Volume 3, Issue 5, May 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Breaking Cryptosystem's through Cache Based Timing Side Channel Attack

Prof.S.Venkateswarlu[1]
Department of CSE, KL University
India.

G. Sri Teja [2]
Department of CSE, KL University
India.

D eepa.G.M [3]
Department of CSE, KL University
India.

G.S.Raghavendra [4]
Department of CSE, KL University
India.

Abstract— A new class of physical attacks against cryptographic modules, called as side-channel attacks (SCA) are a powerful technique to reveal secret keys of cryptographic devices. SCA tries to exploit specific properties of the implementation, operating environment such as timing, power or electro-magnetic analysis of a cryptosystem. This paper presents the timing based side-channel attacks in which the attacker tries to break a cryptographic algorithm by using information about the execution times of its encryption or decryption queries. In general, a timing attack watches data movement of the CPU and of memory, while an algorithm is running. By carefully measuring the amount of time required to perform operations, attackers may be able to find the secret keys, and break cryptosystems. Removing timing-dependencies is difficult in algorithms that use operations that frequently exhibit varied execution time. Timing attacks may become serious threat to security of cryptographic algorithms if the adversary knows the internals of the hardware implementation, and even more so, the cryptosystem in use. Sometimes this timing information is combined with cryptanalysis to improve the rate of information leakage.

Keywords— side channel attacks, timing attacks, software timing attacks, cache timing.

I. INTRODUCTION

Modern cryptology was mainly focused on defining crypto-systems resistant against theoretical attacks. However, with the increasing use of secure embedded systems, researchers have focused on exploiting the physical syndromes leaking from secure devices during a cryptographic operation to disclose the key. As a result, a new kind of attack called Side-Channel Attack (SCA) has appeared.

A side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than theoretical weaknesses in the algorithms. The most common types of side channel information are:

- Timing attacks,
- Simple and differential power analysis attacks &
- Electromagnetic attacks.

Power analysis: attacks which make use of varying power consumption by the hardware during computation.
Electromagnetic analysis (EM analysis): attacks based on leaked electromagnetic radiation which can directly provide plaintexts and other information
Timing Attacks: Timing attacks are based on measuring the time it takes for a unit to perform operations. This information can lead to information about secret keys. For example, By carefully measuring the amount of time required to perform private key operations an attacker might find for eg like Diffie-Hellman exponents, factor RSA keys and break other crypto systems. If a unit is vulnerable the attack is computationally simple and requires only known cipher text. Crypto Systems often take slightly different amounts of time to process different inputs. Reasons include performance optimizations to bypass unnecessary operations, branching and unconditional statements, RAM cache hits, processor instructions (such as multiplication and division) that run in non- fixed time and wide variety of other causes. Timing attacks are fed into a statistical model that can provide the guessed key bit with some degree of certainty (by checking correlations between timing measurements).

It is interesting to notice that despite the overwhelming negative connotation of side-channel attacks, they might not be all bad. It is currently being proposed to use them to do device finger printing. This would allow detecting cloned smart cards, which are programmed with the secret information and the logical functionality of the original, from genuine smart cards as both cards would not have the same characteristic leakage.

Draw Backs of Side Channel Leakage: Typical targets of side-channel attacks are security ICs used in embedded devices and smart cards. Not only are they an alluring target as they are dedicated to performing secure operations, they

are also rather easy to analyze. In general, it is a simple device running a single process at a low to moderate clock frequency with direct access to the side-channel of interest and with (precise) control over synchronization and measurements.

Eg:- Cache Attacks.

A **cache attack** works as follows. The cache is used to store recently used data in a fast memory block close to the microprocessor. Whenever this data is used subsequently, it can be delivered quickly. On the other hand, whenever the microprocessor requires data that is not in the cache, it has to be fetched from another memory with a larger latency. The time difference between both events is measurable and provides the attacker with sufficient data on the state and the execution of the algorithm to extract some or even all secret key bits.

Resource Sharing: The cache is shared between the different processes running on the same CPU and since the cache has a finite size, they compete for the shared resource. This means that one process can evict another. process's data from the cache if some of their data is mapped to the same cache line, where a cache line (aka. cache entry) is the smallest unit of memory that can be transferred between the main memory and the cache.

This has two consequences.

- Firstly, a spy process is able to observe which cache lines are used by the crypto process and thus determine its cache footprint.
- Secondly, the spy process is able to clean the cache. It can remove all of crypto's data from the cache, which will increase the number of cache misses of the crypto process as the cache does not contain any of its data.

This also has the advantage that the attacker can readily hypothesize a known initial state of the cache, which is required to estimate the leakage.

II. TIMING SIDE CHANNEL ATTACKS

Timing attacks exploit the fact that, when the running time of a cryptographic algorithm is non-constant, then it may leak information about the secret parameters the algorithm is handling. Every logical operation in the algorithm takes time. An attacker is interested to gather timing information leaked when thousands of queries are sent and measures the time taken to respond. This type of attack is primitive in the sense that no specialized equipment is necessary.

A. Timing Attack Requirements

- Access to the device.
- Access to time reference such as real time clock.
- Ability to measure calculation time precisely
- Attacker knows the security system (DES, RSA, etc)
- Running times are reproducible.

B. The Basic Principle

Attacker thinks cryptosystem as a black box with input and output which constitute the "main channel" of the system. The only unknown being the secret key used by the cipher, attacker can measure the time it takes for the system to give an output after given an input. The time required for different inputs may vary, forming a timing distribution. If this timing distribution is related to the secret (key bits) in the system, then there is a chance to reveal the secret key.

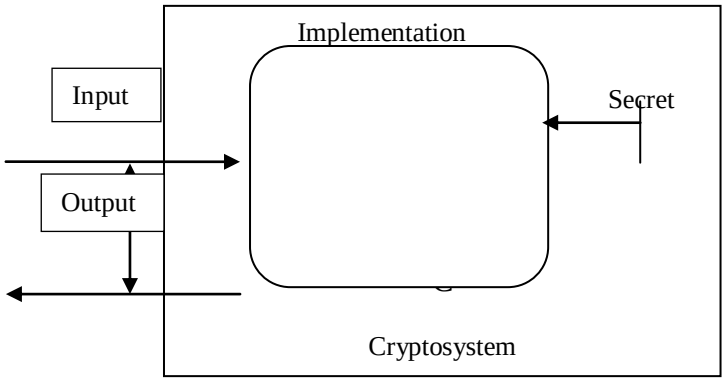

Fig. 1Timing Attack Principle

C. Cache Structure & Operation

Cache memory is positioned between the CPU registers and main memory, in the overall memory hierarchy, minimizing the latency of main-memory. Nowadays common processors offer 3 levels of cache memory, with the first level (L1) being the closest to the CPU registers. Generally, L1 and L2 caches are each split into instruction caches and separate data caches. L3 usually stores both instructions and data. Regardless the purpose or level of the cache, it is structured in the same manner. Similarly to other memory types, a cache is divided into blocks (also called lines) of fixed size (B bytes). General sizes are 32, 64 or 128 bytes, but the size of the blocks can vary for each cache level. Blocks, are grouped into a number (S) of sets such that each set has an equal number of blocks. The number (W) of blocks in a set denotes the associativity of a cache, and is usually a small power of 2. The total cache size is: S _W _ B bytes. Since this size is much smaller than the number of directly addressable bytes in main memory N (e.g. 4 GB on

32-bit CPUs), a mapping strategy needs to be adopted. The cache associativity determines how the main memory blocks map into blocks of the cache. More specifically, if we have a 4-way set associative cache, then a main memory block can map to any of the 4 blocks of a given set.

There exist replacement algorithms that select which block will be replaced in the given set. When a memory reference is made by the CPU, the tag address of the main memory block is compared with all the tags in the corresponding set. If the tag is found then this reference qualifies as a cache hit. Meaning that there is no need to retrieve the data from main memory since it is already located in the cache, and the data can be immediately provided to the CPU. Conversely, if the tag is not found in the corresponding set then the memory reference qualifies as a cache miss. Meaning that the data needs to be retrieved from main memory. Consequently, the data will be provided to the CPU with a noticeable delay relatively to the case where a cache hit occurs. The caching mechanism operation is based on the principals of spatial and temporal locality, which help to minimize the number of cache misses. Temporal locality states that the same data blocks will likely be requested repeatedly during the execution of a process. Spatial locality states that data blocks from nearby addresses are likely to be subsequently accessed. Even though the number of cache misses is reduced by these principles, they are not eliminated. Hence, the variation in the access time of different memory addresses is the key aspect of cache timing attacks.

D. Attack Methodology

The methodology behind timing measurements is as follows: We measure the duration of a certain memory access operation. The time span of this operation then (possibly) reveals certain information about the kernel space layout. Our timing side channel attacks can be split into two categories:

- **L1/L2/L3-based Tests**: These tests focus on the L1/L2/L3 CPU caches and the time needed for fetching data and code from memory.
- **TLB-based Tests**: These tests focus on TLB and paging structure caches and the time needed for address translation

Let us denote a set of inputs (plaintexts) to the system by $S_M = \{M_1, M_2, \ldots, M_n\}$. All the possible keys compose the key set denoted by $S_k = \{K_1, K_2, \ldots, K_d\}$, where d is the number of possible keys. If the cryptosystem implementation attacker wants to attack is vulnerable to timing attacks, the timing distribution of the input will be dependent on the key used in the system. Thus for key K_i, timing distribution is denoted by $P_i(t) = f(S_M, K_i)$ which is different from that of other keys. For the system to be attacked, attacker measures the timing information for a set of input values from the set S_M, and form a timing distribution P(t). The attack to the system will be reduced to a usual detection problem which tries to detect K_i knowing $P_i(t)$ and P(t). The detection problem for example has general form of solution like : if $T(P(t), K_i) > \text{Threshold}(K_i, S_M)$, K_i is detected. As long as there is the proper transform function T and the threshold functions, it's very easy to break the system.

First Category: Cache Probing Attack

Our first method is based on the fact that multiple memory addresses have to be mapped into the same cache set and, thus, compete for available slots. This can be utilized to infer (parts of) virtual or physical addresses indirectly by trying to evict them from the caches in a controlled manner. More specifically, our method is based on the following steps: first, the searched code or data is loaded into the cache indirectly (e.g., by issuing an interrupt or calling sysenter). Then certain parts of the cache are consecutively replaced by accessing corresponding addresses from a user-controlled eviction buffer, for which the addresses are known. After each replacement, the access time to the searched kernel address is measured, for example by issuing the system call again. Once the measured time is significantly higher, one can be sure that the previously accessed eviction addresses were mapped into the same cache set. Since the addresses of these colliding locations are known, the corresponding cache index can be obtained and obviously this is also a part of the searched address.

E. Illustration of Attack Methodology:

Consider the timing attack on the following modular exponentiation algorithm:

Let $S_0=1$

For k= 0 upto w-1

If (bit k of x) is 1 then

Let $R_k = (S_k * y) \bmod n$ ——————— (1)

Else

Let $R_k = (S_k)$

Let $S_{k+1} = (R_k)^2 \bmod n$

End For

If we have known exponent bits 0…(b-1), we will know the value of S_b . If bit b is 1, operation (1) will be performed, and for some values of S_b, operation (1) will take longer than for other values of $S_b$due tomodular reduction operations If attacker finds such a timing difference between these two kinds of value S_b , then bit b is 1. Otherwise bit b is 0. Attacker repeats the attack for the entire loop, there by finally knowing the entire exponent which is supposed to be secret.

Let's assume the bits 0…(b-1) are known ,and have to find the bit b of exponent. Attacker knows the value y which is open to public and can calculate the value of S_b from bits 0 to b-1 of exponent and y. Also he/she has large number of

inputs to cryptosystem and record the corresponding timings used in the system. Since he/she now target bit b of the exponent, attacker can write the timing for a specific input M (the value of y) as :

$T_M = T_{n1}(M) + T_b(M) + T_{n2}(M) + N(M)$,
T_{n1} = time spent for iterations before bit b
T_b = time spent for bit b
T_{n2} = time spent for iterations after bit b
N = time for other operations in the cryptosystem

Now attacker divides the inputs into 4 sets according to value of S_b :
S10: those inputs whose $(S_b * y)^2$ mod n is done with reduction
S11: those inputs whose $(S_b * y)^2$ mod n is done without a reduction
S00: those inputs whose $(S_b)^2$ mod n is done with a reduction
S01: those inputs whose $(S_b)^2$ mod n is done without a reduction

Notice that S10 and S11 form a disjoint partition for entire input set , while S00 and S01 form another disjoint partition for entire input set. But there will have overlaps between S10 and S00, S10 and S01, S11 and S00, S11 and S01. Then using the timing data measured, indicator functions for each input set can be defined as follow:

$$I_{10} = \frac{\sum T_{S10}}{n(S10)} = \frac{\sum((T_{n1}(S10) + T_{n2}(S10) + N(S10))}{n(S10)} + \frac{\sum T_b(S10)}{n(S10)}$$

n(S10) is the number of elements in the input set S10
Similarly

$$I_{11} = \frac{\sum T_{S11}}{n(S11)} = \frac{\sum((T_{n1}(S11) + T_{n2}(S11) + N(S11))}{n(S11)} + \frac{\sum T_b(S11)}{n(S11)}$$

$$I_{00} = \frac{\sum T_{S00}}{n(S00)} = \frac{\sum((T_{n1}(S00) + T_{n2}(S00) + N(S00))}{n(S00)} + \frac{\sum T_b(S00)}{n(S00)}$$

$$I_{01} = \frac{\sum T_{S01}}{n(S01)} = \frac{\sum((T_{n1}(S01) + T_{n2}(S01) + N(S01))}{n(S01)} + \frac{\sum T_b(S01)}{n(S01)}$$

Since the left part of the indication function is independent of the value of the bit b, the probability theory tells that left part in all indication functions will be evaluated very close to same constant value when the number of elements in input sets is large enough. So attacker have following conclusions on the indication functions when looking back at the modular exponentiation algorithm:
If bit b of the exponent is 1, the right part in I_{10} will be significantly larger than that in I_{11} and therefore$I_{10} > I_{11}$. However, since S00 has overlaps with both S10 and S11, and S01 has overlaps with both S10 and S11, there will be no significant difference between I_{00} and I_{01}. On the contrary, if bit b is 0, we will have $I_{00} > I_{01}$ and I_{10} is close to I_{11} .

Consider our mathematic model for the timing attack, we can see that the transform function here is

$$T(P(t), k_i) = \frac{I_{10}}{I_{11}} \text{ , for } k_i = 1$$

$$T(P(t), k_i) = \frac{I_{00}}{I_{01}} \text{ , for } k_i = 0$$

The threshold function is

Threshold (k_i , S_M) =1 for $k_i = 1$, $k_i = 0$

One remarkable property of our attack is that it has an error-detection and error correction policies which can be used in case attacker guesses wrong bit or key.

III. ERROR DETECTION & ERROR CORRECTION POLICIES

A. Error Detection policy

This is easy to understand on an intuitive point of view: remember that the attack basically consists in simulating the computations until some point, then building two decision criteria, with only one of them making sense, depending on the searched value, and finally deciding the bit value by observing which criterion actually makes sense. Also note that each step of the attack relies on the previous ones (we need the previous bits values to simulate the computation). Now, suppose we made an erroneous decision for the value of bit ki. In the following step, we will not correctly simulate the

computations, so that the value m_{temp} we will obtain will not be the one involved in step i + 1. Our attempts to decide whether the Montgomery multiplications will involve an additional reduction or not will thus not make sense, and the criteria we will build will both be meaningless. This remains true for the following bits.

B. Error Correction Policy

Basically, the error correction policy is the following:

1) Start by performing the attack until the end, without trying any correction.
2) identify the last place P1 were the criterion was high: we can reasonably suppose that the guess was correct until this point;
3) try a first correction, by changing the bit value at position P1 +1; continue the attack until its end and identify the last place P2 were the criterion was high;
4) if P2 is further than P1, then the correction we tried was probably right; we can now recursively repeat the process: execute step 3, using P2 instead of P1;
5) otherwise, our correction attempt was incorrect: restore previous bit value, and try to change the bit value at position P1 + 2, then P1 +3, . . .
6) if the correct key is not found this way, we conclude the first error occurred before P1; we thus find the last place, before P1, were the criterion was high, and restart the same process.

IV. CONCLUSION

The attacks described in this paper represent a significant step towards developing realistic timing attacks against cryptosystem's. In its final form the attack methodology can be used to recover the 512 bit secret key which is used by cryptosystem's for secure communications. Thus, implementation-specific timing characteristics provide one such channel and can sometimes be used to compromise secret keys.

In general, any channel which can carry information from a secure area to the outside should be studied as a potential risk. Specifically the attack discussed could be employed to all the time dependent algorithms since they only require timing data and known plaintext or cipher text. Though cryptosystem's have resisted conventional cryptanalysis so far, it will be rendered useless if practical timing attacks are developed. So, the need for software implementations to protect against timing variation which is caused due to cached memory has to be considered in future.

REFERENCES

[1] Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology 4, 3-72. Biham, E. and Shamir, A. 1991..

[2] Biham, E. and Shamir, A. 1993. Di erential cryptanalysis of the full 16{round DES. In E. F. Brickell Ed., Advances in Cryptology | CRYPTO'92 , Number 740 in Lecture Notes in Computer Science (Santa Barbara, California, 1993), pp. 494{502. Springer -Verlag..

[3] Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing by YongBin Zhou, DengGuo Feng

[4] Data Encryption Standard, FIPS PUB 46-3.

[5] Kocher, P. 1996. Timing attacks on implementations of Di ffe-Hellman, RSA, DSS, and in Lecture Notes in Computer Science (Santa Barbara, California, 1996), pp. 104{113.

[6] Hardware Attacks on Cryptographic Devices Implementation Attacks on Embedded Systems and Other Portable Hardware Jem Berkes University of Waterloo Prepared for ECE 628, Winter 2006

[7] D. Chaum, \Blind Signatures for Untraceable Payments,"Proceedings of Crypto 82, Plenum Press, 1983, pp. 199-203.

[8] K. Tiri: Side-Channel Attack Pitfalls, Proc. of the ACM-IEEE Design Automation Conference (DAC 2007), 15-20, ACM-IEEE, 2007

[9] R.L. Rivest, \The RC5 Encryption Algorithm," Fast Software Encryption: Second International Workshop, Leuven, Belgium, December 1994, Proceedings, Springer-Verlag, 1994, pp. 86-96.

[10] P.R. Rogaway and D. Coppersmith, \A Software-Optimized Encryption Algorithm," Fast Software Encryption: Cambridge Security U.K., December 1993, Proceedings, Springer-Verlag, 1993, pp. 56-63.

