SCIENCE CHINA
Information Sciences

• LETTER •

March 2018, Vol. 61 039102:1–039102:3
doi: 10.1007/s11432-017-9108-3

Efficient flush-reload cache attack on scalar multiplication based signature algorithm

Ping ZHOU[1], Tao WANG[1], Xiaoxuan LOU[3], Xinjie ZHAO[2], Fan ZHANG[3,4*] & Shize GUO[2]

[1]*Department of Information Engineering, Ordnance Engineering College, Shijiazhuang* 050003*, China;*
[2]*Institute of North Electronic Equipment, Beijing* 100191*, China;*
[3]*College of Information Science and Electrical Engineering, Zhejiang University, Hangzhou* 310027*, China;*
[4]*Science and Technology on Communication Security Laboratory, Chengdu* 610041*, China*

Received 20 February 2017/Revised 9 May 2017/Accepted 19 May 2017/Published online 16 August 2017

Citation Zhou P, Wang T, Lou X X, et al. Efficient flush-reload cache attack on scalar multiplication based signature algorithm. Sci China Inf Sci, 2018, 61(3): 039102, doi: 10.1007/s11432-017-9108-3

Dear editor,

Cache timing attack is a very powerful side channel attack technique to break cryptographic implementations. Recently, Flush-Reload, a new type of cache attacks, was proposed to attack cryptographic implementations on multi-core platforms. It utilizes a spy processes $\mathcal{S}$ to monitor the victim process $\mathcal{V}$ which shares the same memory pages. $\mathcal{S}$ flushes the specific memory lines of $\mathcal{V}$, evicts data from all levels of caches, and then reloads these memory lines into cache. The corresponding loading time can be measured, therefore the victim's accesses to the specific instructions can be monitored.

The selection of the location of the memory lines to be monitored is very important in flush-reload attacks. In [1], RSA based on square-and-multiply method is attacked where the memory lines in the loop body are targeted. In [2], ECDSA based on Montgomery ladders is attacked, where the memory lines in each arm of the conditional branches are chosen. However, when the scalar multiplication is implemented by binary method, neither of the existing methods are suitable for elliptic curve (EC) based signature algorithm (e.g., ECDSA and SM2-DSA [3], shown in Figure 1(a)). More specifically, it is difficult to find the memory lines which are executed frequently in loop body as in [1]. In [2], the source code of the conditional branch in binary method is so short that the corresponding machine code of different arms may reside in the same or the adjacent memory lines. As a result, the spy program cannot distinguish which arm is executed. Meanwhile, a very high error rate needs to be handled when recovering those scalar bits.

In this letter, we present a new method on how to select the memory line to be monitored, i.e., those contains multiple function calls. Those function calls may significantly increase the probability for the adversary to capture the victim's cache accesses. We apply this method to the EC-based signature algorithms implemented in OpenSSL with the binary method. As a result, the full scalar bits can be recovered by only ONE single signature with a probability of 79%. The presented method is also applied to EC based signatures which employ sliding window method and wNAF method, recovering almost all doublings and additions with an error rate as low as 0.15%.

Flush-Reload cache attack. In EC based signatures, the binary method computes P, a point on the curve, with the scalar multiplication $P = [k]Q$ by scanning the bits of the scalar k, where Q is the base point. The goal is to recover k. As shown

* Corresponding author (email: fanzhang@zju.edu.cn)
The authors declare that they have no conflict of interest.

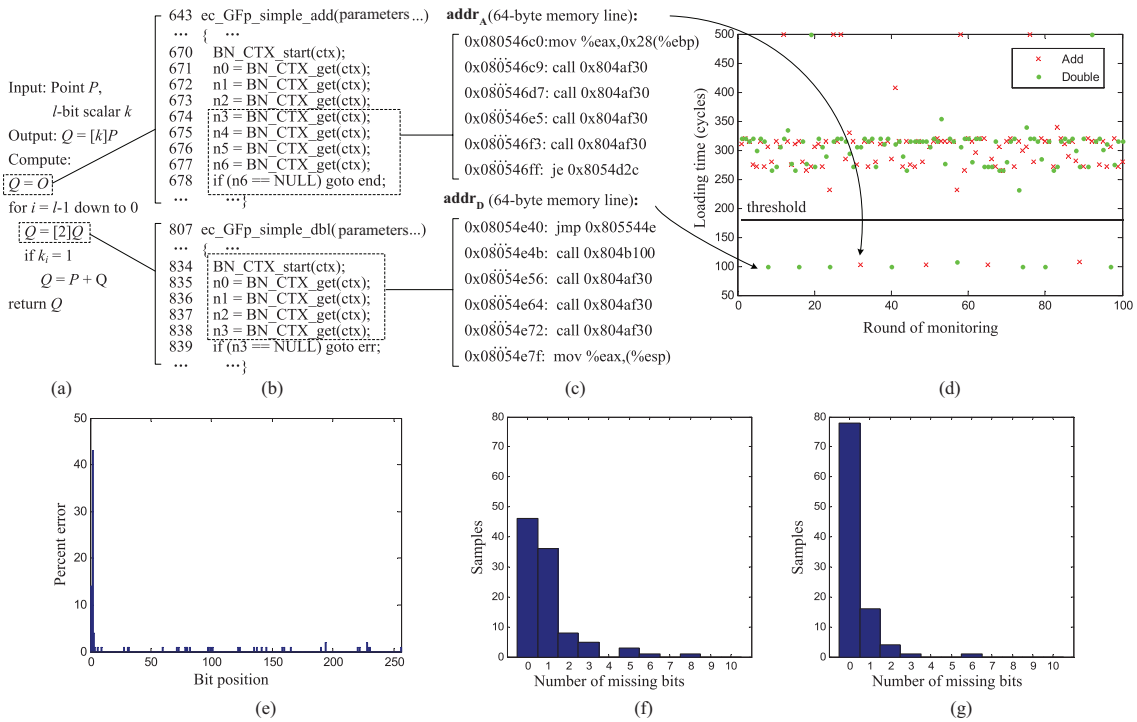

Figure 1 (Color online) Selecting the monitored memory line and the attack result. (a) The binary method; (b) OpenSSL implementation of the loading time addition and doubling; (c) location of the monitored memory line; (d) measurement of the loading time; (e) error bits per bit position; (f) error bits per scalar; (g) error bits per scalar (exclude the first 2 bits).

in Figure 1(a), the doubling is executed for each k_i, followed by an addition if the current bit is set. In the rest of this letter, we denote doubling and addition as D and A, respectively. To recover the AD sequence, and to further extract the scalar bits, the Flush-Reload attack does the follows.

Stage 1: The monitored memory lines of the instruction addresses are selected for the addition and doubling, denoted as addr_A and addr_D, respectively.

Stage 2: The cache side channel leakages are monitored by timing. Each round of monitoring consists of three phases. (1) In Flush phase, $\mathcal{S}$ uses clflush instruction which is supported by x86 processor architecture to evict both of the line addr_A and the line addr_D from all levels of caches. (2) In Trigger phase, $\mathcal{S}$ waits for a time interval t_w without executing any operation. During this phase, $\mathcal{V}$ executes the addition or the doubling, and triggers the corresponding memory loads from line addr_A or addr_D. (3) In Reload phase, $\mathcal{S}$ reloads both of the memory line addr_A and line addr_D and measures the time, denoted as t_A and t_D, respectively.

$\mathcal{S}$ repeats three phases until it reaches the end of the signature. In each round of monitoring, a vector of cache access stateis derived, denoted as $\boldsymbol{t}_i = [t_{iA}, t_{iD}]^{\mathrm{T}}$. The vectors of n rounds of monitoring form a cache access trace $\boldsymbol{T} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_n]$, as shown in Figure 1(d).

Stage 3: Each t_i corresponds to one operation o_i from three cases: doubling, addition or the rest, denoted as A, D, R, respectively. A threshold h is used to deduce o_i from t_i as follows, resulting in a sequence $\boldsymbol{O} = [o_1, o_2, \ldots, o_l]$:

$$o_i = \begin{cases} A, \text{ if } t_{iA} < h \wedge t_{iD} > h, \\ D, \text{ if } t_{iA} < h \wedge t_{iD} > h, \\ E, \text{ else.} \end{cases}$$

Then, the scalar k can be inferred as the bit value of scalar is corresponding to the operations.

Our solution. To raise the chance to capture the victim's memory access during a group operation, $\mathcal{S}$ should monitor the memory line that is accessed frequently. However, in most implementation of scalar multiplication, the source code is very simple and straight-forward during a single group operation. Here are two disadvantages. First, there are very few lines to be monitored. Second, each instruction is executed only once. Therefore, it will be very difficult for $\mathcal{S}$ to catch the correct cache behavior due to small t_w.

However, the adversary can look dig into the machine code level and take advantage of the function calling, which can lead multiple accesses to the same memory line. When the machine codes of the call instruction are executed, the process jumps to the address of the callee function. After executing

the callee function, the process returns and continues from the address after the call instruction. Thus, in a single execution, the memory line which contains n call instructions would be accessed $n+1$ times. This phenomenon provides two advantages for the adversary: The multiple accesses caused by function calling increase the chance for $\mathcal{S}$ to catch the cache behavior. Besides, the execution of the invoked function separates the accesses in different time slots, making the accesses easier to be detected and distinguished.

In our build of OpenSSL, we selected the monitored memory lines addr_A and addr_D where line 674 to 678 of the source code in the addition and line 834 to 838 in the doubling functions are located, as shown in Figure 1(b). These memory lines contain the largest number of call instructions. Take line addr_A as an example. The machine codes in the line addr_A and their virtual address are illustrated in Figure 1(c). It contains four call instructions which cause five accesses to the memory line during a single execution. As a result, the probability for $\mathcal{S}$ to capture at least one of the victim's accesses will be significantly increased, in spite of the potential overlap or undistinguished accesses. With a very high probability, the AD sequence can be completely recovered and the secret key can be extracted.

Experiment. The experiment is conducted on a ThinkCentre desktop, equipped with an Intel i5-3470 processor, a 4 GB DDR3-1600 memory and Fedora 18. The scalar multiplication employs the binary method. The addition and the doubling are implemented by OpenSSL 1.01e. The 256-bit scalar and the curve parameters are generated randomly. We set the time slot t_w as 600 cycles and the threshold h as 180 cycles.

We conduct 100 signings, collect the timings on the monitored memory lines, and try to recover the scalar. On average, the total error rate is 0.38% (0.96 among the 256 scalar bits) which is lower than 4.26% as in [2].

The distribution of the positions of those error bits is showed in Figure 1(e). The errors tend to occur at the first 2 bits. It may be caused by the unstableness at the start of the execution. The error rate of the rest bits goes below 2%. The distribution of the number of error bits in the 100 signatures is displayed in Figure 1(f). All of the recovered scalars have no more than 8 error bits and 46 of them are totally correct. As the errors mainly occur at the first 2 bits whose value can be easily verified via a simple brute forth later, we can ignore them when recover the scalar. Thus, the distribution of the number of error bits in the 100 signatures is displayed in Figure 1(g). All of the recovered scalars have no more than 6 error bits and 79 of them are totally correct.

Conclusion. In this letter, we present a new method to select the monitored memory line which are accessed for several times due to the function calling. The experiments show that our improvement to the Flush-Reload attack can recover the key with a probability of 79%, when the signing uses the binary algorithm in scalar multiplication. Our work can also be extended to attack those scalar multiplications with sliding window method and wNAF method. For 256-bit scalars and 3-bit window, the AD sequence is recovered with an error rate as low as 0.15%. In 73% of the signatures, the AD sequence is exactly recovered. As the scalar cannot fully recovered from the AD sequence when using sliding window or wNAF method, the presented attack needs to combine the lattice techniques [4,5]. The adversary need about 344 signatures to recover the private key.

Acknowledgements This work was supported in part by National Basic Research Program of China (973 Program) (Grant No. 2013CB338004) and National Natural Science Foundation of China (Grant Nos. 61272491, 61309021, 61472357, 61571063).

Supporting information Appendixes A–C. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Yarom Y, Falkner K. Flush+reload: a high resolution, low noise, L3 cache side-channel attack. In: Proceedings of the 23rd USENIX Security Symposium, San Diego, 2014. 719–732
2 Yarom Y, Benger N. Recovering OpenSSL ECDSA nonces using the FLUSH + RELOAD cache side-channel attack. IACR Cryptology ePrint Archive, 2014, 2014: 140
3 State Cryptography Administration of China. Public key cryptographic algorithm SM2 based on elliptic curves. GM/T 0003-2012. http://www.oscca.gov.cn/UpFile/2010122214822692.pdf
4 Nguyen P Q, Shparlinski I E. The insecurity of the ellipitc curve digital signature algorithm with partially known nonces. Design Code Cryptogr, 2003, 30: 151–176
5 Liu M, Chen J. Partially known nonces and fault injection attacks on SM2 signature algorithm. In: Proceedings of Information Security and Cryptology, Guangzhou, 2013. 343–358