ISSN:0975-9646

Advait Kelapure et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (4) , 2015, 4106-4109

Countermeasures against Timing Attack on AES

Advait Kelapure
Department of Computer Engineering and Information Technology, VJTI, Mumbai – 400019.

M. M. Chandane
Department of Computer Engineering and Information Technology, VJTI, Mumbai – 400019.

***Abstract*—Timing Attacks are a type of side channel attacks. In timing attacks, leaking cache timing information is used as the side channel. If the information about cache timing of a software implementation of AES is collected and analyzed by an attacker, the secret key of a crypto system can be deduced. Hence AES is fallible to timing attack.**

There are a few countermeasures in the form of software code implementation which changes behavior of AES to manipulate its timing information. But it causes decrease in speed of computation which reduces the performance. So there is a need to implement a new more efficient countermeasure which this paper tries to suggest.

***Keywords*—Advanced Encryption Standard, Timing Attack, Side Channel Attack**

I. INTRODUCTION

The Advanced Encryption Standards or AES is a symmetric block cipher, and is used throughout the world for encryption. In 1997, National Institute of Standards and Technology (NIST) originated it. Data Encryption Standards (DES) which is predecessor of AES, was becoming vulnerable, and hence AES came into existence [1]. AES was capable to provide protection to the sensitive data. It was to be easy to implement in hardware and software, as well as in restricted environments (for example, in a smart card) and offer good defenses against various attack techniques [1].

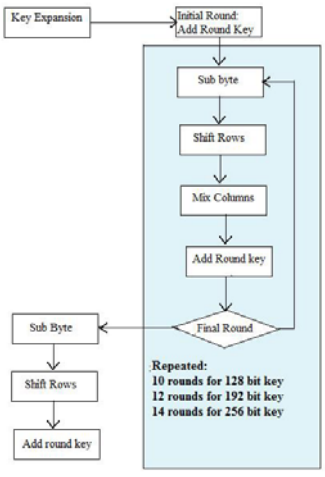

Fig. 1: Working of AES [2]

Fig. 1 depicts working of AES. In AES, according to the size of the key, the number of rounds for encryption is varied. They are 10, 12 or 14 rounds for 128-bit, 192-bit or 256-bit key respectively. In each round except the final round, four operations are taken part. They are Sub Bytes, Shift Rows, Mix Columns and Add Round Key. Byte arrays of size 4x4 (16 bytes) are used for each of these operations. After a particular number of rounds according to the size of key, the plain text is converted into the cipher text[2].

Daniel J. Bernstein [3], in the year 2005 has practically demonstrated timing attack on OpenSSL implementation of AES and successfully deduced the secrete key. So AES is subjected to fall to Side channel attack[3].

In side channel attack, attacker collects side channel information about the cryptosystem. This side channel can be any other information related to the execution of the cryptosystem such as time required for execution, power consumed by the machine to perform encryption, electromagnetic radiations generated by the system, sound produced during computation etc[4]. It does not deal with the internal mechanism of the cryptosystem. These methods are pioneered by Paul Kocher[5], who demonstrated these side channel attacks on Diffie-Hellman, RSA etc.

In all the side channel attacks, the underlying principle is that physical effects caused by the operation of a cryptosystem (on the side) can provide useful extra information about secrets in the system, for example, the cryptographic key, partial state information, full or partial plaintexts and so forth[4].

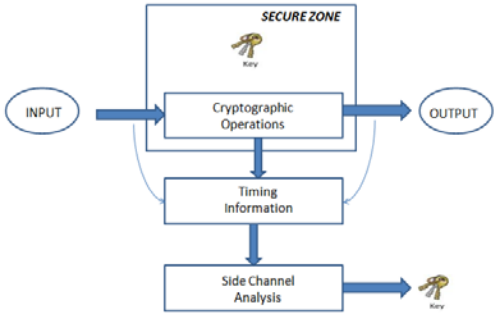

Fig. 2: Timing Attack on AES [6]

Fig. 2 depicts timing attack on AES. Attacker gathers information of time required for encrypting a particular string from side channel and after doing side channel analysis as demonstrated by Daniel J. Bernstein[3].

Timing Attacks are a kind of side channel attacks, which uses leaking cache timing information as the side channel[4]. As mentioned by Janaka Alawatugoda, Darshana Jayasinghe, and Roshan Ragel[2], at the time of execution, variables, data structures and other memory

elements used for a particular program are loaded into the main memory (RAM). Cache memory is a high-speed memory, which is located in the processor for making the memory access faster through spatial and temporal locality. Typically, the recently accessed memory areas are loaded into the cache. Since its high cost, cache memory is limited in size and only a limited amount of data can be stored. When a program needs to read a memory word, cache hardware checks to see if the line needed is in the cache. If so a cache hit occurs, the request is satisfied from the cache and no memory request is sent to the main memory. A cache hit normally takes two clock cycles. When the memory word that the processor is looking for is not found in the cache, a cache miss happens, where the data have to be taken from the main memory and therefore it takes longer time than cache hits. The time difference due to cache hits and misses are used as leaking timing information from the crypto system to perform cache timing attack[2].

II. Related Work

Bernstein[3] performed timing attack successfully by using the OpenSSL 0.9.7a AES implementation on an 850MHz Pentium III Desktop Computer, running FreeBSD-4.8 as a network server. The complete AES key was extracted using a client machine and pointed out that the same technique can be performed on more complicated servers with additional timing information. Also testing is done on an AMD Athlon, an Intel Pentium III, an IBM PowerPC RS64 IV and a Sun UltraSPARC III processor with positive results[3].

In the meantime, Kocher[5] experimented timing attacks on implementations of Diffie-Hellman, RSA, DSS and other crypto systems. Results stated that timing attacks are centred on measuring the time it takes for a unit to perform operations, where it lead to information about secret keys and break the crypto system. The experiment performed by Kocher[5] also stated that the attack is computationally not much difficult and most of the time only known cipher text is required and he has presented some techniques for preventing the attacks.[8]

Darshana Jayasinghe, Roshan Ragel, and Dhammika Elkaduwe[7] suggested countermeasures in the form of software code implementation. This changes behavior of AES to run it in random timing. That means if same secret key and same plain text is used for each time still it will take different timings after execution of encryption function for several trials[7]. But the disadvantage of this scheme is that executing random or fixed for loops increases a lot of burden on CPU which causes AES to slower down.

So, Udyani Herath, Janaka Alawatugoda, and Roshan Ragel[8] in their suggested better option than usage of for loops, which deals with number of CPU cycles to be utilized for each execution of AES independent of its inputs. This algorithm improves speed of AES over the previous ones as random number generation and executing these for loops either for random or fixed numbers increased the encryption time. The 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[8]. Initially by defining a time stamp, number of clock cycles for each round is obtained. Then the average is calculated incrementally for each round. In between each round a 'for' loop is included where it execute from zero to the number that is obtained as the difference of averaged value and the clock cycle value of the particular round[8].

The methodology suggested by Udyani Herath, Janaka Alawatugoda, and Roshan Ragel[8] i.e. using average number of clock cycles is modified using modulo operation in place of average in this paper as average requires multiplication, division, addition and subtraction. The operations particularly division requires more number of clock cycles. Instead if we use operator like Bitwise-AND which takes only one clock cycle, the speed performance will be improved

The objective of this paper is to contribute a methodology that can give better performance over the "AES with average clock cycles" methodology [8].

III. Proposed Solution

The proposed method is modification of AES with average clock cycles [8]. Rather than finding average of the clock cycles, proposed method suggests to execute each round in the CPU clocks that are multiple of 4. The multiple of 4 involves to find the remainder after we divide the The numbers of clock cycles for rounds by 4, and executing that many extra clock cycles which are not more than 3.

As X%4={0,1,2,3}, where X is any natural number.

Algorithm 1 AES with clock cycles that are multiples of 4

```
1: procedure AES WITH CLOCK CYCLES THAT ARE MULTIPLES OF 4
2:    Define StartTime, EndTime, TimeElapsed, ExtraClocks, ClockUsed
3:    ...
4:    Initial steps in AES before execution of rounds
5:    ...
6:    for each round i ϵ N
7:          StartTime = getCurrentTimeStamp();
8:                ...
9:          Execute i^th round of AES
10:               ...
11:         EndTime = getCurrentTimeStamp();
12:         TimeElapsed = EndTime - StartTime;
13:         ClocksUsed =TimeElapsed/CPUFrequency;
14:          ExtraClocks=4-ClocksUsed&3;
15:          for i ϵ ExtraClocks
16:                asm("NOP");
17:         end for
18:   end for
19:     ...
20:   Last steps in AES after execution of rounds
21:    ...
22:   end procedure
```

The method defines the timestamp and obtain time required to execute one round of AES. From the time interval of round execution, number CPU clocks required for execution is calculated. The modulo by 4 for the number of clocks required for a round in AES is calculated. A 'for' loop is included to execute 'NOP' instructions, where it execute from zero to the number that is obtained as a result of modulo operation subtracting from 4. That makes a round to run in multiples of 4. This methodology is titled as 'AES with clock cycles that are multiple of four '

To optimize modulo by 4, the methodology suggests to use Bitwise-AND (&) operation. On any number if we perform Bitwise-AND (&) with 3, it yields the result of modulo by 4. It takes only one CPU clock cycle to execute so it is really an effective way of finding modulo.

Table I shows the number of CPU clocks required to execute some operations:

Operation	CPU Clock cycles required
Addition	1
Assignment	1
Bitwise AND	1
Division	24
Multiplication	3
Subtraction	1

Table I: CPU clock required for various operations [9]

Average is found for a round as follows:

$$AVG_i = (AVG_{i-1}*(i - 1) + CC_i)/i \quad (1)$$

Where AVGi : Average calculated for i^{th} round;
CC_i : Clock count of i^{th} round

This operation involves five operations that is Addition, Multiplication, Subtraction, Division and Assignment.

No. of clock cycles required to find Average

= 1(Assignment) + 3(Multiplication) + 1(Subtraction)
+1(Addition) + 24 (Division)
= 30

The number of clock cycles required just to calculate how many extra clocks are needed in AES with average clock cycles method requires 30 clocks cycles.

No. of Clock cycles required to find multiples of 4

= 1 (bitwise AND) + 1 (Assignment) + 1 (Subtraction)

= 3

Also the number of clocks required to be executed are lesser in this suggested solution than that of AES with average clock cycles. So this shows the effectiveness of the new countermeasure this paper suggests.

IV. Results And Performance Ealuation

Proposed solution, AES with clock cycles that are multiples of 4, has been implemented in Java along with AES without any countermeasure for timing attack and AES with average clock cycles [8]. The results have been recorded and are discussed in the next section.

The three implementations i.e. AES without any countermeasure for timing attack, AES with Average clock cycles and AES with clock cycles multiple of 4 as countermeasure are tested with the 10 various input strings. Keeping the AES without any countermeasure as a benchmark, the results are calculated clocks required for other AES with average clock cycles and AES with clock cycles that are multiple of 4 and the results shown in Table II.

Sr. No.	AES without any security for timing attack	AES with Average Clock Cycles	AES with clock cycles multiple of 4
1	100	738.042	186.85886
2	100	768.267	135.88
3	100	786.668	192.0075
4	100	925.041	182.165
5	100	842.379	165.43
6	100	649.019	144.451
7	100	707.732	156.324
8	100	707.11	166.332
9	100	501.334	175.324
10	100	638.724	170.355

Table II: Results

So the results here clearly indicate that the countermeasure suggested in this paper gives a more efficient substitute to the AES with average clock cycles suggested in [8]. The comparative performance analysis for AES, AES with average clock cycles and AES with clock cycles that are multiple of 4 is shown in Fig. 3 and Fig. 4 in graphical form.

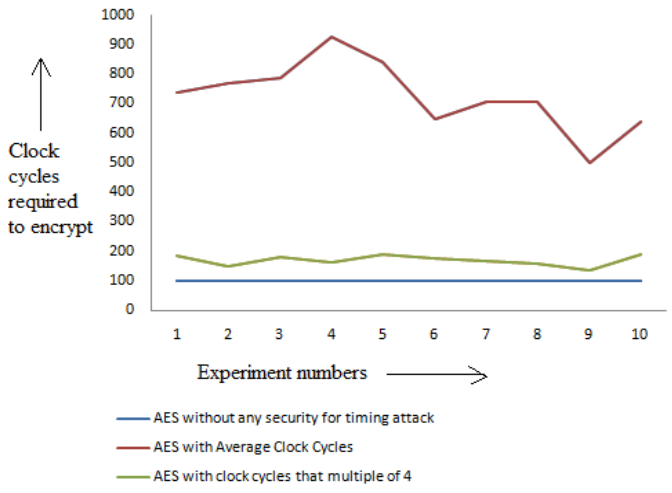

Fig. 3: Performance Graph

Fig. 3 shows that clock cycles were required lesser for AES with clock that are multiple of 4 than AES with average clock cycles. The performance of the algorithm is also tested in a real time network scenario using client-server model and the results are displayed in a graph in Fig. 4.

The performance graphs as shown in Fig. 3 and Fig. 4 has Experiment number on X axis, and clock cycles required to encrypt the input on Y axis. Blue line indicates performance of AES without any countermeasure for timing attack for the sample inputs. The red curve indicates performance of AES with clock cycles that are multiples of 4. The green curve indicates performance of AES with average clock cycles[8].

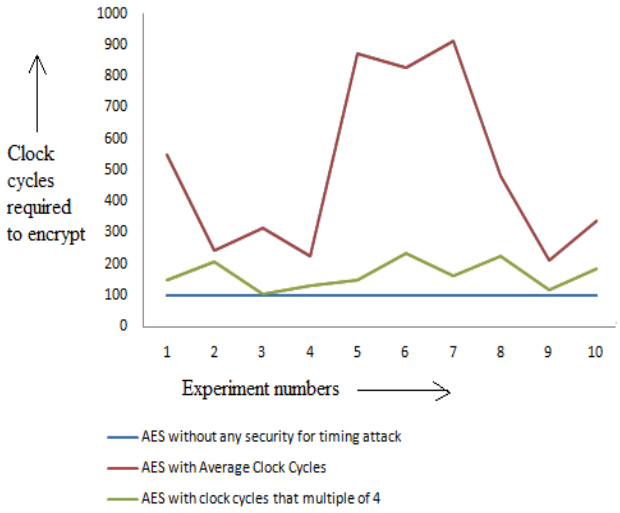

Fig. 4: Performance graph in real time scenario

From the above graphs in Fig. 3 and Fig. 4, it is clear that AES with clock cycles that are multiples of 4 gives better performance over AES with average clock cycles[8].

V. Conclusion

AES is fallible to timing attack. Different countermeasures against timing attack on AES are studied. There are countermeasures against Timing attack which changes behavior of AES by manipulating timing information of AES. But it causes decrease in speed of computation which reduces the performance.

The methodology suggested in this paper, AES with clock that are multiples of 4, takes the multiple of 4 clock cycles technique. The results prove that suggested methodology in this paper has reduced time to encrypt or decrypt using AES with clock cycles that are multiples of 4 over the "AES with average clock cycles".

References

[1] ”What is Adanced Encryption Standards?”, http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard, Dated: 11 July 2015

[2] Janaka Alawatugoda, Darshana Jayasinghe, and Roshan Ragel, Countermeasures against Bernsteins Remote Cache Timing Attack, ICIIS 2011, Sri Lanka.

[3] Daniel J. Bernstein, Cache-timing attacks on AES., Department of Mathematics, Statistics, and Computer Science (M/C 249) The University of Illinois at Chicago, IL 606077045.

[4] ”Side-channel attack”, https://en.wikipedia.org/wiki/Side-channel attack, Dated: 11 July 2015

[5] Paul C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems., CRYPTO 1996, 104113.

[6] ”Side-Channel Analysis”, http://www.microsemi.com/products/fpga-soc/security/side-channel-analysis, Dated: 11 July 2015

[7] Darshana Jayasinghe, Roshan Ragel, and Dhammika Elkaduwe Constant Time Encryption as a Countermeasure against Remote Cache Timing Attacks., ICIAfS, 2012

[8] Udyani Herath, Janaka Alawatugoda, and Roshan Ragel, Software Implementation Level Countermeasures against the Cache Timing Attack on Advanced Encryption Standard., ICIIS 2013, Sri Lanka

[9] Agner Fog, Lists of instruction latencies, throughputs and microoperation breakdowns for Intel, AMD and VIA CPUs,Technical University of Denmark. Copyright 1996 - 2014. Last updated 2014-12-07.

[10] ”Encryption”, https://en.wikipedia.org/wiki/Encryption, Dated: 11 July 2015

[11] https://en.wikipedia.org/wiki/Symmetric-key algorithm Dated: 11 July 2015

[12] David Brumley and Dan Boneh, Remote timing attacks are practical., USENIX Security Symposium. 2003.