Effective Countermeasures for Cache Timing Attack on AES

Dipayan Dev

***Abstract*— This paper describes several software side-channel attacks based on inter-process leakage through the state of the CPU's memory cache. This leakage reveals memory access patterns, which can be used for cryptanalysis of cryptographic primitives that employ data-dependent table lookups. The attacks allow an unprivileged process to attack other processes running in parallel on the same processor, despite partitioning methods such as memory protection, sandboxing and virtualization. In this paper, we propose an algorithm which disables the cache memory .This would make the AES impregnable to cache timing attack.**

***Index Terms*— Side-channel attack, AES, cache memory, security**

I. INTRODUCTION

Traditionally attacks on cryptography are conducted against the math of the cryptographic system which includes differential and linear cryptanalysis. A few categories of such attacks are: cipher text-only, known plaintext, or chosen plaintext attacks. These methods rely either solely on the ciphertext, on both the plaintext and ciphertext, or the ability to define plaintext that is encrypted and the analysis of the resultant cipher text. Today, however it has been realized that encryption devices provide attackers more information than previously presumed. These devices have a tendency to unintentionally reveal more information about the cipher than just the ciphertext. This extra information is known as side channel information's and is defined as "information's that can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the ciphertext resulting from the encryption process [1] .With the ability to gather side channel information's, a new class of attacks called side channel attacks has been developed. These attacks include timing attacks, power consumption attacks, fault analysis attacks, and acoustic attacks. One of the pioneers in this field, Kocher, has pioneered certain timing attacks, simple power analysis attacks, and differential power analysis attacks. Each of these attacks is described. [2, 3, 4.] have implemented them against real systems. It has been said that side channel attacks make certain assumptions about the hardware, the message being sent, or another piece of vital information. This class of attack is important and should be thoroughly researched and attempts be made to mitigate the ability to use this information leakage to help determine key used to encrypt data. When National Institute of Standards and Technology (NIST) held the contest to find a new encryption standard to replace the current Data Encryption Standard (to be termed Advanced Encryption Standard), they were aware of this new class of attacks and considered it in their evaluation of the security of the AES candidates. When evaluating attacks against implementations of candidates, NIST made specific mention of timing and power analysis attacks in of [5]. Additionally, NIST stated [5] that table lookups are not vulnerable to timing attacks. The AES contest winner Rijndael, along with other AES candidates and block ciphers, use large table lookups to increase the throughput of their algorithms. Unfortunately, a major problem with this method is that it allows for a specific timing attack to be carried out against a system that relies on table lookups. The attack obtains the information leaked by cache misses. This paper focuses on a specific implementation of cache timing attack against AES and identifies potential methods in which this type of attack may be mitigated.

Manuscript received August 25, 2013

Dipayan Dev is pursuing M.Tech Degree in Computer Science and Engineering from National Institude of Technology, Silchar

II. CACHE TIMING ATTACK

Cache timing attack – the name speaks for itself. This belongs to a pattern of attacks that concentrates on monitoring the target cryptosystem, and analyzing the time taken to execute various steps in the cryptographic algorithm. In other words, the attack exploits the facts that every step in the algorithm takes a certain time to execute.

Although, the cache-timing attack is well-known theoretically, but it was only until April 2005 that a stout researcher named Daniel Bernstein [6, 7] published that the weakness of AES can reveal timing information that eventually can be utilized to crack the encryption key. This paper explains a successful cache timing attack by exploiting the timing characteristics of the table lookups. Here is the simplest conceivable timing attack on AES. AES software implementations that use look-up tables to perform internal operations of the cipher, such as S-boxes, are the one that are most vulnerable to this attack. For example, the variable index array lookup T0 [k [0] ⊕ n [0]] near the beginning of the AES computation. A typical hacker might think that the time for this array lookup depends on the array index and the time for the whole AES computation is well correlated with the time for this array lookup. As a result, the AES timings leak information about k [0] ⊕ n [0] and it can calculate the exact value of k[0] from the distribution of AES timings as a function of n[0]. Similar comments apply to k[1] ⊕ n[1], k[2] ⊕n[2] etc. Assume, that the hacker watches the time taken by the victim to handle many n's and totals the AES times for each possible n[13], and observes that the overall AES time is maximum when n[13] is, say, 147. Suppose that the hacker also observes, by carrying out experiments with

known keys k on a computer with the same AES software and the same CPU, that the overall AES time is maximum when k [13]⊕n[13] is, say, 8. The hacker concludes that the victim's key k [13] is 147⊕ 8 = 155. This implies that a hacker can easily attack a variable time AES algorithm and can crack the encrypted data and eventually key [6].
Since in many AES algorithms all look up tables are stored in the cache, by putting another thread or some different way, attacker can easily get the encrypted data from the cache.

III. ATTACK MODEL AND STRATEGY

The attacks in this paper assume the computer performing the encryption operation uses cached memory which can be described using a simple model of the cache. A cache is a small, fast storage area situated between the CPU and main memory. When values are looked up in main memory, they are stored in the cache, evicting older values in the cache. Subsequent lookups to the same memory address can then retrieve the data from the cache, which is faster than main memory; this is called a "cache hit."

Complicating matters is the fact that modern caches do not store individual bytes, but groups of bytes from consecutive "lines" of main memory. Line size varies between 32 bytes for a Pentium III and 64 or 128 bytes on more recent Pentium IV or AMD Athlon processors. Since the usual size of AES table entries is 4 bytes, groups of 8 consecutive table entries share a line in the cache on a Pentium III (this value is defined as ∂ in [OST06]). So, for any bytes L, L′ which are equal to the lower $\log_2 \partial$ bits (∂=number of 'lines') (notated as L = L′ in [OST06]), looking up address "L" will cause an ensuing access to "L′ "to hit in cache. We view an AES encryption as a sequence of 160 table lookups to indices l1, l2... l160. `

A "cache collision" occurs if two separate lookups len_i, len_j satisfy $len_i = len_j$. In this situation, lenj should always hit in the cache. If it were the case that $len_i \neq len_j$, then the access to len_j may result in a cache miss if T [len_j] was out of memory prior to the encryption and no previous access fetched it. This should, on the average, take more time as it will require a second cache lookup with non-zero probability. We formalize this assumption:

A. Cache-collision assumption

For any pair of lookups i, j, given a large number of random AES encryptions with the same key, the average time when $len_i = len_j$ will be less than the average time when $len_i \neq len_j$

This assumption rests on the approximation that the individual table lookups in the sequence are effectively independent for random plaintexts, which seems to hold in practice. This assumption greatly oversimplifies many the intricacies of modern caches, as discussed in Appendix A, but is well supported by experimental data as shown in Figure 1. Notice that there is a clear correlation, especially for ≤ 10 collisions, which is where 90% of the data lies. We fit the experimental data with a linear model where the unknowns are defined as bonuses due to collisions between table lookups in the final round, a total of 120 variables. Depending on the mix of the processes running in the background the model explains between 13% and 28% of the variance in the timing data (the results are supported by five-fold cross-validation).

The notion of using collisions in the cache is by no means unique to this paper. Because caches are specifically designed to behave differently in the presence of a collision a non-collision, they are a natural side channel for attacking AES. This general notion has been used in several other attacks on AES [TTMM02, Pag02, TSS+03, Lau05, OST06], we seek to explicitly define the utility of cache collisions as they apply to timing attacks (similar to [Acıı05, NSW06, NS06]).

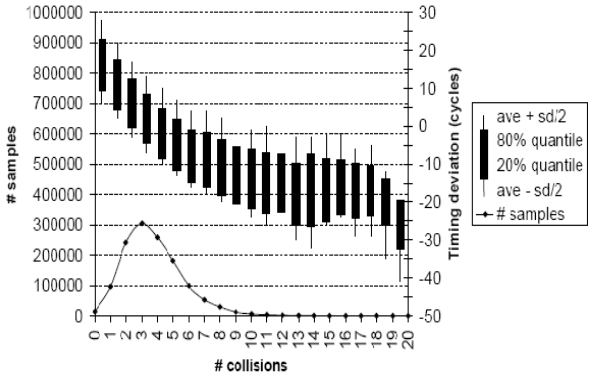

Figure 1: Time deviation vs number of final round cache-collision, Pentium III

IV. COUNTERMEASURES

A. Cache disabling algorithm

One brutal countermeasure against the cache-based attacks is to completely disable the CPU's caching mechanism. Of course, the effect on performance would be devastating. A more attractive alternative is to activate a "no-fill" mode where the memory accesses are serviced from the cache when they hit it, but accesses that miss the cache are serviced directly from memory (without causing evictions and filling). The encryption routine would then proceed as follows:
(a) Preload the AES tables into cache
(b) Activate "no-fill" mode
(c) Perform encryption
(d) Deactivate "no-fill" mode

The section spanning (a) and (b) is critical, and attacker processes must not be allowed to run during this time. However, once this setup is completed, step (c) can be safely executed. The encryption would not be slowed down significantly (assuming its inputs are in cache when "no-fill" is enabled), but its output will not be cached, leading to subsequent cache misses. Other processes executed during (c), via multitasking or simultaneous multithreading, will however incur a performance penalty. Breaking the encryption chunks into smaller chunks and applying the above routine to each chunk would reduce this effect somewhat, by allowing the cache to be occasionally updated to reflect the changing memory work set.

The server makes call to some assembly routine to disable the cache memory after he has accessed it. He first ensures the area of memory he is accessing currently.

B. Flowchart

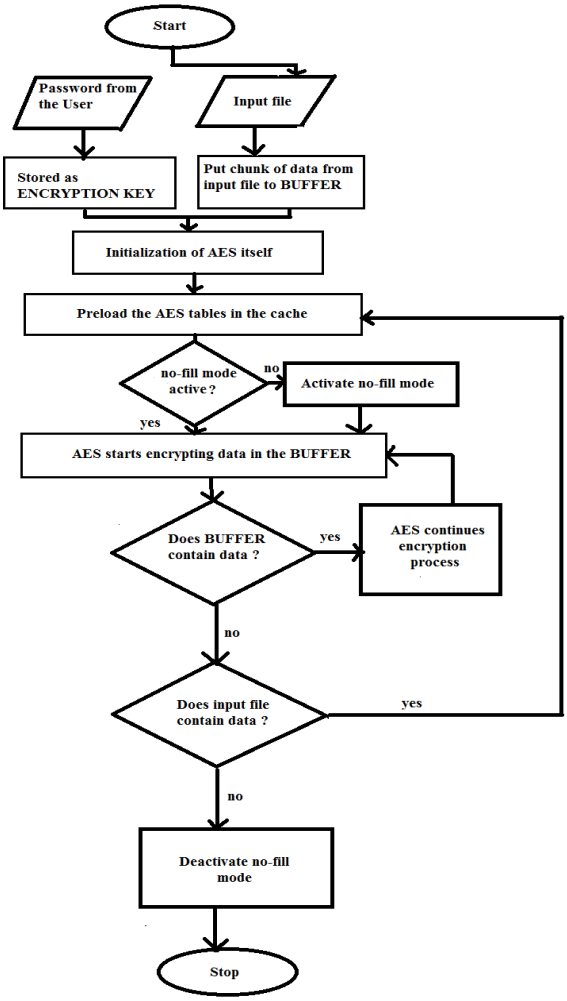

Figure 2: The detailed flowchart of the proposed cache disabling algorithm

V. CONCLUSION

The proposed model narrated the novel idea of cache time attack. A proper synchronizations need to be maintained to avoid cache time attack. However time analysis need to be done.

REFERENCES

[1] Discretix Technologies Ltd. "Introduction to Side Channel Attacks."

[2] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO '96 Proceedings, 1996.

[3] P. Kocher, J. Jaffe, B. Jun, "Introduction to Differential Power Analysis and Related Attacks," December 1998.

[4] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis"December 1998.

[5] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Forti, E. Roback, "Report on the Development of the Advanced Encryption Standard (AES),"

[6] Daniel J. Bernstein, "Cache-timing attacks on AES", The University of Illinois at Chicago, IL 60607-7045, 2005.

[7] Joseph Bonneau and Ilya Mironov, "Cache-Collision Timing Attacks against AES", (Extended Version) revised 2005-11-20

Dipayan Dev is pursuing his M.Tech Degree in Computer Science and Engineering from National Institute of Technology, Silchar He received his B.Tech Degree in Computer Science and Engineering from Bengal College of Engineering and Technology, (WBUT) in 2012. His research interest includes Information Security and Cloud Computing