Analysis on the Parameter Selection Method for FLUSH+RELOAD Based Cache Timing Attack on RSA

ZHOU Ping[1], WANG Tao[1], LI Guang[2], ZHANG Fan[3], ZHAO Xinjie[2]

[1] Department of Information Engineering, Ordnance Engineering College, Shijiazhuang 050003, China
[2] The Institute of North Electronic Equipment, Beijing 100083, China
[3] Department of Information Science & Electrical Engineering, Zhejiang University, Hangzhou, 310027, China

Abstract: FLUSH+RELOAD attack is recently proposed as a new type of Cache timing attacks. There are three essential factors in this attack, which are monitored instructions, threshold and waiting interval. However, existing literature seldom exploit how and why they could affect the system. This paper aims to study the impacts of these three parameters, and the method of how to choose optimal values. The complete rules for choosing the monitored instructions based on necessary and sufficient condition are proposed. How to select the optimal threshold based on Bayesian binary signal detection principal is also proposed. Meanwhile, the time sequence model of monitoring is constructed and the calculation of the optimal waiting interval is specified. Extensive experiments are conducted on RSA implemented with binary square-and-multiply algorithm. The results show that the average success rate of full RSA key recovery is 89.67%.

Keywords: side channel attack; Cache timing attack; RSA; square-multiply algorithm; exponentiation

I. INTRODUCTION

Side Channel attacks [1][2] are kind of attacks target on the implementation of cryptographic systems. Such attacks exploit the physical information such as time, sound, power consumption, electromagnetic radiation etc. leaked from the hardware platform during encryption and decryption to reveal the internal states of the algorithm, thus the knowledge of key can be inferred.

Cache timing attacks [3] are one category of side channel attacks. The time difference between Cache access and non-Cache access are considered as the leakage source. Typically, a spy process can be used to set the Cache to a known state and monitor the change of the state to gather the information on the Cache accesses of victim process. Since the proposal of Cache timing attack, it has been used to break various public-key cryptography and block cipher, such as RSA [4], DSA [6], ElGamal [14], AES [7][15][16][17].

Existing Cache timing attacks on RSA are divided into two categories. The first category is based on monitoring the specific instructions [5][8]. The dummy instructions of the spy process precisely maps to the same L1 Cache location with the specific instructions of victim process. In this way, the adversary creates a conflict between dummy instructions and the specific instructions. Therefore, the execution of the specific instructions by the victim can be detected. The other category is based on monitoring the entire L1 Cache [4][6]. The adversary can fill the entire L1 Cache with

dummy instructions to cause capacity miss, thus the operations executed by the victim can be inferred as operations have different Cache access patterns.

However, these attacks depend on that the spy process shares the same L1 Cache with the victim process. Therefore, it is only feasible for attacking RSA implemented on single-core processors. As to multi-core processors, the spy process and the victim process are likely to be assigned to different cores, thus L1 Cache can no longer be shared.

To address this issue, Yuval Yarom and Katrina Falkner [9] proposed a L3 Cache timing attack called FLUSH+RELOAD attack. The method relies on the fact that the spy process and the victim process share the same physical memory pages. The spy process will flush the specific instructions of the victim process in the shared memory page from all levels of the Cache, then reload them into Cache and measure the time. Thus, the access of the victim to the specific instructions can be monitored. FLUSH+RELOAD attack was applied to extract private keys from GnuPG and recovered 98% of the bits of the RSA private key.

Then, Yarom and Benger [18] applied this method in the case of OpenSSL's implementation of ECDSA over binary fields. Naomi Benger et al. [19] combined the FLUSH+RELOAD attack and the lattice techniques to attack the wNAF point multiplication algorithm and reconstructed secret keys for 256-bit elliptic curves after obtaining less than 256 signatures.

However, existing studies seldom analyzed the impacts of the monitored instructions, the waiting interval and the threshold. How to find the optimal value of these parameters is not carefully discussed either. Based on [9], this paper provides detailed analysis for these previously mentioned three issues as well as experimental verification.

Firstly, we analyze the relationship between the instruction access of the victim process and the execution of operations. Then, we propose the rules of selecting monitored instructions based on the necessary and sufficient conditions. In addition, the code of exponentiation of OpenSSL library has been analyzed and proper monitored instructions are selected.

The threshold selection method based on Bayesian binary signal detection principal is also proposed. By calculating the minimum cost of incorrect determination, the optimal threshold can be found and its value is calculated. Our experiments show that, with the optimal threshold value, the success rate of the attacks can be improved and the rate of incorrect estimation of operation is decreased.

Meanwhile, we provide the time sequence model of monitoring and the formula used to calculate the waiting interval time. Experiment results show that the capture rate of the victim access is above 62.82% when the interval is within the calculated theoretical value.

With the selected optimal parameters from the proposed method, extensive experiments are conducted on the RSA decryption implemented with square-multiply algorithm. The results show that the success rate of the proposed attack is between 86% and 95%, while the success rate is 64% with the value of parameters used in [9].

The remainder of the paper is organized as follows: Section II provides background information of RSA. Section III reviews and discusses the typical FLUSH+RELOAD attack. Section IV presents our methods for parameter selection. Section V provides the experimental results and discussions. Finally, Section VI concludes the full paper.

This paper aims to study the impacts of three parameters-monitored instructions, threshold and waiting interval, and the method of how to choose optimal values. The complete rules for choosing the monitored instructions based on necessary and sufficient condition are proposed.

II. RSA Encryption and ITS Implementation

2.1 RSA encryption

RSA [13] is the most widely used public-key cryptosystem. It can be used for encryption and digital signature. The steps of key generation for RSA public-key encryption is as follows:

- Generate two large random primes p and q.
- Calculate $N = pq$ and $\varphi = (p\text{-}1)(q\text{-}1)$.
- Select a random integer e, $1 < e < \varphi$, such

Algorithm 1 Square-and-multiply algorithm

Input: base x, modulus N, exponent $e = (e_t e_{t-1} \ldots e_1 e_0)_2$, $e_t = 1$.
Output: $y = x^e \bmod N$
Compute:
$y = 1$
for i from t to 0 {
  y = Square(y)
  y = ModReduce(y, N)
  if $e_i = 1$, then{
    y = Mult(y, x)
    y = ModReduce(y, N)
    }
}
return(y)

that gcd $(e, \varphi) = 1$.

- Calculate the unique integer d, $1 < d < \varphi$, such that $ed \equiv 1 \pmod{\varphi}$
- The public key is (e, N) and the private key is d.

The integers e and d above are called encryption exponent and the decryption exponent respectively and N is called the modulus. Let m denote the message, c represents the cipher text. To encrypt a message m, the sender should do the following:

- Represent the message as integer m in the interval $[0, n-1]$.
- Compute $c = m^e \bmod N$.

To recover plaintext m from c, the receiver should do the following:

- Compute $m = c^d \bmod N$

2.2 Square-and-multiply algorithm

The most important arithmetic operation for RSA is modular exponentiation. Square-and-multiply algorithm is a basic method for binary exponentiation. The exponent is firstly represented in a binary basis: $e = \sum_{i=0}^{t} 2^i * e_i$. Then a modular exponentiation is performed by scanning the bits of the exponent. The exponent bits can be scanned from the LSB to the MSB, or from the opposite way. These methods are called "right-to-left" and "left-to-right" exponentiation. Without loss of generality, we focus on the second case in this paper.

In the "left-to-right" method, each iteration begins with the execution of a square and a modulo reduce calculation. A multiplication and a modulo reduce calculation can follow them, depending on the current exponent bit value. This method is detailed in Algorithm 1.

We can see that the operations executed depend on exponent bits in the binary square multiplication algorithm. One square operation and one reduce operation will be executed when the current exponent bit is 0. Extra multiply and reduce operation will be executed when the current exponent bit is 1. Therefore, an adversary can exactly recover the exponent by tracing the execution of the square-and-multiply exponentiation algorithm.

III. FLUSH+RELOAD ATTACK

3.1 Attack principle

FLUSH+RELOAD is a recently proposed Cache attack targets on the implementation of public-key cryptosystem on multi-core processors.

It monitors and traces the accesses of a victim process to instructions in shared memory pages by a spy process. It is possible for the spy process to obtain the knowledge of executing instructions by a victim process, while the spy process only needs read access to the shared memory pages. The spy process can infer the behavior of the victim process by monitoring whether a specific instruction (or memory line) in these pages is accessed by the victim process.

There are three phases in each round of monitoring.

Phase I: the spy process flushes the monitored instructions from all levels of the Cache using the *clflush* instruction.

Phase II: the spy process waits for some time interval. During this moment, the victim process may load monitored instructions in shared memory pages.

Phase III: the spy process reloads the monitored instructions and measures the time it takes.

If the victim process did not access the

monitored instructions during the second phase, the monitored instructions should be reloaded form the memory thus it takes relatively longer time. Otherwise, it takes significantly shorter time to reload these instructions as they have been cached. Executing these three steps iteratively, the spy process can learn the sequence of monitored instructions performed by victim process.

3.2 Revisit to the attack

In FLUSH+RELOAD approach, three particularly important parameters are very important for the success rate of the attack. However, none of the existing literature discussed and analyzed how exactly these parameters worked and how to choose appropriate values for them.

The first issue is how to select the monitored instructions. The spy process infers the sequence of operations (square, multiply and modular reduction) performed during decryption by capturing accesses of victim process to the monitored instructions. Each monitored instruction should be corresponding to the operation it presents. Besides, they should be chosen so that the spy process can capture the victim access with sufficient high probability. Yuval Yarom and Katrina Falkner [9] stated "*to increase the chance of a probe capturing the access, we selected memory lines that are executed frequently during the calculation*". However, we find this principle cannot ensure the correspondence between the victim access to monitored instructions and the performing of operations. Improper selection may lead to incorrect operation determination.

The second issue is how to find the optimal threshold for the reload time. The threshold is used to compare with the reload time to determine whether the monitored instructions are reloaded from memory or Cache. The optimal threshold can decrease the rate of incorrect determination. The work in [9] set the threshold as 120 cycles. In our experiment, we find it is not optimal.

The third issue is how to choose the waiting interval. In Phase II, the monitored instructions may be accessed more than one time if the waiting interval is set too long. The spy cannot detect them accurately. However, if the waiting interval is too short, the spy will miss the access as the victim memory access overlaps the spy measurement. This is because that the reload time will be much longer than normal when overlap happens. The work in [9] set the waiting interval as 2048 cycles, but they did not explain the reason. Although it has been pointed out that "*choosing the length of the time slot presents a tradeoff between the attack resolution and the probability of missing a memory access*" [18], whether there exists upper or lower limit of waiting interval is still a question.

IV. Parameter Selection Method for Flush+Reload Attack

4.1 Rules for monitored instructions selection based on necessary and sufficient condition

In each round of the attack, the spy process monitors one instruction of each of the square, multiply and modulo reduce operations.

For convenience, let *addr* be the address of a monitored instruction. Let $addr_s$, $addr_m$ and $addr_r$ denote the monitored instruction of the square, multiply and modulo reduce operations respectively. A denotes the set of all the addresses of instructions of victim program. s, m, r and e present the square, multiply, modulo reduce and other operations, while S, M, R and E present the set of their addresses respectively. Let b be the size of a Cache block in byte. Obviously, we have $A = S \cup M \cup R \cup E$ and the right side of the equation may not be empty set.

The determination can be made from the measurement directly is whether the monitored instructions are reloaded from the Cache or the memory, i.e., the truth of proposition p:

Proposition p: $addr_i$ is cached during waiting interval. ($i = s, m, r$)

The attacker needs to infer the truth of proposition q from proposition p:

Proposition q: operation i is performed. ($i = s, m, r$)

According to the attack, both of the proposition p and proposition q must be true or false at the same time, i.e. if p then q and if $\tilde{}p$ then $\tilde{}q$. So, from this logical relationship, $addr_i$ should be selected such that p and q are sufficient and necessary condition of each other:

$$p \overset{addr_i}{\longleftrightarrow} q, \quad q \overset{addr_i}{\longleftrightarrow} p$$

To ensure that $addr_i$ is cached when operation i is performed during waiting interval ($q \rightarrow p$), $addr_i$ should be in accordance with the following rule:

Rule 1: $addr_i$ must be accessed by victim process during the execution of operation i:

$$addr_s \in S, \quad addr_m \in M, \quad addr_r \in R$$

To ensure that $addr_i$ is non-cached when operation i is not performed during waiting interval ($p \rightarrow q$), $addr_i$ should be in accordance with the following rules:

Rule 2: $addr_i$ should not be accessed by victim process during the execution of any operation except i:

$$addr_s \notin M \cup R \cup E, \quad addr_m \notin S \cup R \cup E,$$
$$addr_r \notin S \cup M \cup E$$

Rule 3: the execution of other operations should not lead to $addr_i$'s being cached indirectly:

$$[addr_s - 2b,\ addr_s + 2b] \notin M \cup R \cup E,$$
$$[addr_m - 2b,\ addr_m + 2b] \notin S \cup R \cup E,$$
$$[addr_r - 2b,\ addr_r + 2b] \notin S \cup M \cup E.$$

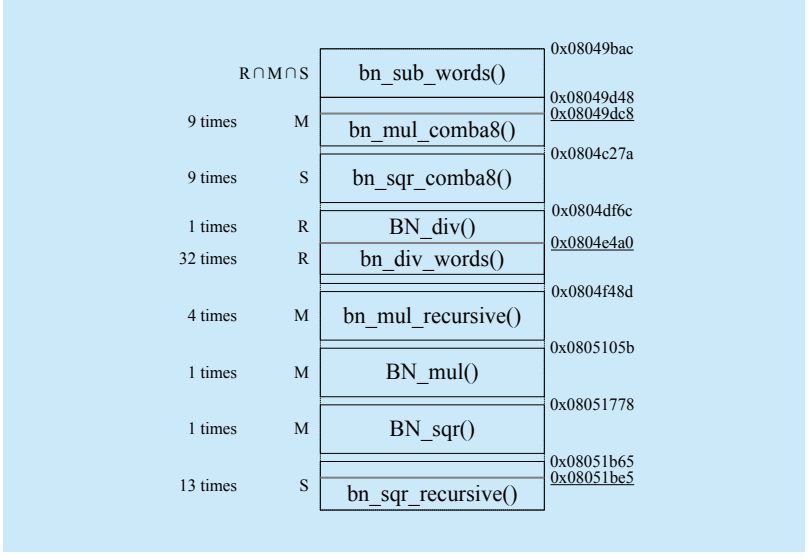

Fig.1 *Address allocation*

There are two reasons for $addr_i$ to be cached indirectly. The first is that the instructions and data are cached in blocks. If $addr_i$ and $addr_j$ are mapped into the same Cache block, then $addr_i$ will be loaded into Cache together with $addr_j$ due to the execution of operation j. Therefore, it should be ensuring that there are not any instructions of other operations in the range of [$addr_i$-b, $addr_i$+b]. The other reason is Cache prefetching. Some processors will prefetch a pair of 2b-byte Cache block if the first access to one of them while it is in memory. Therefore, no other operations' can be in the range of [$addr_i$-2b, $addr_i$+2b]. The former case should carefully be avoided while the latter one depends on the feature of the processor.

In addition, in order to increase the chance of capturing the access and improve the capacity of anti-noise, $addr_i$ should try to follow these rules:

Rule 4: $addr_i$ should be accessed more times than any address else.

$$n(addr_i) = \max n(addr),$$
$$addr \in \{x | p \overset{x}{\longleftrightarrow} q\}, \quad (i = s, m, r)$$

where $n(addr)$ denotes the time the $addr$ is accessed.

In our experiment, we implement the square-and-multiply algorithm with OpenSSL library. The Figure 1 shows the main functions involved in exponentiation and their first addresses.

On the left side of Figure 1, the set the instructions belong to and the accessed time in each operation are marked. According to the above rules, instructions located in $addr_s$=0x08051be5, $addr_m$=0x08049dc8, $addr_r$=0x0804e4a0 are selected as monitored instructions. Note that, according to Rules 3, $addr_s$ and $addr_m$ take 128 bytes, i.e. the size of two Cache blocks, away from the first addresses of the functions respectively. In addition, although the function bn_sub_words() is accessed much more than 32 times, its instructions cannot be selected according to Rule 2, as it is called during the execution of all of the three operations.

4.2 Time threshold selection based on binary signal bayesian test principle

The time threshold is used to determine if the monitored instruction is reloaded from Cache or the memory. If the loading time is greater than the threshold, it will be presumed that the instruction was loaded from the memory instead of the Cache. Therefore, one can infer that the victim process accessed the instruction before the reloading of the spy process.

Due to the interference of other processes and the random factors of the storage, the spy can only measure the reloading time with noises. The existence of noises may lead to two types of determination errors. The first is to mistake reloading from memory for reloading from Cache, resulting in wrong that the monitored instruction is accessed by the victim. The second is to mistake Cache reload for memory reload, thus the spy will miss the access of the victim process. Therefore, it is important to select appropriate threshold, to minimize the effect of false determination.

The effect of different thresholds on recovering the private key or operation sequence was not analyzed in [9]. The way to choose threshold in [9] is as follows: Firstly, the loading times from memory and L1 Cache were assessed primarily in the [33, 49] and [200, 300] intervals. Then, since the access time of L3 Cache is 22~39 cycles longer than access time of L1 Cache, "*the threshold is set to 120 cycles based on measured results and the Intel manual*".

4.2.1 Bayesian principles and average cost

In this paper, we adopt the binary signal Bayesian detection principle to select the threshold, which is a principle of statistical test for optimal signal status using minimal average cost as a criterion. In our scenario, reload has two states: the monitored instruction is reloaded from Cache, set as Hypothesis H_0; the monitored command is reloaded from memory, set as Hypothesis H_1. Then there are four possible determination results, denoted as $(H_i|H_j)$ $(i, j=0, 1)$, meaning that if H_j $(j=0, 1)$ is true, the result in which H_i $(i=0, 1)$ holds. Corresponding determination probability is expressed as $P(H_i|H_j)$ $(i, j=0, 1)$.

The formula for computing determination probability is:

$$P(H_i|H_j) = \int_{R_i} p(x|H_j)dx \quad i, j = 0, 1 \quad (4\text{-}1)$$

In this formula, $p(x|H_0)$ is the probability density function of measured reload times from Cache; $p(x|H_1)$ is the probability density function of measured reload times from memory. R_i indicates the two subspaces of observational space of reload times that are divided by threshold η. Specifically, when the measured value is smaller than threshold η it falls into space R_0, holds hypothesis H_0; when the measured value is greater than threshold η it falls into R_1, holds hypothesis H_1.

For the four determination scenarios, when $i=j$ the determination is correct, otherwise incorrect. Cost factor c_{ij} $(i, j=0, 1)$ is used to denote the cost of each determination scenario, meaning that if H_j $(j=0, 1)$ is true, the cost of determining that H_i $(i=0, 1)$ holds. The restrictions of cost factors are $c_{jj} \geq 0$, $j=0, 1$; $c_{ij} \geq c_{jj}$, $i, j=0, 1$, $i \neq j$.

Then, the average cost is

$$C = P(H_0)[c_{00}P(H_0|H_0)+c_{10}P(H_1|H_0)] + P(H_1)[c_{11}P(H_1|H_1)+c_{01}P(H_0|\mathrm{H}_1)] \quad (4\text{-}2)$$

According to Bayesian test principle, the optimal η minimizes the average cost C.

4.2.2 Cost factor

In our scenario, the effects of four determination results to the attack are different. When the determination is correct, there is no cost, thus $c_{00}=0$, $c_{11}=0$. When the determination is incorrect, then $c_{01}>>c_{10}>0$. It is because that in each operation, the monitored instruction will be executed multiple times (e.g., in computing the modulo reduce, the monitored instruction will be executed 32 times), detecting several times in them will suffice to help correctly determine the computation executed in the victim process. Therefore, the cost of $(H_1|H_0)$ determination is not significant. On the other hand, once the reloading form memory is mistakenly determined as from Cache, it will

cause incorrect speculation to be accessed by the victim. Even though errors are taken into consideration in the key recovery algorithm, the efficiency and success rate of private key recovery will be greatly reduced. Therefore, the cost of $(H_0|H_1)$ determination will be obviously higher than the cost of $(H_1|H_0)$ determination. In this paper we set c_{01}=100, c_{10}=10. Note that this selection of $c_{01}$and c_{10} is to reflect the comparative cost of two determination results. As long as the values satisfy that $c_{01}>c_{10}>0$, same results will be obtained in the subsequent analyses.

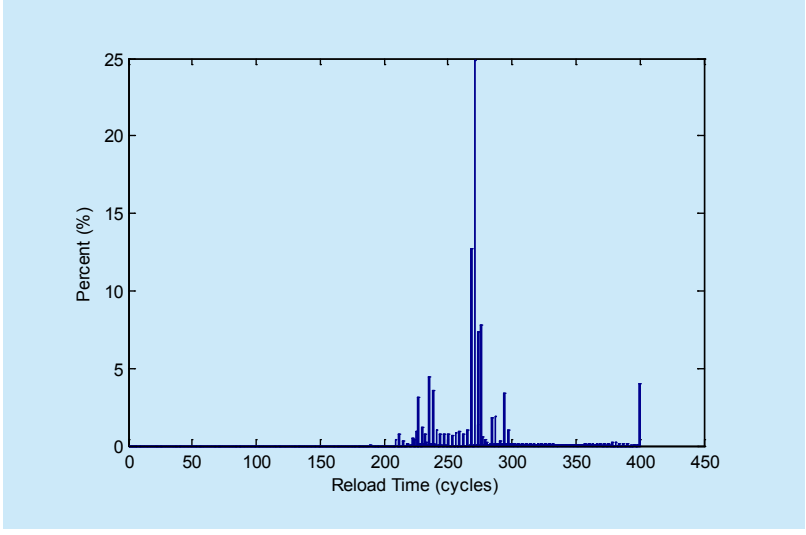

Fig.2 *Distribution of reload times from memory*

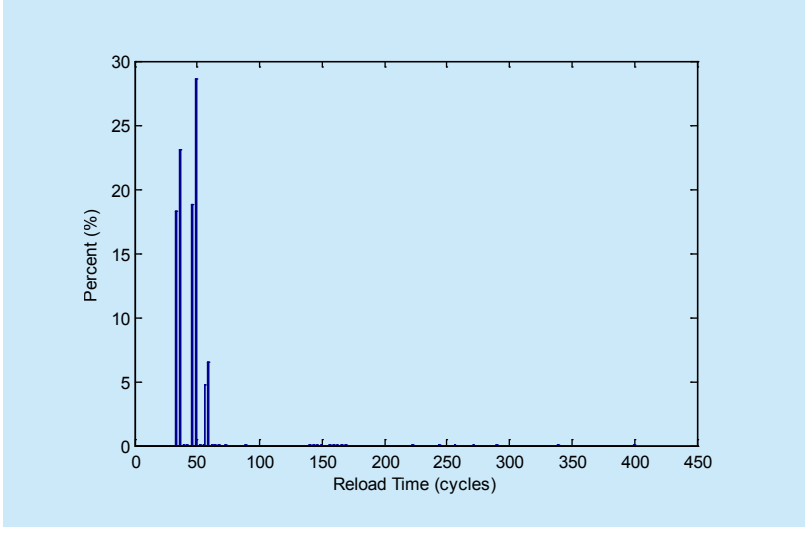

Fig.3 *Distribution of reload times from Cache*

4.2.3 Priori probability and probability density function

Since only one operation can be executed in a given moment for the victim process, only one of the three monitored instructions will be reloaded from Cache. Therefore, the priori probability of two determination hypotheses is $P(H_0)$=1/3 and $P(H_1)$=2/3, respectively.

We use the distribution of reload times from memory and Cache of monitored instructions as the probability density function for computing the average cost.

To measure the reload time of monitored instruction from the memory, this study did not execute the decryption but only the spy process. In this way, after each time the monitored instruction is loaded into Cache, it gets flushed out by the spy process, ensuring that the instruction will be reloaded from memory in the next time. 300,000 reload times from memory was measured, whose distribution is displayed in Figure 2.

In Figure 2, the horizontal axis and the vertical axis indicate the reload time and the percentage of sample in the overall sample size respectively. To show this figure in a proper scale, outliers are truncated. From Figure 2, we can see that all reload times from memory exceeded 189 cycles. In fact, most of the reload times from memory were greater than 206 cycles.

To obtain the reload time from Cache of monitored instruction, we removed the *clflush* instruction from the spy program. In this way, the monitored instruction will always be reloaded from Cache. 300,000 reload times from Cache was measured, whose distribution is shown in Figure 3.

From Figure 3, most reload times from Cache ranged within 33~59. However, some samples were distributed in 60~400 cycles, exceeding the minimum time of memory access. Moreover, the access time of different Cache level can also be observed from the Figure.

4.2.4 Calculation of optimal threshold

After determining the cost factors, the priori

probability of the hypotheses and the probability density function of reload times, the average cost is a function of threshold η. Using MATLAB, the average cost with different threshold is shown in Figure 4.

In Figure 4, the horizontal axis indicates threshold η. The vertical axis indicates average cost (for convenience of displaying, the results are logged). From Figure 4, it is clear to see that the optimal value of η is 170~189 cycles, and its average cost is minimal.

4.3 Time sequence model of monitoring and waiting time interval selection

In the spy process, each round of monitoring is divided into flushing, waiting, and reloading. At the beginning of the monitoring, the monitored instruction will be flushed out of Cache by the spy process. After waiting for a certain time interval, it is reloaded by the spy process and the reload time is measured. It can be decided that whether the victim process accessed the monitored instructions during the waiting interval by the length of reload time. The work in [9] selected the waiting time interval to be 2048 cycles. However, no explanations were provided.

In this paper, we believe that the effects of waiting interval on attack are two folds. If the waiting interval is relatively long, the monitored instruction may have been executed for multiple times, or multiple different monitor instructions have been executed during the same waiting interval, which cannot be distinguished by the spy process, leading to low "resolution". If the waiting interval is relatively short, the frequent flushing and reloading may be more likely to conflict with the access of victim process, leading to severe fluctuations of measurement, which subsequently affects capturing the victim access.

To investigate the range of waiting intervals, let t_{wait} be waiting interval, t_{probe} be the time used for each monitoring operation (including flushing, reloading, measuring, recording etc.). Let t_{rdtsc} denotes the time used for the timer to record time, t_{memory} and t_{L3_Cache} denote the reload time of instruction from memory and from L3 Cache respectively. Let ΔT_{target} denotes the execution time interval between two adjacent monitored instructions that are to be distinguished. All variables use cycles as their units. The time sequence model of monitoring is displayed in Figure 5.

For every specific monitored instruction, the time between flushing and the next reloading includes waiting interval t_{wait}, as well as the time for another two monitoring tasks $2*t_{probe}$, and the time cost t_{rdtsc} for timer instruction *rdtsc*. Based on the times sequence model of

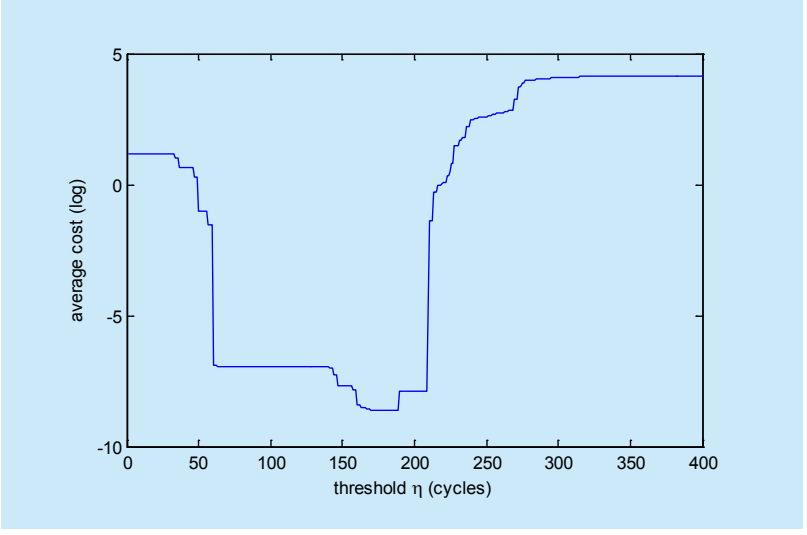

Fig.4 *Average cost with different thresholds*

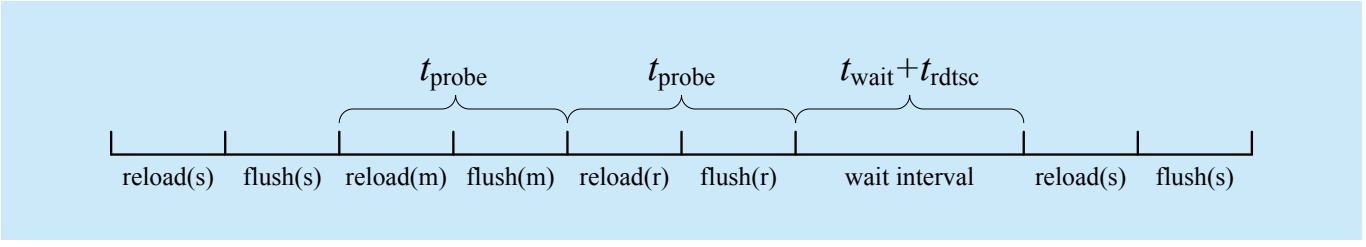

Fig.5 *Time sequence model of monitoring*

monitoring, waiting interval t_{wait} should satisfy:

$$t_{memory}\text{-}t_{L3\_Cache}<t_{wait}+2*t_{probe}+t_{rdtsc}<\Delta T_{target} \quad (4\text{-}3)$$

In the FLUSH+RELOAD attack, the adversary needs to distinguish whether the reload is from memory or Cache, the interval between two rounds of monitoring i.e., $t_{wait}+2*t_{probe}+t_{rdtsc}$, should at least be greater than the time needed for the monitored instruction to be loaded to L3 Cache from memory, i.e., $t_{memory}\text{-}t_{L3\_Cache}$. Otherwise, the spy process will start reloading and measuring before the victim process completes loading the monitored instruction from memory to L3 Cache. The conflict caused by different processes access the same physical address will bias the measured reload times.

Meanwhile, the interval between two rounds of monitoring should ensure that the access behaviors would not happen within the same waiting intervals. In our case, t_{wait} needs to be smaller than the time interval between two executions of each monitored instruction. In addition, it also needs to be smaller than the time interval between last execution of the monitored instruction in one computation and the first execution of monitored instruction in the next computation.

The average values of the aforementioned parameters and the t_{wait} range calculated from Formula 4-3 are shown in Table I.

Table I *Values of time parameter and boundary of the optimal* t_{wait}

Parameter	Average Value (cycles)	Boundary of the Optimal t_{wait} (cycles)
t_{memory}	285.31	
t_{L3_Cache}	57.74	
t_{probe}	512.31	
t_{rdtsc}	72.98	
ΔT_{target} (*s*)	2059.53	<961.93
ΔT_{target} (*m*)	3370.33	<2272.73
ΔT_{target} (*r*)	1476.04	<378.44
ΔT_{target} (between operations)	1670.40	<572.8

V. Experiments

5.1 Experiment setup

The experiment was conducted on the ThinkCentre desktop, which is characterized by an Intel i5-3470 processor, a 4GB DDR3-1600 memory and a Fedora 18 operating system. The RSA decryption employed the binary square-and-multiply algorithm. The square, multiplication and modulo reduce calculation were implemented by OpenSSL. Each decryption round was performed with the randomly generated 1024-bit modulus and key. The selected monitored instruction was the same as that in Section 4.1 and its address was obtained from the Eclipse debug tool.

In the attack, if more than two successive samples of reload times of $addr_i$ are less than the threshold, it is believed that the operation *i* is performed. A successful attack means that the 1024-bit is recovered entirely and exactly without exhausting search.

5.2 Selecting the threshold

In the experiment, we use the error rate of operation determination and the success rate of attack to evaluate the impact of the threshold and verify the conclusion in Section 4.2. The error rate of operation determination refers to the ratio between the number of wrong guessed operations and the total number of operations performed during a decryption round. The wrong guesses include the admittance of the operations that actually occur and the consideration of the operations that never occur.

In the experiment, the waiting interval was set as [0, 50, 100, 150, 200, 250, 300, 400, 600, 800, 1024, 1536, 2048, 3072, 4096] cycles. For each interval, we executed the attack for 100 times and recorded the timing data. Then the data processing and key recovery were conducted with different threshold values. The relationship among the error rate of operation determination, the success rate of attack and the threshold value are presented in Figure 6.

From Figure 6, we can see that in the range between 80 and 160 cycles, the error rate of operation determination decreases while the success rate of the attack goes up with increasing threshold η. The reason is that the probability of incorrect determination $P(H_1|H_0)$ decreased. Especially, the error rate exhibits

a sharp descent in the range between 120 and 160 cycles, while the success rate shows a significant increase in the range between 140 and 160 cycles. These properties result from the fact that plenty of the measured values of the reload times from Cache lie near to 150 cycles.

In addition, the error rate becomes more stable and ranges between 1.78% and 1.91% in the domain bounded by 180 and 200. It achieves its minimum value of 1.78% at η=180. The success rate of the attack stabilizes between 84.47% and 88.4% and it arrives its peak value of 88.4% at η=180. It is in the domain that both $P(H_1|H_0)$ and $P(H_0|H_1)$ are relatively small.

The phenomenon that the error rate turns into a sharp increase while the success rate of the attack descends dramatically is attributed to an increasing $P(H_0|H_1)$ after the η becomes greater than minimum value of reload times from memory.

Note that the threshold value η is evaluated in a relatively better domain between 160 and 200; the optimal value of the threshold is 180. These values are consistent with the conclusion in Section 4.2.

5.3 Selecting waiting interval

In this section, we use the standard deviation of measurements, the capture rate and the overlapping rate to evaluate the impact of the waiting interval and to verify the conclusion in Section 4.2. Specifically, the standard deviation of measurements is used to determine the lower boundary of the optimal waiting interval t_{wait}. The capture rate and the overlapping rate are used to determine the upper boundary of the optimal t_{wait}.

Normally the reload times are stable. If fluctuation happens, it means that the flushing and reloading are too frequent so that the accesses of victim to monitored instruction is disturbed. The standard deviations of measurements under different t_{wait} are shown in Figure 7.

It is clear to see that as waiting interval t_{wait} increases, the standard deviation of measurements decreases and then remains stable. Particularly, when t_{wait} is smaller than 200, the measured reload times are significantly unstable, which may lead to incorrect determination during data processing.

The capture rate refers the ratio between the number of captured victim accesses and the number of victim accesses in actual. During a long waiting interval, it is possible that the monitored instruction has been executed for more than one time and the capture rate decrease. Figure 8 shows the influences of different values of t_{wait} to the capture rates.

The vertical dashed lines denote the calculated boundary of the optimal t_{wait} according to

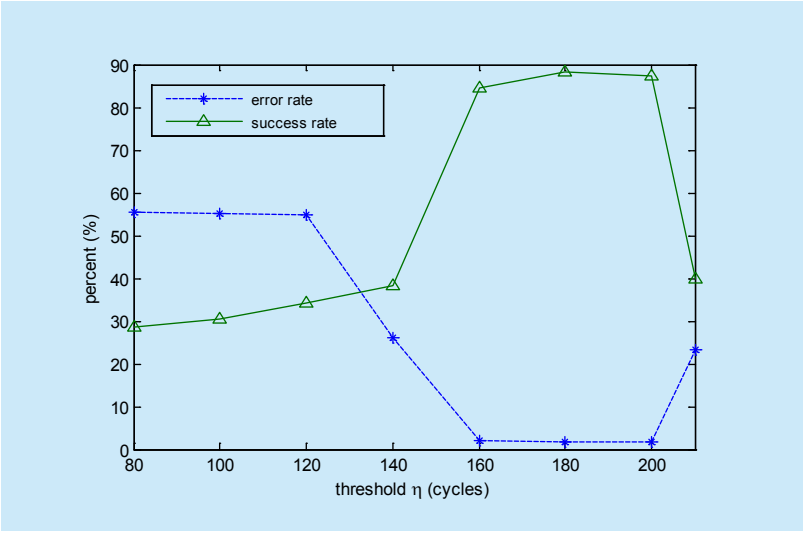

Fig.6 *Error rate and success rate of attacks with different thresholds*

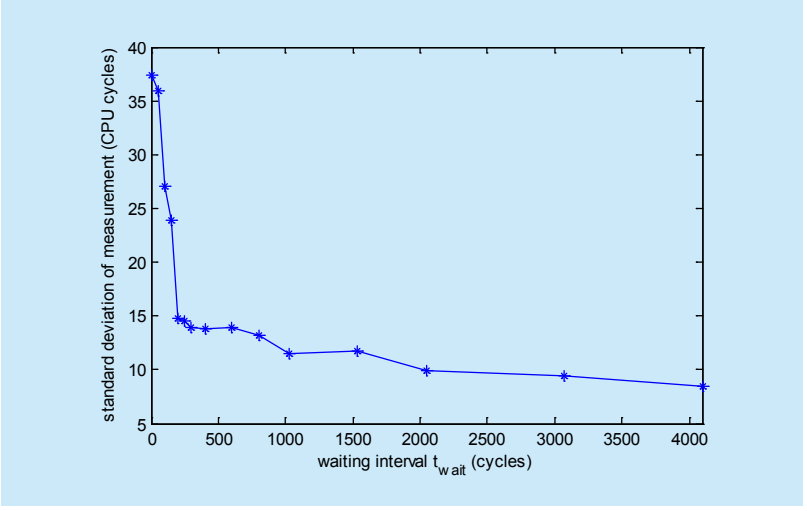

Fig.7 *Standard deviation of measurements with different waiting intervals*

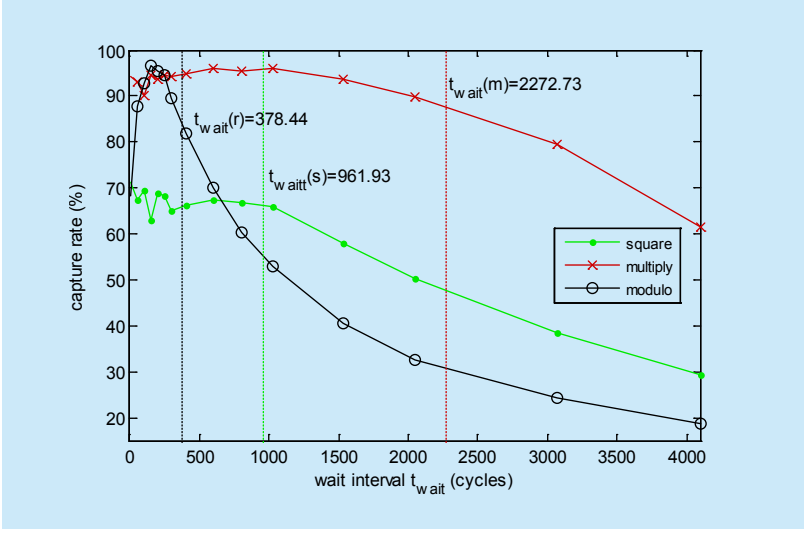

Fig.8 *Capture rate with different waiting intervals*

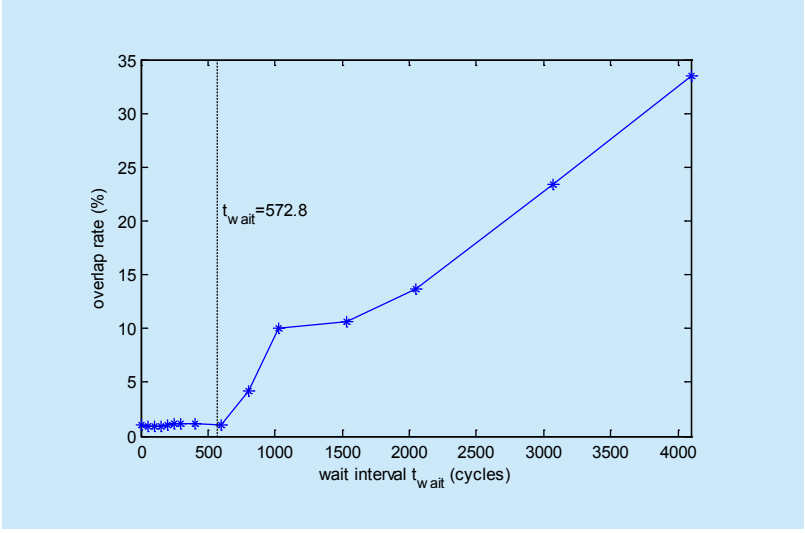

Fig.9 *Overlap rate with different waiting intervals*

Table II *Monitoring round corresponding to operations*

Number of the Monitoring Rounds	Operation
1317-1334	S
1337-1361	R
1361-1394	M
1394-1427	R
1428-1443	S
1447-1479	R

Formula 4-3. The overall capture rate decreases as t_{wait} increases. The capture rates decreases dramatically after the t_{wait} exceeding the calculated boundary of optimal values. Within the optimal value, the capture rates maintain above 62.82%.

In addition, a long waiting interval t_{wait}, will lead to overlap between different operations, which may cause difficulty in distinguish operations. The overlap rate is the ratio between the number of overlap and the number of operations performed during a decryption round. The relationship between t_{wait} and overlap rate is shown in Figure 9.

From Figure 9, when the waiting interval t_{wait}>600, the overlap rate increases dramatically. The experimental result is very close to the calculated ceiling of the optimal t_{wait}, i.e., <572.8.

Note that these results are consistent with the conclusion in Section 4.3. According to the experiments above, the following experiments will be carried out with setting the waiting interval as 200~600 cycles.

5.4 Attacking experiments

In the attack experiments, we set t_{wait}=200, η=180. Figure 10 presents a part of measurement times during one decryption operation.

In Fig. 10, the horizontal axis is the round of monitoring and the vertical axis is reload time, and the dotted line is the threshold. We can see that when the reload time is less than the threshold, the monitored instruction is reloaded from the Cache, which determines the type of operation performed by victim process at this moment.

Table II shows the monitoring round corresponding to each operation:

Note that in round 1361 modulo reduce calculation and multiplication overlap each other, and in round 1434 an incorrect determination is mixed in square operations, but the sequence of operations can still be recognized obviously. This is because that the selected monitored instruction is frequently executed in each operation so that the fault tolerance is improved.

According to experimental results in Section 5.2 and Section 5.3, we set 160<t_{wait}<200 and 200<η<600. For each (t_{wait}, η) we executed the attack for 100 times, and for each attack

we used randomly generated 1024-bit key. The success rates of attacks are shown in Table III:

With the optimal parameters, the success rates of attacks ranges between 86% and 95%, with an average of 89.67%, while the success rate of attack is 64% with the parameters chosen by [9].

VI. Conclusion

Three important issues for FLUSH+RELOAD attack have been analyzed in details in this paper, including the selections of the monitored instructions, the threshold and the waiting intervals. We provide the complete rules for selecting the monitored instructions based on necessary and sufficient condition. Following these rules can ensure that the reload times of monitored instructions reflect the real operations performed by the victim process. Based on Bayesian binary signal detection principle, we propose the method of choosing the optimal threshold. The experimental results indicate that the threshold should be smaller than but close to the lower boundary of loading times form memory. The time sequence model of monitoring was established and an approach to calculate the optimal waiting interval was provided. Generally, the shorter waiting interval can provide higher resolution. However, the measured reload times may be unstable if the waiting interval is too short. The method above is verified by experiments to recover private key of RSA decryption employing binary square-and-multiply algorithm. The average success rate reached 89.67%, which is better than 64% with the value of parameters used in [9].

It should be noted that the methods proposed in this paper can also be applied to various implementations of RSA or other public-key cryptography, by simply adjusting some details. For example, in the case of literature [18], the cost factor c_{01} should be set equal to c_{10} when calculating the optimal threshold, since that the costs of two kind of incorrect determinations are the same, which is quite different with our case.

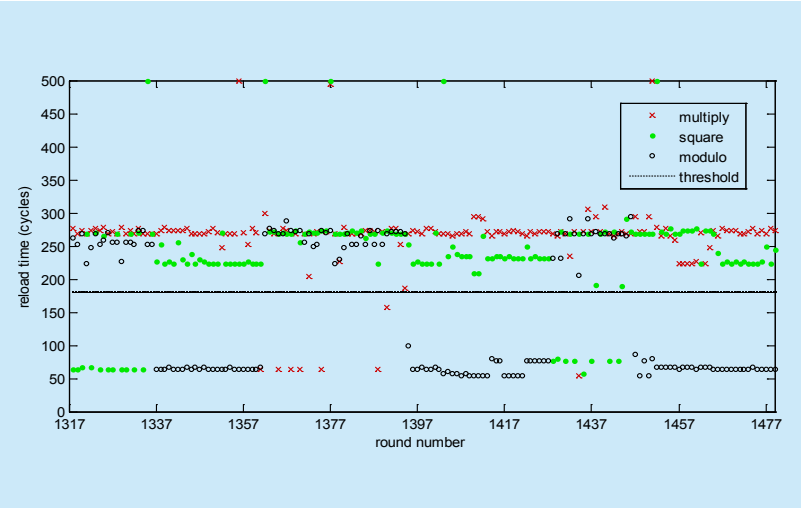

Fig.10 *Measurements of reload time*

Table III *Success rates of attacks with different waiting intervals and thresholds*

Waiting Interval /Threshold	200	180	160	120([9])
200	86%	87%	87%	
250	89%	88%	87%	
300	90%	91%	91%	
400	87%	89%	89%	
600	94%	95%	95%	
2048([9])				64%

Acknowledgements

The authors would like to thank the reviewers for their detailed reviews and constructive comments, which have helped improve the quality of this paper. This work is supported by National Natural Science Foundation of China (No. 61472357, No. 61309021, No. 61272491, No. 61173191), the Major State Basic Research Development Program (973 Plan) of China under the grant 2013CB338004.

References

[1] KOCHER P C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems[C]// Proceedings of CRYPTO 1996: August 1996, Santa Barbara, California, USA. Berlin: Springer, 1996: 104–113.

[2] KOCHER P, JAFFE J, JUN B. Differential power analysis[C]// Proceedings of CRYPTO 1999: August 1999, Santa Barbara, California, USA. Berlin: Springer, 1999: 388-397.

[3] PAGE D. Theoretical use of Cache memory as a cryptanalytic side-channel[R]. Technical Report

CSTR-02-003, Department of Computer Science, University of Bristol, 2002.
[4] PERCIVAL C. Cache missing for fun and profit[EB/OL]. <http://www.daemonology.net/papers/ htt.pdf>. 2005.
[5] ACIICMEZ O., SCHINDLER W. A Vulnerability in RSA Implementations Due to Instruction Cache Analysis and Its Demonstration on OpenSSL[C]// Proceedings of the CT-RSA 2008: April 2008, San Francisco, CA, USA. Berlin: Springer, 2008: 256–273.
[6] ACIIÇMEZ O, BRUMLEY B B, GRABHER P. New Results on Instruction Cache Attacks[C]// Proceedings of the CHES 2010: Santa Barbara, USA. Berlin: Springer, 2010: 110-124.
[7] NEVE M, SEIFERT J P. Advances on Access-driven Cache Attacks on AES[C]// Proceedings of SAC 2006: August 2006, Montreal, Canada. Berlin: Springer, 2007: 147-162.
[8] ACIICMEZ O. Yet Another MicroArchitectural Attack: Exploiting I-cache[C]// Proceedings of the ACM Workshop on Computer Security Architecture: November 2007, Fairfax, Virginia, United States. New York, NY, USA: ACM, 2007: 11–18.
[9] YAROM Y, FALKNER K. FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache side-Channel Attack[EB/OL]. Cryptology ePrint Archive, <http://eprint.iacr.org/> Report 2013/448, 2013.
[10] BRYANT R E, O'HALLARON D R. Computer Systems A Programmer's Perspective[M]. Beijing: China Machine Press, 2014.
[11] Intel 64 and IA-32 Architecture Optimization Reference Manual[R]. Intel Corporation, April 2012.
[12] ZHAO Shujie, ZHAO Jianxun. Signal Detection and Estimation Theory[M]. Beijing: Publishing House of Electronics Industry, 2013.
[13] RIVEST R L, SHAMIR A, ADLEMAN L. A Method for Obtaining Digital Signatures and Public-key Cryptosystems[J]. Communications of the ACM, 1978, 21(2): 120-126.
[14] ZHANG Y, JULES A, REITER M K, et al. Cross-VM side channels and their use to extract private keys[C]// Proceedings of the 19th ACM Conference on Computer and Communication Security: October 2012, Raleigh, North Carolina, United States. New York, NY, USA: ACM, 2012: 305-316.
[15] ZHAO Xinjie, GUO Shize, ZHANG Fan, et al. A Comprehensive Study of Multiple Deductions-based Algebraic Trace Driven Cache Attacks on AES[J]. Computers & Security, 2013, 39(4): 173-189.
[16] WANG Tao, ZHAO Xinjie, GUO Shize, et al. Research of Cache Timing Template Attack on AES[J]. Chinese Journal of Computers, 2012, 35(2): 325-341.
[17] ZHAO Xinjie, WANG Tao, GUO Shize, et al. Driven Cache Timing Attack Against AES[J]. Journal of Software, 2011, 22(3):572-591.
[18] YAROM Y, BENGER N. Recovering OpenSSL ECDSA nonces using the FLUSH + RELOAD cache side-channel attack[EB/OL]. Cryptology ePrint Archive, <http://eprint.iacr.org/> Report 2014/140, 2014.
[19] BENGER N, van de POL J, SMART N P, et al. "Ooh Aah... Just a Little Bit": A Small Amount of Side Channel Can Go a Long Way[C]// Proceedings of the CHES 2014: September 2014, Busan, South Korea. Berlin: Springer, 2014: 75-92.

Biographies

ZHOU Ping, is a currently a Ph.D. student in Department of Information Engineering, Ordnance Engineering College, Shijiazhuang, China. He received a B.S. degree from University of Electronic Science and Technology of China, a M.S. in Ordnance Engineering College. His main research interest is side channel analysis of public-key cryptography. Email: zhou.ping_01@hotmail.com

WANG Tao, was born in 1964. He received his Ph.D. degree in computer application from Institute of Computing Technology Chinese Academy of Sciences in 1996 and master's degree in computer application from Ordnance Engineering College in 1990. He is currently a Professor in Ordnance Engineering College. His research interests include information security and cryptography.

LI Guang, received his M.S. and B.S. degrees in Department of Computer Engineering, Ordnance Engineering College, Artillery Command Academy in 2007 and 1996, respectively. He is currently an Senior Engineer in Institute of North Electronic Equipment. His main research interest includes computer security and side channel analysis of block ciphers.

ZHANG Fan, was born in 1978. He received his Ph.D. degree in the Department of Computer Science and Engineering from University of Connecticut in 2012. His research interests include side channel analysis and fault analysis in cryptography, computer architecture, security in wireless sensor network, and more. Currently he is working in the Department of Information Science & Electrical Engineering, Zhejiang University, China. *The corresponding author. Email: fanzhang@zju.edu.cn

ZHAO Xinjie, received his Ph.D., M.S. and B.S. degrees in Department of Information Engineering, Ordnance Engineering College in 2012, 2009 and 2006, respectively. He is currently an Engineer in Institute of North Electronic Equipment. His main research interest includes side channel analysis, fault analysis and combined analysis of block ciphers. He won the best paper award in Darmstadt - the 3rd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2012).