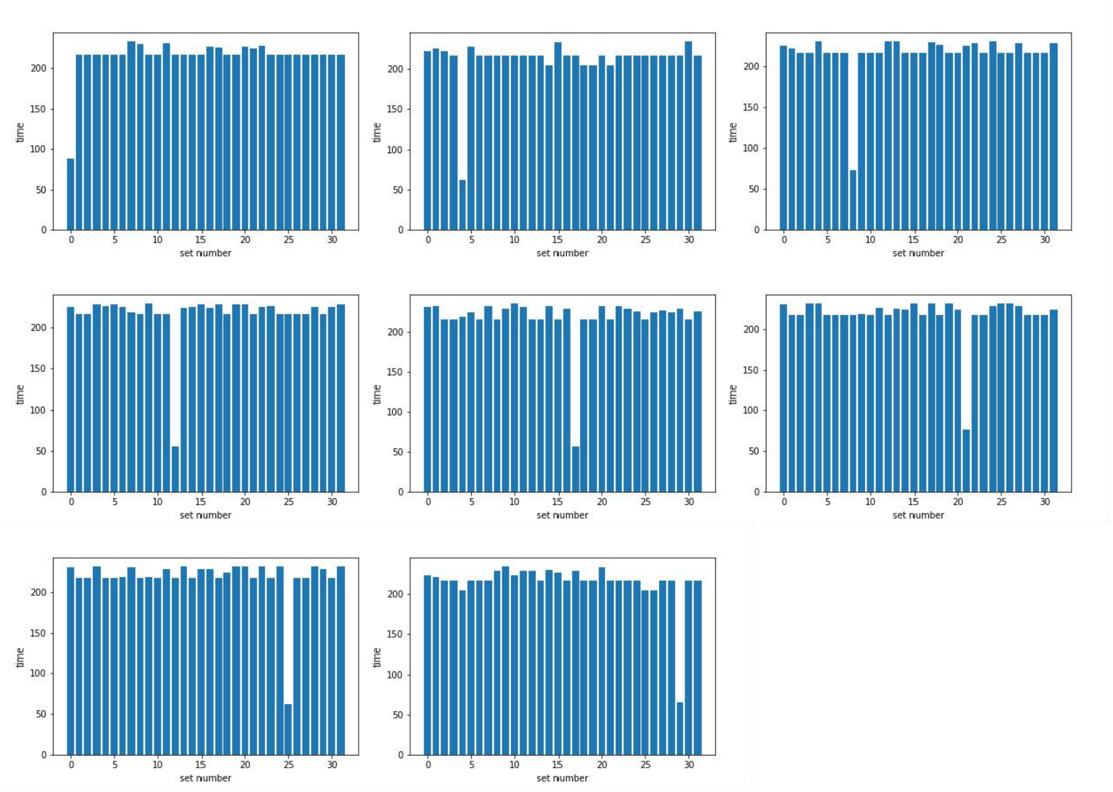

Figure 4 Leakage of $K_i[0:4]$

Recall from Section 4 that it leaks the first five bits of $K_i$: 0x0, 0x4, 0x8, 0xc, 0x11, 0x15, 0x19, 0x1d.

Recall from Section 3.1 that in the 32-byte cache line environment, the size of S7 is 128 B, so S7 occupies 4 cache lines. From the start to the end of S7, we perform Flush+Reload on the four cache lines in turn and record the latency. The shortest time corresponds to the first two bits of the S7 input. Figure 5 shows the latency during Flush+Reload.

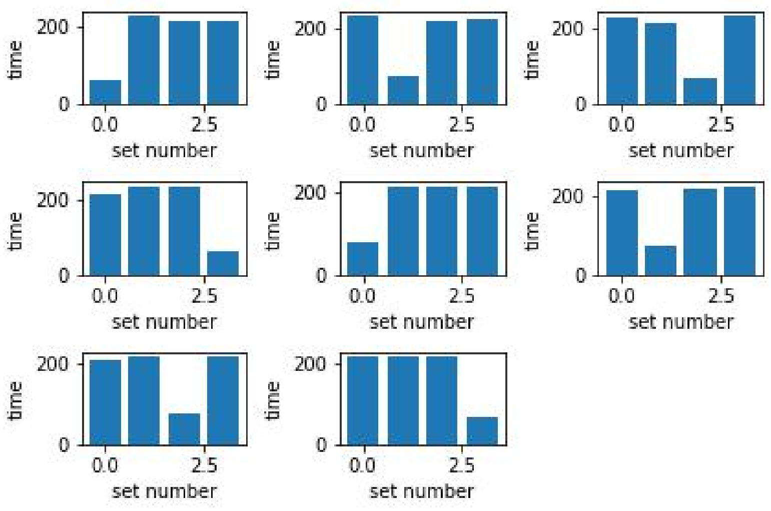
Figure 5 Leakage of $K_i[9:10]$

From Figure 5, we can know $K_i[9:10]$: 0,1,2,3,0,1,2,3.

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

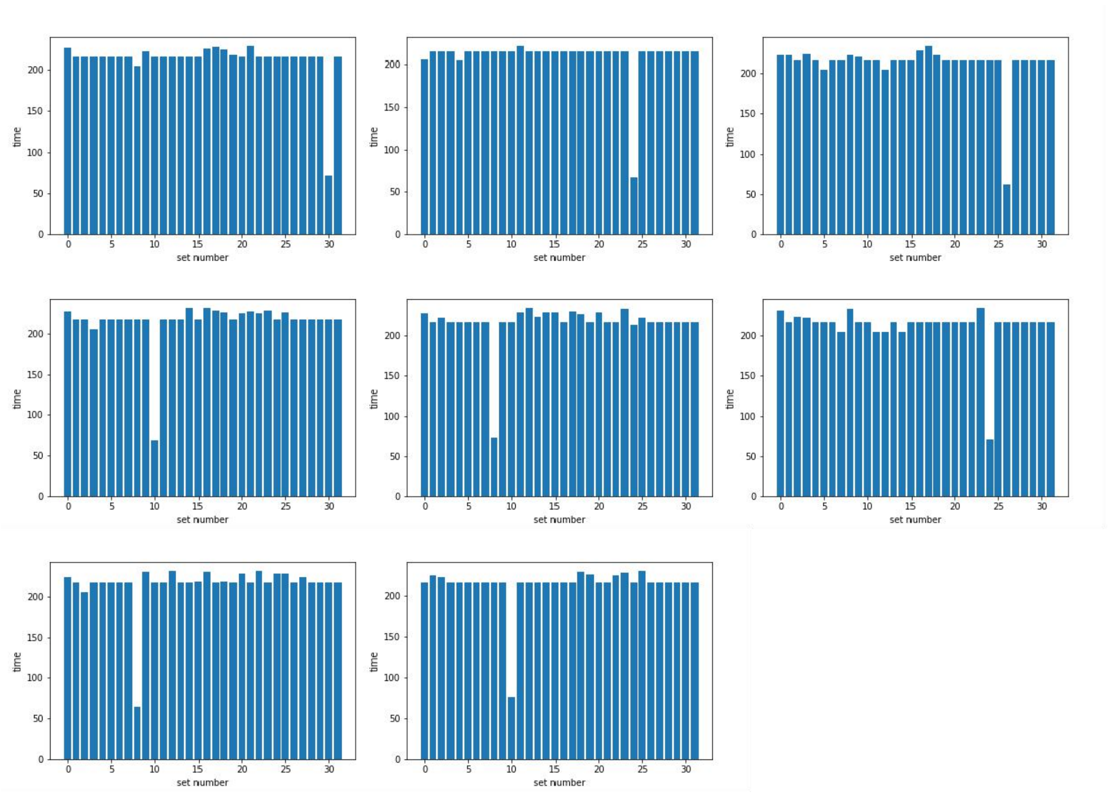
Figure 6 Leakage of $y_i[0:4]$

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

6.1.2 64-byte cache line

In the 64-byte cache line environment, S9 occupies 16 cache lines. 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 corresponds to the first four bits of the S9 input, as shown in Figure 7.

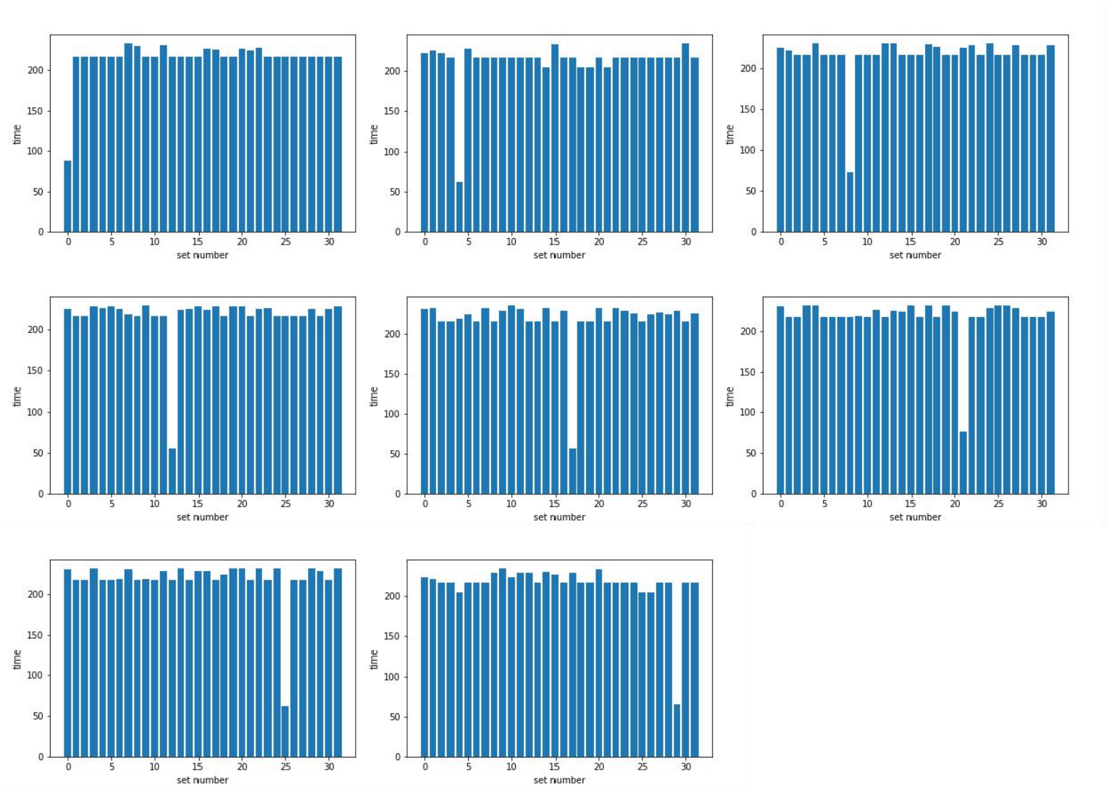

Figure 7 Leakage of $K_i[0:3]$

Recall from Section 5 that they leaks the first four bits of $K_i$: 0x0, 0x2, 0x4, 0x6, 0x8, 0xa, 0xc, 0xe.

In the 64-byte cache line environment, S7 occupies 2 cache lines. From the start to the end of S7, we perform Flush+Reload on the two cache lines in turn and record the latency. The shortest time corresponds to the first bit of the S7 input, as shown in Figure 8.

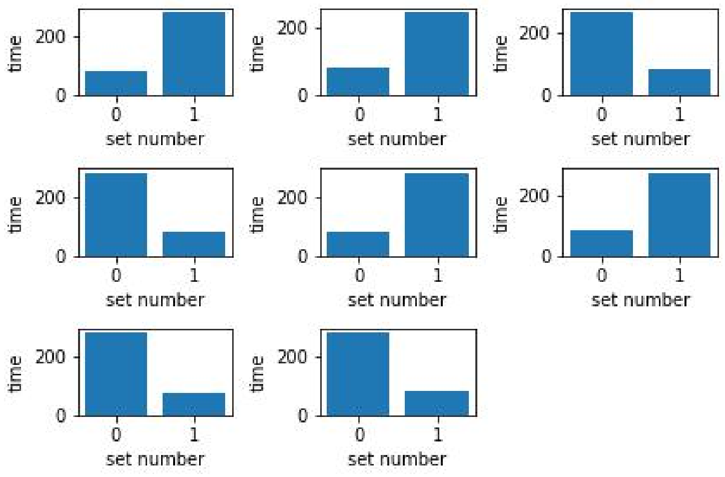

Figure 8 Leakage of $K_{i,9}$

From Figure 5, we can know $K_{i,9}$: 0,0,1,1,0,0,1,1.

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

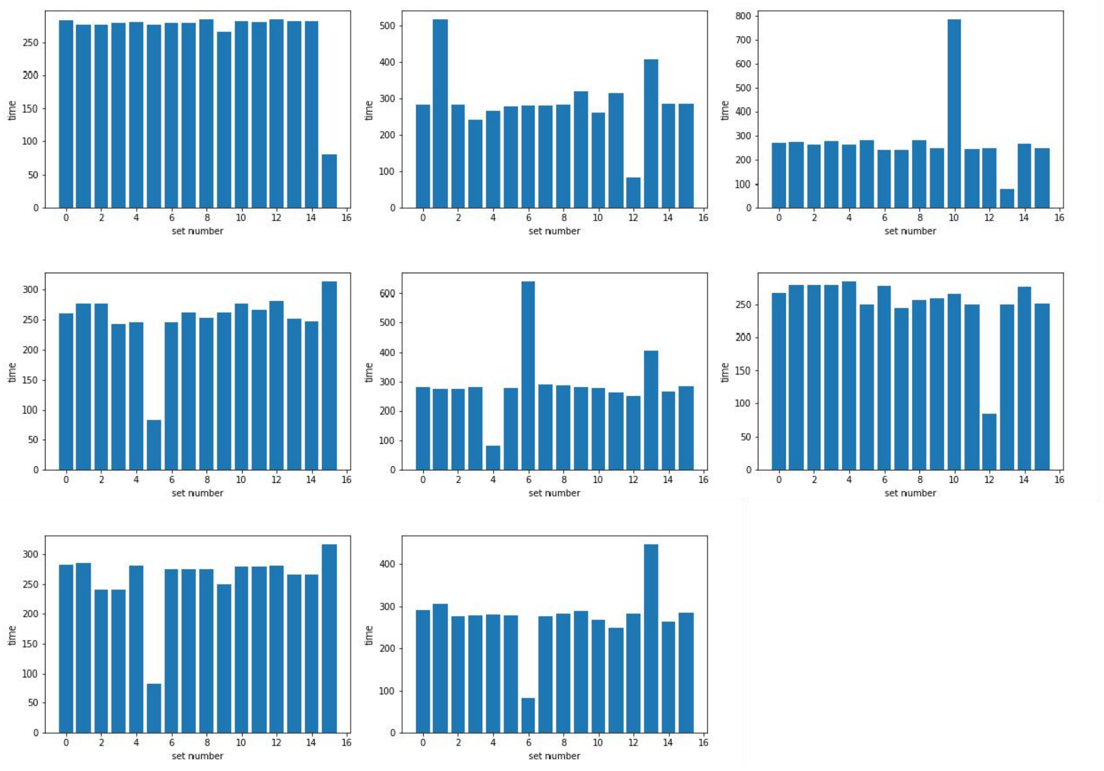

Figure 9 Leakage of $y_i[0:3]$

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

## 6.2 Confirm the leakage of $K_i'$

Recall from Section 3.2 that we can accurately determine $K_i'[0:8-k]$ after observing some encryption. This is vital for Algorithms 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.

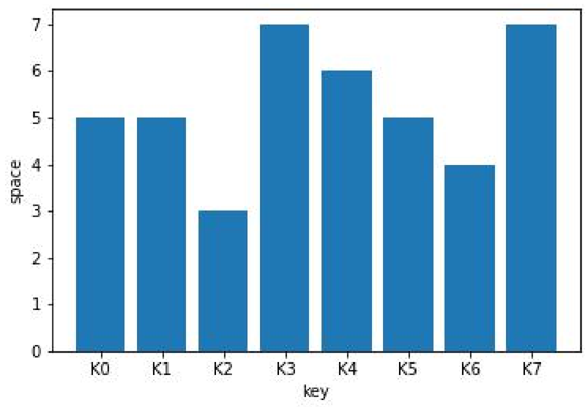

Figure 10 Space of $K_i'[0:3]$ after observing one encryption

After observing one encryption, the adversary can eliminate wrong guess, and drop the guess key space from $2^{32}$ to $2^{18.75} \approx (5\times5\times3\times7\times6\times5\times4\times7)$.

After observing five encryption, the adversary can determine the value of $K'_{n_i,2}[0:3]$, as Figure 11 shows.

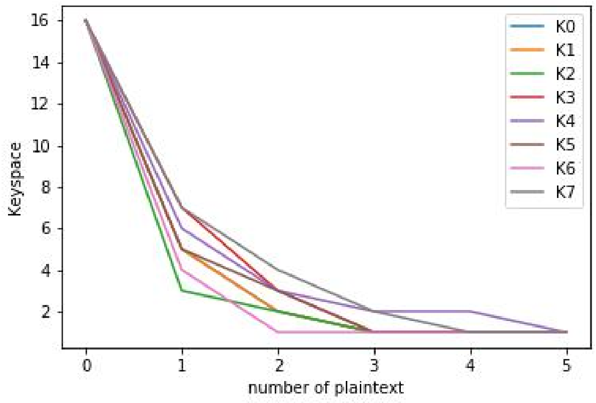

Figure 11 Determining $K_i'[0:3]$ after observing five encryption

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.

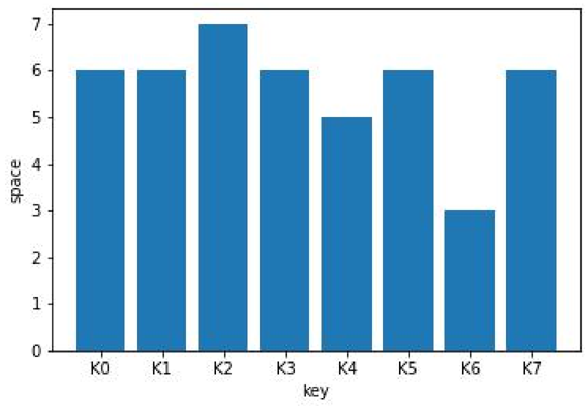

Figure 12 Space of $K_i'[0:2]$ after observing one encryption

After observing one encryption, the adversary could eliminate wrong guess, and drop the guess key space $2^{24}$ to $2^{19.64} \approx (6\times6\times7\times6\times5\times6\times3\times6)$.

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

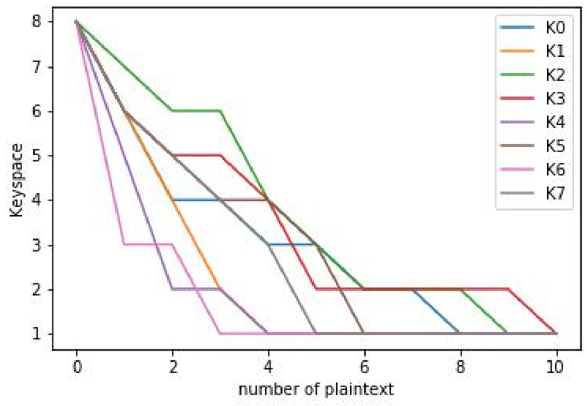

Figure 11 Determining $K_i'[0:2]$ after observing 10 encryption

## 6.3 Recover the whole 128-bit key

### 6.3.1 32-byte cache line

When more than five 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 0.035 s.

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

### 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 s (about 2.1 h).

Let us consider a worse case, assuming that the adversary only observes one-time encryption. Then the adversary can exhaust the $K_i'[0:2]$ space and use Algorithm 4 for the attack. The attacker needs $O(2^{57.64})$ time complexity at worst for the attack. Table 4 shows 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 encryptions | | $O(2^{7})$ | $O(2^{38})$ |

## 7 Discussion and Conclusion

**Countermeasures.** There are two basic ideas for defence: one, to completely eliminate cache leakage or the other, to eliminate the correlation between cache leakage and secret information.

In the first case, two effective approaches are to (i) use a constant-time program to prevent secret-related memory access [28] and (ii) tune the operating system so that it can preload certain data every time a process is activated [7]. 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 miss.

In the second case, the randomization-based approach [29] separates the memory line from the cache set, which may prevent the adversary from finding the cache set corresponding to the memory location. 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, and cannot initialize the particular set desired for use as a target in the tertiary cache, nor can the user determine if the set is used by the victim.

**Use of S9.** MISTY1 uses S9 to increase the security of the algorithm. However, since the output length of S9 is larger than eight bits, it must use two bytes of memory to store it; hence, S9 requires 1 KB of memory. This means the cache line size has to be larger than 1 KB to not generate leakage, which is difficult to achieve. To avoid cache attacks, a 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 disable the clflush command and use non-inclusive cache. Fortunately, many advanced cache attack techniques are proposed to support it [17, 30]. The adversary does not need to share any virtual memory with the victim, nor to share the same processor core. The cache attack can succeed even when the victim and adversary 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 [30]. When the adversary can observe the accesses to S9 and S7, our attack algorithm can be run to recover the whole key of MISTY1.

**Conclusion.** We implemented a Flush+Reload attack on the block cipher MISTY1 and recovered the 128-bit key in 5145.33 s (about 1.4 h) and 0.035 s in the 32-byte cache line environment when observing one and five encryption, respectively. Moreover, in the 64-byte environment, we observed 10 encryption to recover the 128-bit full key in 7661.06 s (about 2.1 h). Our work demonstrates that the application of S9 and the design of the key scheduling part make MISTY1 more vulnerable to cache attacks.

**Acknowledgments**

We thank LetPub (www.letpub.com) for its linguistic assistance during the preparation of this manuscript.

# 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: Efficiently 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] Binkert N, Beckmann B, Black G, et al. The gem5 simulator[J]. ACM SIGARCH computer architecture news, 2011, 39(2): 1-7.
[26] Julius C, Duque, Gitpan. Crypt-Misty1[OL]. https://hub.fastgit.org/gitpan/crypt-misty1, 2014.
[27] Yuval Yarom. Mastik: A Micro-Architectural Side-Channel Toolkit. http://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf. Sept. 2016.
[28] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. "The Security Impact of a New Cryptographic Library". In: LATINCRYPT. 2012, pp. 159–176.
[29] 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.
[30] 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.