2014

Cache-timing attack against aes crypto system - countermeasures review

Yaseen H. Taha
University of Khartoum

Settana M. Abdulh
University of Khartoum

Naila A. Sadalla
University of Khartoum

Huwaida Elshoush
University of Khartoum

DOI: 10.4225/75/57b65fd1343d3

12th Australian Information Security Management Conference. Held on the 1-3 December, 2014 at Edith Cowan University, Joondalup Campus, Perth, Western Australia.

CACHE-TIMING ATTACK AGAINST AES CRYPTO SYSTEM - COUNTERMEASURES REVIEW

Yaseen.H.Taha, Settana.M.Abdulh, Naila.A.Sadalla, Huwaida Elshoush
University of Khartoum, Sudan
showyaseen@ribat.edu.sd, settana333@gmail.com, nailaschool@gmail.com, htelshoush@uofk.edu

Abstract

Side channel attacks are based on side channel information, which is information that is leaked from encryption systems. Implementing side channel attacks is possible if and only if an attacker has access to a cryptosystem (victim) or can interact with cryptosystem remotely to compute time statistics of information that collected from targeted system. Cache timing attack is a special type of side channel attack. Here, timing information caused by cache effect is collected and analyzed by an attacker to guess sensitive information such as encryption key or plaintext. Cache timing attack against AES was known theoretically until Bernstein carry out a real implementation of the attack. Fortunately, this attack can be a success only by exploiting bad implementation in software or hardware, not for algorithm structure weaknesses, and that means attack could be prevented if proper implementation has been used. For that reason, modification in software and hardware has been proposed as countermeasures. This paper reviews the technique applied in this attack, surveys the countermeasures against it, and evaluates the feasibility and usability of each countermeasure. We made comparison between these countermeasure based on certain aspect furthermore.

Keywords

AES algorithm, side channel attack, cache timing attack, cache timing countermeasures.

INTRODUCTION

Nowadays, attacks rely on mathematical analysis to break math-based cryptographic algorithms. These attacks use several methods such as linear and differential cryptanalysis. In order to break cryptography algorithms, the attacker can either work on ciphertext alone or a plaintext-ciphertext pair. This is done using various types of attacks such as: cipher text only, known plaintext, chosen ciphertext or chosen plaintext. In addition, attacks can be performed by obtaining extra information from encryption devices, which simplifies discovering the key or plaintext. These devices can allow a new way of leaking important information about the cipher, which may increase the likelihood of recognizing the original text or the key used in the encryption processes. This leaked information gathered from the encryption device is called side channel information. It can be about plaintext or ciphertext as obtained during the encryption process. This caused the development of a new type of attack dubbed the side channel attack. It branches out into timing attacks, power consumption attack, fault analysis attack, and acoustic attack and other. (Osvik , Shamir and Tromer, 2006).

This paper focuses on a specific implementation of cache timing attack against AES algorithm which is a Bernstein cache timing attack (that is explained in section 3.2). It then identifies countermeasures that developed to avoid this attack by reviewing the literature and examining different works that propose countermeasures to such attack from several aspects. Finally several comparisons of these countermeasures were presented. This paper is organized as follows: In section two, a brief description of AES is given. Section three explains the cache timing attack and Bernstein practical implementation of cache timing attack is described to show how the attack works and to prepare for countermeasures of the attack that is introduced in section four. In section five, these countermeasures are discussed and compared. Finally section eight conclude the paper.

ADVANCE ENCRYPTION STANDARD

In 1997, national institution for standard and technology (NIST) posed a competition for researchers around cryptography community to propose their algorithms in order to choose AES and to replace data encryption standard (DES). Rijndael cipher (Daemen, J., Rijmen, V., 2003). was chosen as AES algorithm, which was

developed by two researchers, Joan Daemen and Vincent Rijmen. Rijndael is a family of ciphers with different keys and block sizes. (Schwartz, 2000)
AES consist of four operations which are byte substitution, shift rows, mix column and add round key. These operations are repeated in a number of rounds except the final round which does not contain mix column operation. Input block size in AES is 128 bit and key size is optional 128,192 or 256. Number of rounds depend on the key size, it uses 10, 12, 14 round with 128, 192, 256 key size respectively.
Substitution operation in AES is based on lookup tables, which is termed as S-box. Also it may use several lookup tables that make it faster in execution due the ability to perform lookup mapping in parallel manner, S-box lookups use input-dependent indices that loaded from addresses in memory.
In spite of strength of AES against known attacks, cache timing attack could success due to the increases in size of lookup table, it might be became vulnerable to leaking information from cache memory which can be exploited to perform an attack in different way , resulting in a security defect on AES algorithm.

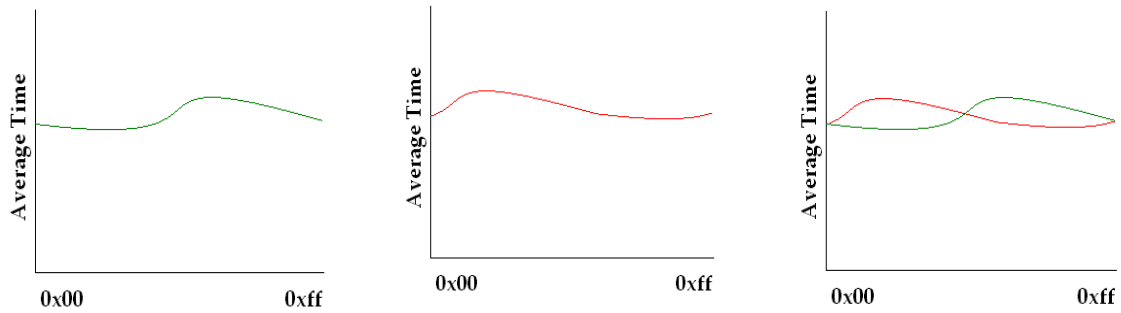
CACHE TIMING ATTACKS ON AES

Historical Background

Cache timing attack is used to extract encryption key or plaintext. This attack is based on the fact that each mathematical computation takes certain time. Gathering side channel information from the cache in order to break AES encryption has been discussed in several previous works.
Hu (1992) noted this issue via covert channel in context of international transmission.
After four years, Kelsey (1998) stated that large S-box has a probability to be attacked by exploiting cache hit rate to steal sensitive information.
(Francois and Quisquaterm 1999) described timing attack that exploits bad implementation of AES software that performs algebraic equation in bad manner (such as performing Mix Column using conditional branch).
After that Page (2002) theoretically mentioned that cache misses can be exploited by an attacker, but it only work if there exist high temporal period of cache miss, but he did not show this attack in practical implementation.
As mentioned in (Tsunoo et al., 2002) and (Tsunoo et al., 2003) that lookups process inside cipher can be used to launch timing attack due to the delay in accessing the memory.
This attack was known theoretically until Bernstein (2005) demonstrates a real implementation of the attack on AES cryptosystem. He described several weaknesses that attackers can use to crack AES encryption. His attack uses time variances that caused by cache effect, and relies on statistical timing pattern that has been collected from memory access.

Bernstein Cache Timing attack

What Bernstein proposed was a technique to reduce the AES key space by performing brute-force for locating the final key the Bernstein attack assumes that the computer executing the encryption uses cache memory, which can be described in a simple model of the cache. In this model, values are looked up in the main memory, and then transferred to the cache, evicting existing values to the memory. This noted assumption, however, has provided the attacker with ways to obtain the encrypted data from cache easily. (Bernstein, 2005)
Bernstein utilized the leaked time variances, resulting from different inputs using a known initial key. Based on these measurements, comparisons were made to time variances caused by encryption using an unknown key. This timing information can be used to extract the full encryption key in a specified key space. (Bernstein, 2005)
This attack is based on measuring the time variances in encryption of various inputs under a known key, and comparing these to time variances under a secret unknown key. With enough timing information the full key can be recovered. The first stage of the attack is to create a replica server with the same CPU and running the same implementation of AES as the victim server, but with a known key. From another machine, many random packets of 800-byte, 600-byte and 400-byte lengths are encrypted and sent to the server. The time taken to encrypt each packet is recorded and used to build a time pattern for each input bit. The next stage is to launch the attack against the victim server. First, an encrypted zero from the server is obtained and recorded for later use. Again packets of varying length are sent from the attacking machine and encrypted by the server. The time taken for encryption of each packet is recorded. Finally the timings from the victim server are compared with those of the replica server to produce a set of possible key bits. The set of possibilities is searched until a combination has been found that produces the encrypted zero obtained earlier. This combination is the victim machine's AES key (O'Hanlon & Tonge, 2005). Figure 1 shows these steps.



(b) (c)

Figure 1: (a) prepare for the attack by collecting a large body of reference timing data for each xi0 (b) collect a large body of timing data from a target machine for the plaintext byte pi (c) The target machine's timing data should be shifted from the reference data by exactly ki.

CACHE TIMING COUNTERMEASURES

Cache No Fill mode

Dev (2013) countermeasure for this attack is to activate a no-fill mode in cache memory, in this mode the memory accesses are serviced from the cache when they hit it, but miss accesses is serviced directly from memory (without causing evictions and filling). The encryption routine would then proceed as follows:
(i) Prefetching lookup table of AES into the cache.
(ii) Then, set cache mode to no-fill mode.
(iii) Execute encryption process.
(iv) Finally no-fill mode is disabled.
In step (i) and (ii) it is necessary to ensure that no another process is running. However, once this step is done, step (iii) can be carried out safely. Performance of encryption would not be affected because its inputs are pre-fetched into cache before no-fill mode is activated. On the other hand, its output will not be cached, and this cause subsequent cache misses. Simultaneous processes executed during (iii), through multitasking, thus will degrade performance as a result. Separating encryption process into smaller pieces then performing previous procedure to each piece thus could reduce effect somewhat, by allowing the cache to be occasionally updated to reflect the changing memory work set. (Dev, 2013)

Cache Partitioning

One of effective countermeasures is the cache partitioning. The basic idea is to allocate locations in cache to load values of t-table, thus t-table will usually be located in specific locations that is never overwritten by data of another process until encryption is performed. As a result, access pattern of cache will be changed in encryption process by this manner; figure 2 shows the code of cache partitioning.

```
static const u32 Te0[256]__attribute__((aligned(0x10)))= {……};
static const u32 Te1[256]__attribute__((aligned(0x1000))) = {…..};
static const u32 Te2[256]__attribute__((aligned(0x10000))) = {…..};
static const u32 Te3[256]__attribute__((aligned(0x100000))) = {…..};
static const u32 Te4[256]__attribute__((aligned(0x1000000) )) = {…..};
```

Figure 2: Piece of code of cache partitioning

Fixed Number Of Clock Cycles

(Jayasinghe, Ragel and Elkaduwa, 2014) stated that the AES software implementation is rescheduled such that it will take constant time for execution. Figure 3 shows the part of the code executed for the first round inside AES

encrypt function implementation. Initially a fixed number is defined (line 1). By using a time stamp, number of clock cycles for each round is obtained (lines 2-7). Then the difference between the two is obtained (line 8), i.e the fixed number and the number of clock cycles for each round. Then, a loop is included where it runs from zero to the number that is obtained (line 9). This is done for every round.

```
static int fixed_cycles= 250; //a fixed number  of clock cycles
int j=0;
int startT[THE_SIZE]; //the start of the timestamp
static int num_cycles[THE_SIZE]; //array for number of clock cycles
int cyc_diff[THE_SIZE]
/* round  1: */
startT[0] = timestamp1  ();
.....round  1 AES implementations.....
num_cycles[0]  = timestamp1() - startT[0];
cyc_diff[0] = fixed_cycles - num_cycles[0];
for(j=0; j< cyc_diff[0]; j++)
{
asm("nop");
}
```

Figure 3: AES fixed cycle round algorithm.

Fixed number method takes additional clock cycles than the usual. The researchers observed that it is approximately 2.1 times higher than the number of clock cycles in the unprotected AES implementation. In addition, it was detected that some of the key bytes are missing after the usage of this modified AES implementation. The attacker will be unable to recognize the authentic timing pattern properly because of the incorrect timing information. Also it was observed that the total number of possible keys is considered a large value when compared with the original AES implementation. (Jayasinghe, Ragel & Elkaduwa, 2014)

Average number of clock cycles

reference (Jayasinghe, Ragel and Elkaduwa, 2014) also proposed that instead of defining a fixed number of clock cycles by an outsider (such as a programmer), numbers of clock cycles for rounds themselves are used to *calculate an average and to perform constant encryption time by equaling clock cycles up to the averaged value.* Figure 4 shows the code fragment implemented for the fifth round inside AES encryption function in AES implementation.

Initially by defining a time stamp, number of clock cycles for each round is obtained (lines 1-7). Then the average is calculated incrementally for each round (lines 8, 9 and 10). In between each round a FOR loop is included where it executes from zero to the number that is obtained as the difference of average value and the clock cycle value of the particular round (line 11).

After implementing real experiments in this algorithm the (Jayasinghe, Ragel and Elkaduwa, 2014) observed that it is approximately 1.19 times greater than the number of clock cycles in the unprotected method.

```
1. int startT[THE_SIZE]; //the start of the timestamp
2. static int num_cycles[THE_SIZE]; //array for number of clock cycles
3. static int Cycle_avg[THE_SIZE]; //the average of clock cycles
4. int cyc_diff[THE_SIZE]; //the difference between average and a clock cycle
5.int sum = 0;
/* round 5: */
6. startT[4] = timestamp1();
……..round 5 AES implementations……..
7. num_cycles[4] = timestamp1() - startT[4];
8. sum = sum + num_cycles[4];
9. Cycle_avg[4] = sum/5;
10. cyc_diff[4] = Cycle_avg[4] - num_cycles[4];
11.for(j=0; j< cyc_diff[4]; j++) asm("nop");
```

Figure 4: AES average cycle round encryption algorithm.

After implementing their AES algorithm in real server (Jayasinghe, Ragel and Elkaduwa, 2014) noticed that some of the key bytes have been missing. So, the attack has been unable to recognize the authentic timing pattern properly because of the wrong timing information. Depending on their implementation results, the total number of possible keys was very large. That is a considerably large value when compared with the original AES implementation. Therefore, it will be impossible to find the correct key even with the attack tested.

CPU Architecture Enhancements

Instead of prevent cache timing attack by software implementation enhancements, CPU architecture can be built in a way that can protect from this attack. In the following two sections AES Instruction set and Multicore CPU is described then following a real implementation of this technique shows its immunity against cache timing attack.

AES Instruction set (AES-IN)

Intel and AMD develop an extension of X86 instruction set for multiprocessor to increase speed and security of AES encryption. (Mowery, Keelveedhi and Shacham, 2008)

AES-IN provides hardware Implementations of key generation, encryption rounds, and decryption rounds. The cryptographic operations are moved out of RAM and into custom hardware, improving performance and eliminating cache side channels. (Mowery, Keelveedhi and Shacham, 2008).

1.1.1 Experiments on modern CPU Architecture

(Mowery, Keelveedhi and Shacham, 2008) make a demonstration of Bernstein cache timing attack on Intel Core i5, Intel Xeon E5410 and Intel Atom N2800 and show how these new processors enhanced protection cache timing attack. Bernstein attack demonstration consists of three parts: preparing the attack, executing the attack, and analyzing the outcome. In the preparation, known AES key, that is, all zeros with number of packets were sent to the server having n[14] = 250.

The attack was carried out in a similar fashion as the preparation, the different is that it sending a 16-byte secret key taken from the pseudorandom number generator of the server's OS (assume to be a victim machine) instead of a known AES key (assume to be in attacker machine). After sending random packet analyzing of the attack take place by correlating the lines obtained from the preparation and from the attack to produce 16 correlations, one per byte, which show if the secret key could be easily guessed. If the produced number of possibilities is lower than 256 for a given key byte, then the range of possibilities to guess that secret byte is narrowed (Mowery, Keelveedhi and Shacham, 2008).

In test one and two, the range of possibilities was 256 for all the correlations, which indicates that the correlation was not extracting any useful information about the secret key, showing that the attack to the

Intel Core i5 and Intel Xeon E5410 was not successful. But in test three, the range of possibilities that was produced by correlating the preparation and the attack was narrower than the previous test (around 113), which indicates that the correlation was extracting some information, although limited, about the secret key, showing that the attack to the Intel Atom N2800 was somewhat successful (Mowery, Keelveedhi and Shacham , 2008).

CONSTANT TIME ENCRYPTION VIA SCHEDULING THE INSTRUCTION

(Jayasinghe et al., 2012) achieved constant time encryption via rescheduling the instructions while maintaining the same cache structure of the system. They did not make any hardware changes, their approach depended on the encryption cycles to thwart delays due to cache misses.

Their proposed countermeasure has three major steps:-

i. **Decomposition of OpenSSL AES encryption code into smaller bitwise operations**

The decomposition process was made so that it will operate like the instruction scheduling process of a compiler. They performed manual scheduling without the compiler optimization which resulted in increasing the size of the code and execution penalty. However they recommended integrating these schedules into the compiler so it would be able to carry out "side-channel aware" scheduling.

ii. **Add each and every bitwise instruction sets to queues.**

This was done by queuing the decomposed instructions to automate schedule implementation.

iii. **Processing each queue.**

Each queue can be carried out independently since they are data and control independent from each other. However, this step modified the normal execution by adding the instruction asm("nop") to stall the processor to hide data loading time in arithmetic operations for each queue.

COUNTERMEASURE COMPARISION

After surveying different countermeasures to understand their mechanisms, they are now to be discussed in terms of their advantages and disadvantages.

Cache no fill mode as a countermeasure prevent cache timing attack by activating cache no fill mode. In this approach lookup table is first pre-fetched into cache then activate no fill mode and execute encryption, So all processes will cause cache hit, therefore leading to great advantage of increasing encryption time than the normal execution and all input encryption will have equal time so no information will be leaked, and this due to eliminating the cache miss effect. On the other hand, activation of no-fill mode occurs concurrently with other processes, thus it will preventing the use of the cache memory by the other processes. As a result, this leads to degradation in other processes performance until the encryption is completed and no fill mode is deactivated.

Fixed cycle time of round countermeasure enforce each round to take fixed constant time, if the round time is less than constant time loop of asm(nop) instruction will append until fixed time has been achieved, this technique success in prevent attack without any information leaked because the attacker will observe that any encryption will take a fixed time. The drawback of this countermeasure is it need to determine the constant time by an outsider (e.g programmer) and it's difficult to determine the appropriate fixed time, if the time is too long this may result in degradation of performance and if time is short some rounds will exceed the fixed time and some information will be leaked and reducing security.

Average cycle time of rounds as countermeasure seems to be better than fixed time approach in determining time efficiently, but it also has degradation in performance comparing with original approach.

CPU architecture enhancement is very effective countermeasure because hardware implementation protect from the attack and not provide any information that can be used in this attack. The major advantage is that there is not any effect on encryption performance. However, AES-IN need to make modification in all software to be secure against cache timing and can't protect against the attack if software is not altered to call AES-IN.

Countermeasures Comparison					
Countermeasure	*Effect in performance of*		*Key Space*	*Countermeasure Type*	
	Encryption	*Other Process*		*Hardware*	*Software*
Cache NO Fill Mode	—	*High*	*High*	—	
Cache Partitioning	*Low*	*Medium*	*High*	—	
Fixed Cycle Round Time	*High*	—	*High*	—	
Average Cycle Round Time	*Medium*	—	*Medium*	—	
AES-IN	—	—	*High*	✓	✓
Multicore	—	—	*High*	✓	—
Scheduling the Instructions	*Medium*	—	*High*	—	✓

Table 1: countermeasure comparisons in performance and key space and type.

CONCLUSION

First a general background about AES algorithm is given together with a brief introduction and review about cache timing attack. The Bernstein implementation for cache timing attack is described. In this work, several countermeasures are explored, and the techniques they followed are presented and how they prevent the attack. Cache timing can be avoided using software or hardware countermeasures. Each countermeasure has its advantages and disadvantages, a table showing this comparison is presented. As any other tool, each is appropriate in a certain environment for a particular purpose. Finally a comparison between these countermeasures in terms of type, performance and key space size that attacker can gain was shown.

REFERENCES

Bernstein, D. J. (2005). Cache-timing attacks on AES.

Biham, E. (1997, January). A fast new DES implementation in software. In Fast Software Encryption (pp. 260-272). Springer Berlin Heidelberg.

Daemen, J., & Rijmen, V. (1999). AES proposal: Rijndael. National Institute of Standards and Technology. p. 1. Retrieved 21 February 2013.

Dev, D. Effective Countermeasures for Cache Timing Attack on AES.

Koeune, F., Quisquater, J. J., & Quisquater, J. J. (1999). A timing attack against Rijndael.

Hu, W. M. (1992, May). Lattice scheduling and covert channels. In Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on (pp. 52-61). IEEE Symposium on Research in Security and Privacy. [online] Available at: http://dx.doi.org/10.1109/risp.1992.213271 [Accessed 10 April. 2014].

Jayasinghe, D., Ragel, R. G., & Elkaduwe, D. (2014). Constant time encryption as a countermeasure against remote cache timing attacks. arXiv preprint arXiv:1403.7293.

Alawatugoda, J., Jayasinghe, D., & Ragel, R. (2011, August). Countermeasures against Bernstein's remote cache timing attack. In Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on (pp. 43-48). IEEE.

Kelsey, J., Schneier, B., Wagner, D., & Hall, C. (1998). Side channel cryptanalysis of product ciphers. In Computer Security—ESORICS 98 (pp. 97-110). Springer Berlin Heidelberg.

Kocher, P. C. (1996, January). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Advances in Cryptology—CRYPTO'96 (pp. 104-113). Springer Berlin Heidelberg.

Kocher, P., Jaffe, J., & Jun, B. (1999, January). Differential power analysis. In Advances in Cryptology—CRYPTO'99 (pp. 388-397). Springer Berlin Heidelberg.

Kocher, P., Jaffe, J., & Jun, B. Introduction to differential power analysis and related attacks, 1998. URL www. cryptography.com/resources/whitepapers/DPATechInfo. Pdf.
Kocher, P. C. (1996, January). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Advances in Cryptology—CRYPTO'96 (pp. 104-113). Springer Berlin Heidelberg.
Liu, W., Di Segni, A., Ding, Y., & Zhang, T. (2013). Cache-timing Attacks on AES.
Mowery, K., Keelveedhi, S., & Shacham, H. (2012, October). Are AES x86 cache timing attacks still feasible?. In Proceedings of the 2012 ACM Workshop on Cloud computing security workshop (pp. 19-24). ACM.
Osvik, D. A., Shamir, A., & Tromer, E. (2006). Cache attacks and countermeasures: the case of AES. In Topics in Cryptology–CT-RSA 2006 (pp. 1-20). Springer Berlin Heidelberg.
Page, D. (2002). Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. IACR Cryptology ePrint Archive, 2002, 169.
Percival, C. (2005). Cache missing for fun and profit (2005). URL: http://www. daemonology. net/papers/htt. Pdf.
Schwartz, J. (2000). US selects a new encryption technique. New York Times, C12.
Stefan, D., Buiras, P., Yang, E. Z., Levy, A., Terei, D., Russo, A., & Mazières, D. (2013). Eliminating cache-based timing attacks with instruction-based scheduling. In Computer Security–ESORICS 2013 (pp. 718-735). Springer Berlin Heidelberg.
Tsunoo, Y. (2002). Cryptanalysis of block ciphers implemented on computers with cache. preproceedings of ISITA 2002.
Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., & Miyauchi, H. (2003). Cryptanalysis of DES implemented on computers with cache. In Cryptographic Hardware and Embedded Systems-CHES 2003 (pp. 62-76). Springer Berlin Heidelberg.
Herath, U., Alawatugoda, J., & Ragel, R. G. (2014). Software implementation level countermeasures against the cache timing attack on advanced encryption standard. arXiv preprint arXiv:1403.1322.
Walter, C. D., Koç, Ç. K., & Paar, C. (Eds.). (2003). Cryptographic Hardware and Embedded Systems--CHES 2003: 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings (Vol. 5). Springer.