An Improved Trace Driven Instruction Cache Timing Attack on RSA

Chen Cai-Sen^{1*}, Wang Tao¹, Chen Xiao-Cen² and Zhou Ping¹

¹ Department of Computer Engineering, Ordnance Engineering College, China ² Department of Optical and Electrical Engineering, Ordnance Engineering College, China caisenchen@163.com

Abstract. The previous I-cache timing attacks on RSA which exploit the instruction path of a cipher were mostly proof-of-concept, and it is harder to put them into practice than D-cache timing attacks. We propose a new trace driven timing attack model based on spying on the whole I-cache. An improved analysis algorithm of the exponent using the characteristic of the size of the window is advanced, which could further reduce the search space of the bits of the key than the former and provide an error detection mechanism to detect some erroneous decisions of the operation sequence. We implemented an attack on RSA of OpenSSL under a practical environment, proving that the feasibility and effectiveness of I-Cache timing attack could be improved.

Keywords. Instruction cache-timing attacks, side channel attack, RSA cryptographic algorithm, Trace-driven.

1 Introduction

Side channel attacks are based on information that is gained from the physical implementation of the system, which were firstly proposed by Kocher in 1996 [1]. The channel information can be power consumption, timings, electromagnetic radiation and so on. MicroArchitectural Attack (MA) is one kind of the side channel attacks, which exploits the microarchitectural behavior of the modern systems, was first proposed by Aciicmez in 2007. There are currently three different types of MA that had been identified so far: D-Cache, Branch Prediction, and I-Cache Analysis.

In this paper, our focus is trace driven I-Cache attack on RSA. The idea of using the nonuniform memory access timings to launch attacks on crypto systems was firstly proposed by Kelsey *et. al.*[2]. The side channel information is gained by measuring cache access times. All cache attacks can be categorized into three classes [3]: cache trace attacks, cache access attacks, and cache timing attacks. We mainly consider the trace attacks which require detailed profiling of cache access patterns during the encryption or decryption and are first used to Cache attack on AES [3, 4]. Com-

Supported by the National Natural Science Foundation of China under Grant No.60772082; the Natural Science Foundation of Hebei Province under Grant No.08M010

pare with D-Cache attacks which reveal the data-access patterns of cryptosystems, the I-Cache analysis reveals the instruction flow of cryptosystems. The cryptosystem implementations with fixed instruction flow, which is usually the case for block ciphers like AES, are not vulnerable to I-cache and Branch prediction analysis whilst D-cache attacks can exploit the table lookups of such ciphers. It is also possible to determine the execution flow of a cipher (e.g. RSA) by analyzing the data access patterns as done in [5]. I-cache analysis mainly compromises the implementations of which the execution flow remains key-dependent, such as RSA, which has been proved by Aciicmez [6].

The goal of this paper is to improve the feasibility and effectiveness of current Icache attacks on RSA, especially Aciicmez's I-cache attack (Aciicmez's attack in short) [6, 7]. We analyze the complications in Aciicmez's attack in practical attack, there are some reasons show that it is hard to put into practice. For example, the spy routine requires the knowledge of logical address space of the multiplication function instructions (target instructions in short), but it is hard to get it in real attack; Even if we could get these logical addresses of the target instructions, the spy routine could not always spy on the access state of the I-cache set which maps to the target instructions, because the spy routine instructions; Finally, the number of bits evolved by the former analysis algorithm is too limited to break RSA in polynomial time. The detailed analysis will be described as follows.

Our Contributions.

- The first contribution of our work is to propose a new trace drive I-cache timing attack model on RSA, which utilized a spy routine to spy on the access state of the whole L1 I-cache, instead of the special instruction cache to which the target instructions mapped;
- The main contribution is that we advance an improved analysis algorithm of the exponent based on the characteristic of the size of the window in Sliding Window Exponentiation Algorithm (SWE), which can infer more bits of the exponent and further reduce the search space of the bits of the key than the former. It is possible for I-cache timing attack to break RSA in polynomial time with this improvement;
- In our improved analysis algorithm, an error detection mechanism is provided to detect the erroneous decision when recovering the sequence of operations, and reduce the number of the erroneous bits which are analyzed from the sequence of operations, therefore the veracity of decision could be improved. And our experimental results may be the first results of I-cache timing running under a dual-core processor, which can prove that the dual-core processor is also vulnerable to I-cache timing attack.

Outline. In Section 2 we firstly recall the MicroArchitectural Analysis and analyze the complications in the previous I-cache timing attack on RSA, then propose a new attack model. In Section 3, we advance an improved analysis algorithm of the exponent in Section 4. The experimental results and discussion are shown in Section 5. We conclude our paper in the last section.

2 A New I-Cache Timing Attack Model on RSA

2.1 MicroArchitectural Attacks and I-Cache Attack Concept

MicroArchitectural Attacks which exploit the microarchitectural components of a processor to reveal cryptographic keys is a newly evolving area of side channel cryptanalysis [6]. The functionality of some processor components generates data dependent variations in the execution time and power consumption during the execution of cryptosystems. This channel information either directly gives the key value out during a single cipher execution [8] or leaks information which can be gathered during many executions and analyzed to compromise the system [3, 4, 9]. There are currently three types of MA in the literature [10]: D-cache, Branch Prediction, and I-cache Analysis (BPA).

The cache attacks exploit the cache behavior of a cryptosystem by obtaining the execution time or power consumption variations generated via cache hits and misses. The cache attacks are firstly data-path attacks, exploiting the data access patterns of a cipher, especially S-box based ciphers like DES and AES [3, 4]. The D-cache attack on RSA was proposed by Percival in 2005 [5]. BPA and SBPA have been introduced by Aciicmez *et. al* [8, 11], who indicated that a carefully written spy-process running simultaneously with a RSA-process, was able to collect almost all of the secret key bits during one single RSA signature execution.

I-cache analysis relies on the fact that instruction cache misses increase the execution time of software applications. Each I-cache miss mandates an access to a higher level memory, i.e., a higher level cache or main memory, and thus results in additional execution time delays. It is also aimed to reveal the instruction flow of cryptosystems just like SBPA. This attack mainly compromises the implementations of which the execution flow remains key-dependent, such as RSA [6] and ECDSA [12]. The cryptosystem implementations with fixed instruction flow, which is usually the case for block ciphers like AES, are not vulnerable to I-cache and Branch Prediction Analysis.

In I-cache analysis, an adversary needs to execute a spy process, which keeps track of the changes in the state of I-cache, i.e., metadata, during the execution of a cipher process. The first I-cache attack on RSA took advantage of the fact that OpenSSL employs SWE Exponentiation which generates a key dependent sequence of modular operations in RSA, such as modular multiplications and square operations [6]. It was shown that if an adversary can run a spy routine and evict either one of these functions, he can easily determine the operation sequence of RSA. Another approach was proposed to detect the occurrences of extra reduction steps, by creating conflicts between the spy routine and the instruction executed during extra reduction step[7]. As the latter approach is similar to the former, we just consider the first approach in our paper. These attack schemes have been demonstrated in theory. However there are still some complications during practical attack.

2.2 Complications in the Previous I-Cache Timing Attack on RSA

Although the cache vulnerability of computer system has been known for a long time [1, 3], and Aciicmez *et. al* have shown that they could realize actual realistic and

practical cache attacks [6, 7]; there are still some complications in a real I-cache timing attack on RSA. In this paper, we mainly consider Aciicmez's attack as the previous correlative researches, as they are the most remarkable in the research field of Icache timing on RSA.

Firstly, the hypotheses of Aciicmez's attack is too unpractical. During the attack, these dummy instructions of the spy process have to map to the same I-cache location with the instruction of multiplication function, to create conflicts between the spy routine and the instruction of multiplication function. This hypotheses requests the adversary to disassemble the executable file, to determine the logical addresses of target function in practical attack. However, it is hard for the adversary to get the executable file, because of the measure of protection in the target systems.

Secondly, even if the hypothesis was reasonable, we could not assure that there would be conflicts between the spy code and the target instructions. Although the adversary could make the dummy instructions of spy precisely mapped to the same I-cache set with the instructions of multiplication function, according to the principle of cache mapping, the spy code and the target instruction may be mapped to different cache line in the same cache set [16]. The spy process would be failed to spy on the trace of the target function sometimes.

Finally, Aciicmez's attack on RSA which using SWE algorithm for modular exponentiation, could only infer about 200 "scattered" bits of the exponent for a 512-bit exponent in theory. It is not clear today whether the knowledge of 200 bits scattered over 512-bit exponent is sufficient to break RSA [6]. In other words, we do not know any methods that can directly leverage this information to factorize the public modulus. That will lead to the fail of Aciicmez's attack.

The cache timing attacks need high-accuracy timing, and it is easy to be disturbed by the noise, so there may be some erroneous decisions during the analysis process. How to get the high-accuracy timing of accessing each cache set is a universal problem for all cache timing attacks.

2.3 A New Trace Driven Timing Attack Model Based on the Whole I-Cache

In order to improve the feasibility of Aciicmez's attack, we propose a new trace driven timing attack model which utilizes a spy routine to spy on the access state of the whole instruction L1 I-cache, instead of the special instruction cache to which the target function mapped. The spy's dummy instructions fill not only the special cache but also other cache, on the assumption that we don't know the logical addresses of the target instructions. The new attack model is described in Fig.1.



Fig. 1. The trace driven timing attack model based on the whole I-cache

The structure of memory and cache has been simplified as described in Fig.1. The spy process and the cipher process are denoted by SP and CP in short respectively. The SP mostly uses the NOP instructions to fill the whole L1 I-cache (Fig.1 (a)). The attack flow is similar to the flow of Aciicmez's attack. The difference is that the SP in the new model measures to fill each I-cache set, while Aciicmez's SP only measures to fill a particular I-cache set. The main attack steps are as follow:

- 1. The CP is running simultaneously with the SP on the same physical core of a simultaneous-multithreading (SMT) processor while the spy is executing its loop. The SP startups before the CP and fills the whole cache firstly (Fig.1 (b)), then repeats to fill each cache set and measure the time for each set.
- 2. The instructions executed by the CP will alter the I-cache state and cause evictions of spy's dummy instructions. When the SP measures the time to fill each I-cache set and if some cache sets have been evicted out by the CP since the previous iteration of the spy, the measurement result will be higher because the evicted dummy instruction(s) needs to be fetched from a high level memory. Thus, the SP can detect the states of which I-cache sets are changed by the CP between consecutive spy iterations (Fig.1 (c-f)).
- 3. The SP stops after when the CP ends, so it can get the whole execution trace of the instruction of CP (Fig.1 (g)). For the whole execution trace, we can select the special trace corresponding to the target instructions, according to the character of the sequence of operations, for example there should not be two or more continuous multiplication operations in SWE Exponentiation. Finally, we can get the execution flow (Fig.1 (h)), infer the bits of the exponent and break the RSA key.

3 An Improved Analysis Algorithm of Exponent

3.1 Inferring More Bits from the Sequence of Operations

Based on the proposed attack model, we can get the operation sequence in RSA signature/decryption, by collecting the I-cache timing data using a spy process run-

ning with the cipher process in parallel. For RSA realized by the square and multiplication algorithm, we can directly give the key value out during a single cipher execution, according to the fact that: If the bit of d is 1, there is one more multiplication operation than the situation that the bit of d is 0. While the OpenSSL implements RSA algorithm with Chinese Remainder Theorem (CRT) to improve the speed of execution [14], the private key d is changed into d_p and d_q . If the d_p and d_q are gotten, we are able to factorize the public modulus. The bits of d_p and d_q are inferred with the same approach, so we only focus how to get the bits of d_p exponent in m^{d_p} using Icache timing attack.

The analysis algorithm of Aciicmez's attack could get some bits of d_p based on the fact that the modular exponentiation is implemented using SWE Exponentiation. From the sequence of multiplications and squarings, they can typically obtain about 200 bits out of each 512-bit exponent [6]: For each multiplication a "1" bit can be inferred, since the multipliers are all odd powers of *m*, and any time there are more than five squarings without an intervening multiplication, the presence of one or more "0" bits can be inferred, since the multipliers are of degree at most 31. The size of the window for a 512-bit exponent is *win_size* = 5; the detail analysis process is shown in Fig.2. Compare with D-cache attack, I-cache attack could not get more bits from each exponent using the relationship between the indexes of table and the value of the window. D-cache timing attack on RSA could get 110 more bits based on this fact [5]. However the 200 "scattered" bits of each 512-bit exponent are too limited for us to break RSA in polynomial time for Aciicmez's attack.



Fig. 2. Analysis the bits of the exponent from the sequence of multiplications and squarings

As shown in Fig.2, according to the analysis algorithm of Aciicmez's attack, four "1" bits can be obtained from the sequence in which there are four multiplications, three "0" bits can be inferred from the 8 squarings without an intervening multiplication. We propose an improved analysis algorithm based on the characteristic of the size of the window in SWE Exponentiation: Every time the window slides, it always starts from a bit of 1, sliding the size of one window, and ends at the location of "1" bit. As shown in Fig.2, there are 4 windows, the k_1^{th} , k_2^{th} , k_3^{th} , and k_4^{th} windows; the size of each window is different. According to the rule of sliding the window, we can conclude that if the size of the current *now_size* is smaller than *win_size*, there are *win_size-now_size* "0" bits after the current window. In Fig.2, from three continuous

squarings "S3", we can evolve the max size of the k_2^{th} window is 3, so there are two "0" bits after the k_2^{th} window. Based on this conclusion, we can further deduce that there are most three squarings in the five continuous squarings "S5" for the k_3^{th} window, so two "0" bits can be also evolved after the k_3^{th} window, which can be also inferred by the former analysis algorithm. The combination of the new and former analysis algorithm can get more bits from the sequence of operations. In Fig.2, at least 2 more bits can be inferred among the 22 bits. For a 512-bit exponent, about 50 more bits can be inferred with the new analysis algorithm. The search space of the bits of the key can be further reduced.

3.2 Algorithm with Error Detection

In the practical attack, as the instruction of SP may be evicted from the I-cache by other instructions instead of the target instructions during the CP execution, or the cache timing data may conclude some noises from other processes in the operating system, affecting the precision of timing, so it is possible to make some erroneous decisions when determining the operation sequence. If we had got some erroneous sequence of operations, some erroneous bits of the exponent would be inferred also. The more erroneous decisions occurred, the less correct bits of the exponent would be inferred, and there would be more difficulty in breaking RSA. In this paper, we provide an error detection mechanism that can detect some erroneous decisions to reduce the number of the erroneous bits which are inferred from the sequence of operations.

According to the rule of sliding the window in SWE Exponentiation, we can find some characteristics of the operation sequence: Firstly, there could not be continuous multiplications in one window; secondly, there could be no more than one multiplication among five suqarings. Based on this fact, if the inferred operation sequence does not accord with these characteristics, we can decide that an erroneous decision has occurred. Such as, if the sequence of operations was got as follow: MSSMSSMS, we can conclude that the second or third multiplication may be an erroneous decision, and then we can increase the sample size to reduce the noise, or enumerate two instances for each of these two operations. With the error detection for the sequence of operation, more correct bits can be inferred.

4 Experimental Results and Discussion

4.1 Environment Configuration and Timing collection

Just as Aciicmez's attack, our attack is also performed against RSA in OpenSSL v0.9.8d with the choice of SWE Exponentiation on the commodity PC platforms. In our I-cache attacks, we rely on the concept of executing a spy code, which keeps track of the changes in the state of I-cache during the execution of a cipher process. The SP is written according to the structure of the L1 I-cache. The spy code can refer to the generic I-cache spy process in [12], but we use the *movnti* instruction instead of the *movbi* instruction to reduce the influence on accurate timing from the time recording operation. We can realize this with the I-cache as well using a spy process that is essentially the I-cache analogue of Percival's D-cache spy process [5]. The *fork* function

was called to ensure that the SP can run simultaneously with the CP and determine which instructions were executed by the cipher under the Linux operating system. The SP achieves this goal by spying on the cipher execution via observing the I-cache state transitions. We concentrate on Intel Core i3 processor featuring Intel's Hyper-Threading Technology (HT), and it is a dual-core processor, while the former cache timing attacks were running on a single-core processor, such as Atom and Intel Pentium 4 [12, 13]. The detail environment configuration of our attack is shown in Table.1:

Table 1. Environment configuration of RSA I-cache timing attack

Configuration Item	Parameter	
Operating system	Linux Fedora 8	
Simultaneous multi-	Turn on	
threading		
OpenSSL	OpenSSL v.0.9.8d	
CPU	Intel Core i3 2.53 GHz	
Memory	2 GB	
L1 Cache	Cache size: 32KB	associative size: 4 way
	Cache line size: 64B	number of cache sets: 128
L2 Cache	Cache size: 256KB	associative size: 8 way
	Cache line size: 64B	

The structure of the L1 I-cache of Core i3 is: 4-way associative 32 KB cache, C = 128 cache sets, 4 cache lines in one cache set. We lay out contiguous 64-byte regions of code (precisely the size of one cache line) in labels: $L = \{L_0, L_1, ..., L_{511}\}$. Denote subsets $l_i = \{L_j \in l: j \mod C = i\}$ in this case, where all regions map to the same cache set yet critically do not share the same address tag. These subsets naturally partition $l = \bigcup_{i=0}^{c-1} l_i$. Observe that stepping through a given l_i pollutes the corre-

sponding the *i*th cache set and repeating for all *i* completely pollutes the entire cache.

The time for accessing each cache set is the CPU cycle counter, which is accessible via the "RDTSC" instruction that returns the 64-bit cycle count since the CPU initialization [16]. Based on the attack model, the SP measures to fill each I-cache set running simultaneously with the CP. It begins with regions that map to cache set zero: $l_0 = \{L_0, L_{128}, L_{256}, L_{384}\}$, stores the execution time t_0 , then continues with cache set one: $l_1 = \{L_1, L_{129}, L_{257}, L_{385}\}$, stores the execution time t_1 and so on through all 0 i < 128. When the SP measures to fill the whole I-cache, one timing trace T_0 is collected, and then the SP repeats to collect the next trace T_1 and so on until the end of the CP execution. In our experiment, according to the consume time for the CP and one loop execution of SP, the SP repeats 2000 times to collect 2000 timing traces.

4.2 Experimental Results and Comparison

Determine the sequence of operations

One timing trace is a list of items where each item component is a timing measurement for a distinct cache set. We illustrate in Fig. 3, where we hand picked 32 of 128 possible I-cache sets which are seemed to carry pertinent information. Each cell in the Fig. 3 indicates a cache set access time. Technically, time moves within each individual cell, and then from bottom to top through all the selected cache sets, then


Fig. 3. Partial I-cache timing traces in our experiment

The experimental result has shown that even if we could not get the logical addresses of the target instructions, the operation sequence can also be determined by analyzing the I-cache timing data.

In the practical attack, the cache timing data may conclude some noises from other processes in the operating system, affecting the veracity of the analysis, so it is hard to get the whole sequence of operations accurately. In order to reduce the noise, one attack can be executed repeatedly some times, and then the cache timing data are averaged for analyzing. We use the integrality of the operation sequence which is denoted by (the number of correctly determined operations) / (the total number of operations), as one evaluation for cache timing attack. The relationship between the sample size and the integrality of the operation sequence for Aciicmez's attack and our attacks is illustrated in Fig.4. Our attacks are divided into the improved attack and the improved attack with additional error detection.



Fig. 4. The relationship between the sample size and the integrality of the sequence of operations

In our experiment, the other processes have been minimized as possible as we can in the operating system, so the noise is minimized. From the results in Fig.4, under the same sample size, the integrality rate of the improved attack is 12~20% than the former. Compared with the improved attack, the additional error detection has a little improvement, but if there is more noise, we are sure that the error detection will play a big role in the attack.

Infer the bits of the exponent

Just the sequence of squarings and multiplications that the SWE algorithm passes through implies a significant amount of information about the exponent input, we analyze the bits of the exponent from the sequence of the operations in Fig.3. The former analysis algorithm of Aciicmez's could infer the bits as follow: 100XXXX1XXX1XXX1XXX1000XXXX1XXX1, by X denotes the unknown bit. Finally it could get total 12 bits. While the proposed algorithm can infer the bits as follow: 100XXXX1XX10XX100XX1000XXX1XXXX1, the total 15 bits are got. The experimental results show that: For one random RSA key of OpenSSL, there are 89 multiplications and 511 squarings for the 512-bit d_p exponent, and 70 partial sequences of operations which conclude more than five squarings without an intervening multiplication. The former analysis algorithm could infer 90 "1" bits (including the highest bit of d_p) and 122 "0" bits, it can infer the total 212 bits. The improved analysis algorithm can infer 46 more "0" bits than the former, and finally get the total 258 bits. About 256 bits of d_q can be inferred in the same way. Then the RSA can be broken using the lattice attack method in polynomial time [15].

4.3 Discussion and Future Work

In Aciicmez's attack, the spy routine was called inside the RSA process with a certain frequency, i.e., after each exponentiation step, so it is only as a simulation experiment and it is hard to put into practice, although Neve in [9] had shown that the theoretical and actual results taken from such a spy process are indeed very close to each other. The advantage is that the SP can accurately spy on the trace of each exponentiation step and the noise can be reduced, but it is difficult to realize this attack in practice. The new proposed attack scheme can improve the feasibility of I-cache timing attack on RSA, however in the future research, there are still some works needed to be considered: Firstly, how to reduce the noise effectively is a big problem. We can utilize some statistical tools to deal with the data; Secondly, during analyzing the cache timing data; and we will try to infer more bits of the exponent to further reduce the search space of the bits of the key.

The combination of vector quantization and hidden Markov model cryptanalysis has been demonstrated that it is a powerful tool for automated analysis of cachetiming data in [13]. In the future work we will also use this automated tool for analyzing the I-cache timing data. At the same time, some other cryptographic algorithms, especially the public cryptographic will be considered as the target of I-cache timing analysis.

5 Conclusion

In this paper, we have proposed a new I-cache timing attack model on RSA, by concurrently running a spy process to keep track of the changes in the state of the whole I-cache, while the former only spy on the state of the in the I-cache mapped to the target instruction. An improved analysis algorithm is provided to infer more bits of the exponent from the time traces, and an algorithm with error detection is advanced to detect the erroneous decisions.

We demonstrate the effectiveness by carrying out an I-cache attack on RSA of OpenSSL v0.9.8d employing SWE Exponentiation. The experimental results prove that I-cache cryptanalysis on RSA is realistic, practical and a serious threat for software systems. The improved analysis algorithm can infer about 50 more bits than the former. With this improvement, about half of the bits of d_p (d_q) can be got by I-cache timing attack, so it is possible for I-Cache timing attack to break RSA algorithm in polynomial time using the lattice attack method. And the error detection mechanism can detect some erroneous decisions to reduce the number of the erroneous bits which are inferred from the operation sequence. The dual-core processor is also vulnerable to I-cache timing attack; this result would attract more attention of the processor designer to the security of cache against cache attacks.

References

 Kocher PC. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996).

- Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side Channel Cryptanalysis of Product Ciphers. J. Comput. Secur. 8(23), 141–158 (2000).
- Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006).
- Bernstein, D.J.: Cache-timing attacks on AES (2004). http://cr.yp.to/papers.html#cachetim ing.
- Percival, C.: Cache missing for fun and profit (2005), http://www.daemonology.net/papers/ cachemissing.pdf.
- Aciicmez O, Gueron S, Seifert JP. Micro-Architectural cryptanalysis. IEEE Security and Privacy 5(4), 62–64 (2007).
- Acticicmez O and Werner Schindler. A Vulnerability in RSA Implementations Due to Instruction Cache Analysis and Its Demonstration on OpenSSL. In: Malkin T (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 256–273. Springer, Heidelberg (2008).
- Aciicmez O, Koc. C. K., Seifert. J.-P. On the power of simple branch prediction analysis. In: Robert D, Pierangela S (eds) ASIACCS 2007, pp. 312-320. ACM Press, New York (2007).
- M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. In: Biham E (ed) SAC 2006, LNCS, vol. 4356, pp. 147-162. Springer, Heidelberg (2007).
- Aciicmez O, Seifert J.-P. Cheap Hardware Parallelism Implies Cheap Security. In: 4th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), pp. 80-91. IEEE Press, New York (2007).
- Aciicmez O, Koc. C. K., Seifert. J.-P. Predicting Secret Keys via Branch Prediction. In: Masayuki A (ed.) CT-RSA 2007, LNCS, vol. 4377, pp. 225-242. Springer, Heidelberg (2007).
- Aciicmez O, Brumley B. B, Grabher P. New Results on Instruction Cache Attacks. In: Mangard S (ed.) CHES 2010, LNCS, vol. 6225, pp. 110-124. Springer, Heidelberg (2010).
- Billy B. B, Risto M. H. Cache-Timing Template Attacks. In: Matsui. M (ed.) ASIACRYPT 2009, LNCS, vol. 5912, pp. 667–684. Springer, Heidelberg (2009).
- 14. The OpenSSL Project: OpenSSL: The open source toolkit for SSL/TLS, http://www.openssl.org.
- 15. Coppersmith D. Finding a small root of a bivariate integer equation: factoring with high bits known. In: Maurer U (ed), EUROCRYPT 1996, LNCS, vol. 1070, pp. 178-189. Springer, Heidelberg (1996).
- Randal E. B, David R. O. Computer Systems: A programmer's Perspective. Second Edition. PEARSON Press, 408-433 (2011).