Breaking security of crypto systems using cache side-channel attack

Bruno Lopes
Master Student
Lisbon, Portugal
brunomsl96@gmail.com

ABSTRACT

The focus of this research pretends to acknowledge the concept and structure of a typical side-channel attack and its variations. In a second phase, to dive into cache side-channel attacks that use timing records as side-channel information, for uncovering the secret key used by a victim cryptographic application. Thus, we implement an attack, based on Prime + Probe strategy, relying on the time differences between L1-D and the other cache levels latency. Our attack requires an unprivileged attack process running in the same CPU core as our victim, using SMT technology. The attack process can choose the plaintext values to input in our victim. Our victim application uses the inputted data to perform an AES encryption using OpenSSL functions. Additionally, we evaluate the success of the performed attack, using the amount of key information discovered, according to different vectors, such as the amount of side-channel information produced.

Author Keywords

Cache; Side-channel attack; Timing information; AES.

ACM Classification Keywords

Security and privacy -> Cryptography -> Cryptoanalysis and other attacks; Side-channel analysis and countermeasures.

INTRODUCTION

Computational systems can unintentionally leak information that in a first instance can be considered as innocent information. However, someone with malicious intentions can analyze it, along with other information, to compromise system's security-critical data. Therefore, side-channel attacks correlate the variations of the leaked (side-channel) information against the respective input/output from a system's cryptographic computation. The main purpose of this attack is to discover a secret such as a cryptographic secret key used by a victim application, compromising this way the system's security-critical data. A general side-channel attack is divided into two phases: online phase, where the side-channel information is extracted, and an offline phase where it is correlated and processed. The leaked information consumed by these attacks can take different forms: the type of sound emitted, the amount of power consumption or the duration time of a computation from a system derived from the different memory level's latencies.

BACKGROUND

Caches

The cache serves as a small piece of high-speed memory, [1], managed by machine's hardware. It keeps the CPU as busy as possible by minimizing the bottleneck of load/store latency to a lower memory level. Modern caches are divided into three levels L1, L2, L3. The CPU looks for the data by the same order: if there is a cache miss in a given cache level, the next level is consulted. The more cache levels accessed, the longer it takes to get the required data.

Each cache level is composed by a number of lines, [5]. Where each line has a specific number of cache lines. Each cache line can store a fixed-size block of information, called cache block.

Simultaneous Multithreading

Simultaneous multithreading [6] is a technology implemented by most modern CPU's that allows two hardware threads to run in the same CPU core at the same time. Consequently, these two threads share the same group of memory hierarchy levels, which includes L1, L2 and L3. Intel CPU's often references this technology by" Hyperthreading" technology.

CPU Dynamic Frequency Scaling

It's a technique employed in modern architectures that allow the CPU's frequency to adapt "on the fly" to the actual computation need, [7]. It conserves power and reduces the amount of heat produced, since in moments it's not required a significant amount of computational effort, the CPU's underclocks and consequently saves energy.

AES

The Advanced Encryption Standard, [8], is a block cipher algorithm using symmetric keys. This algorithm operates at a 128-bit block at a time, accepting a 128, 192 or 256-bit key and performing 10, 12, 14 rounds respectively. The idea behind the AES algorithm is to pick 16-byte blocks of the input each time, also known as state. This state will receive a set of key dependent transformations and substitutions for each round generating a final state, which represents the desired output.

The set of transformations performed in each round are:
SubBytes, which substitutes one byte by a different predefined one using a nonlinear function.
ShiftRow, shifts the 2nd, 3rd and 4th row of the state by 1, 2, 3 positions to the left respectively.
MixColumn, each column of the state matrix becomes its multiplication in a GF(256).
AddRoundKey, where it's applied an exclusive OR operation between the state and the respective round key.

AES 128-biy key encryption pseudocode:

```
state = plaintext
AddRoundKey(state, roundkey[0])
For round in [1;8]:
        SubBytes(state)
        ShiftRows(state)
        MixColumn(state)
        AddRoundKey(state, roundkey[round])

SubBytes(state)
ShiftRows(state)
AddRoundKey(state, roundkey[9])
```

AES implementations are true reproductions of the AES algorithm with some modifications that enhance the whole process performance. Most of these modifications consists on replacing some transformations by a lookup on a pre-computed static table.
The most popular implementation are S-box and T-box:
S-box replaces the SubByte transformation by a lookup to a table called S-box.
T-box makes a lookup on a T-box table instead of performing the SubBytes and MixColumn transformations.

STATE OF THE ART

Side-channel attack

A side-channel attack aims to discover a secret in the system for the advantage of an attacker. The most common example of a secret is the key used by a victim in order to perform a cryptographic computation over some information for local storage or network dispatch purposes, [4].

A side-channel attack strategy relies on extracting information of the system that is not meant to be leaked. This information is then cryptoanalysed achieving partially or entirely our goal. Given the procedure of a typical side-channel attack, we may easily divide it in two different phases:

- An online phase, where the attacker needs the victim to be performing some cryptographic computation for extracting the side-channel information.
- An offline phase, which no longer requires the presence of the victim, and where the side-channel information is processed (cryptoanalysed).

A side-channel attack can use different forms of side-channel information, like timing [4][3], power consumption [2] or even acoustic information.

Most known timing side-channel attacks

Evict + Time Attack

1) Attacker triggers a victim encryption execution using a known random plaintext
2) Attacker evicts a specific line of the targeted cache level
3) Attacker trigger a victim encryption execution using the same plaintext and times it

The triplet extract from the previous process is <plaintext p; line evicted l; duration d>. It's up to the attacker to repeat the same process again and again, gathering triplets having the same or not attack input. The group of all the triplets acquired correspond to the side-channel information.

For the lines with higher 2nd encryption duration: It means the table blocks mapped by the same respective lines were required during the 2nd encryption, always causing a cache miss, since they were evicted from the cache in the line eviction phase, delaying the overall encryption time.

For the rest of the lines: It means the table blocks mapped by the same respective lines were not required during the 2nd encryption, causing no cache miss, and thus not penalizing the overall encryption time.

AES specification equations that generate the 1st and 2nd round state bytes are relatively simple equations that tell the relation between plaintext information with key information and table elements. On the other side, the attacker is aware of the plaintext and the used table blocks (and consequently the unused ones). This allows to the attacker to exclude the key values that when placed on the equations they access table blocks detected as blocks from lines with a low 2nd enc. duration. The remaining key bytes not excluded represents the key discovered by this process.

Prime + Probe

1) Attacker loads a cache sized data array
2) Attacker triggers a victim encryption execution using a known random plaintext
3) Attacker fills and times each line with array data

From the previous process, it's created some triplets in the form of: <plaintext p; line evicted l; duration d>. It's up to the attacker to repeat the same process again and again, gathering triplets having different attack input.

The meaning of the side-channel information as well as its crypto-analysis is performed likewise in Evict + Time attack.

Flush + Flush

1) Attacker performs a clflush instruction using any table block address as argument
2) Attacker triggers a victim cryptographic execution using a known random plaintext
3) Attacker performs and times a clflush using as argument the same first table block address

There are extracted some triplets with the following format: <plaintext p; table block evicted t; duration d>. It's up to the attacker to repeat the same process again and again, gathering triplets having different plaintexts, [9].

The meaning of the side-channel information as well as its crypto-analysis is performed likewise in Evict + Time and Prime + Probe attack.

Countermeasures

Countermeasures solutions are divided in 2 different groups: hardware-based solutions [10] [11] and software-based solutions [4] [3].

Hardware Based

• Static Partitioning Cache: It separates the cache for the victim and for the attacker.

• Partition Locked Cache: A cache where protected cache blocks cannot be replaced by nonprotected blocks.

• Non-monopolizable Cache: separated by the cache's way.

• Fully Associative Cache using random as replacement policy.

• Random Eviction Cache: Typical cache that evicts a random block when a certain number of memory accesses is achieved.

Software Based

• Consider using alternative AES table architectures of reduced size: S-box (256-byte table); S-box table and a table composed by 2×S-box values (two 256-byte tables); T0 (1024-byte table) and recreate T1,T2,T3 by doing rotations, T0...T3 compressed into one table with non-aligned lookups (2048-byte).

• Adding extra noise to an AES computation by doing extra table accesses e.g.: Performing a second encryption in order to interfere with the side-channel information leaked by the main encryption computation.

• Cache state normalization by loading all the lookup tables right before an encryption start, could easily defeat the big majority of timing attacks: Evict+Time and Prime+Probe based attacks.

• Replace the table lookups by a set of logical operations in order to get the same result.

• Use lookup tables and place them on registers, in the case there's enough room for them.

SOLUTION PROPOSAL

The implemented attack corresponds to a cache side-channel attack that targets AES implementation in particular T-box tables. This attack is fueled by timing information which means different access data computations are performed and respective time is captured. Then, each computation time tells the attacker whether or not the cache levels below L1-D were accessed or not. For this reason, our attack aims at the L1-D cache level. Unlike most related side-channel attacks, this one does not require the previous knowledge of the T-box mapping on L1-D. Likewise other related attacks, our online phase makes use of the prime + probe technique, and during the offline phase it exploits the AES equations that generate the 1st and 2nd round state bytes. This attack can uncover the whole AES victim's secret key.

Problem

First, we acknowledge the existence of a victim application that ciphers a plaintext that can be inputted by a user or a program. This application contains an inner secret key that is passed along the received plaintext to call an AES encryption function. That encryption function belongs to the OpenSSL application, which is a cryptographic library containing a large set of cryptographic functions to be used by external users or programs. Victim application is a custom software and it can be compared to applications such as dmcrypt1 or a simple virtual private network software. The attacker application materializes the strategy used to uncover the secret key from the victim.

In other words, our attacker application can have two distinct interactions: The first one, resides on the ability to trigger victim application executions, inputting any desired plaintext. The second, resides on calling OpenSSL encryption functions, inputting any desired plaintext and any key value, likewise it happens with the victim application. The gathered side-channel information is then crypto-analysed, revealing the victim's secret key value.

- Attacker application is composed by the following programs: atk.c , atk_enc.c and crypto.py.
- Victim application is composed by the following programs: vic_enc.c.

Attack Structure

This section briefly describes each phase of the attack.

1) 1st phase - L1-D T-Box Mapping phase: the attacker application performs OpenSSL encryptions to discover which L1-D lines map each T-box element.
2) 2nd phase - Online phase: the attacker application triggers victim executions inputting a given plaintext in order to extract side-channel information.
3) 3rd phase - Offline phase: attacker application gathers the information from 1) and 2) and exploits the AES equations that generate the state bytes for the 1st and 2nd round in order to uncover the victim application secret key

OpenSSL

The OpenSSL is the system cryptographic software that contains the AES implementation that can be used by applications and users. For this attack, we are using the OpenSSL 0.9.8, containing 4 encryption T-box tables, each containing 256 4-byte elements (4KB of table information). This T-box technology keeps the tables static in memory, which means different calls to OpenSSL will always load the tables into the same L1-D lines. Additionally, this type of T-box implementation positions its tables consecutively and contiguously in memory. In practical terms, it means T0, T1, T2 and T3 tables are placed in memory by the same order. The targeted encryptions are the AES 128-bit encryptions and the encryption mode is the Electronic CodeBook ECB.

Hardware Requirements

The list of the physical demands that have to be assured in order to provide the proper environment for our attack is:

1) Application processes run in the same CPU core: It's required for both attacker and victim applications run in the same core , this is possible either using a uni-core system, or force both threads to run on the same core on a multi-core CPU.
2) Simultaneous Multithreading active: The CPU core where the applications run require to have the SMT technology enabled.

Measurement Concept

A measurement represents an atomic process that allows the attacker to retrieve some information from the victim. Both 1st and 2nd attack phases resort on the usage of measurements aligned with each one strategy.

The measurement concept considers two distinct processes, running at the same time in the same CPU core:

1) *Process 1* performs the same AES encryptions of a given plaintext p and key k
2) *Process 2* continuously fills each L1-D line by loading the necessary array cache blocks until *process 1* finishes.

After the *process 2* fills a specific line with its data, process 1 will access some T-box elements evicting or not some cache blocks of that specific line. At this moment, the state of this L1-D line can either contain only *process 2* cache blocks or *process 2* and *process 1* cache blocks. Thus, process 2 performs once again a line fill and times it. The time resulted from the previous computation tells us if the process 1 loaded T-box cache blocks in that line:

- If the time is low: we conclude there were no cache misses during the process of filling that specific line. Which means no *process 2* cache blocks were evicted and for that reason no T-box elements that are mapped in that line were not required by the *process 1*.
- If the time is high, we conclude there was at least one cache miss during the process of filling that specific line. Which means at least one *process 2* cache block was evicted and for that reason at least 1 T-box cache block that is mapped in that line was required by the *process 1*.

Expanding this process for every L1-D line, it's possible to understand which lines are being used by *1* and those who are not. Thus, each measurement produces a timing array, where each index corresponds to the L1-D line and each element to the respective timing value acquired.

We are using PAPI 6.0.0 for capture the total number of clock cycles giving us the proper timing resolution the situation demands.

1st Phase - L1-D T-box Mapping

L1-D T-box mapping phase serves to extract critical information about the system to be used during the crypto-analysis phase. In particular, which L1-D lines map the different AES software T-box elements. The OpenSSL version we are using in this attack keeps the respective T-box table static in memory. Once the tables are discovered they still mapped by the same lines of L1-D cache throughout different executions. Note that most side-channel attacks do not solve this problem. Instead they simply assume the knowledge of T-box mapping on the respective cache level.

atk.c process behaves like the *process 2* and atk_enc. c process like *process 1* using the vocabulary from Measurement Concept.

We perform a set of measurements, half inputting a plaintext with value P1 and the other half with a plaintext with value P2. Each P1 measurement generates a timing array. We average all the P1 measurements timing arrays, creating an averaged p1 array. The same is process is done with the p2 measurements, creating an averaged p2 array.

We pick the p1 and p2 averaged arrays and compute the respective difference array. The computation corresponds to the operation (p2 − p1) by the line. This very array let the attacker know every line that maps each table element.

2nd Phase – Online Phase

The purpose of this phase relies on producing enough side-channel information to be consumed by the crypto-analysis phase. It's performed several measurements, where, atk.c process behaves like the *process 2* and vic_enc.c process like *process 1*. The plaintext inputted is aligned with the plaintext configuration p3.

The attacker's result from each measurement corresponds to a timing array with the size of the number of L1-D lines, which is 64. The respective measurement plaintext and timing array of a given measurement is stored inside a meas#i.out file, where i represents the measurement index number. These files correspond to our side-channel information that will be used during our offline phase

3rd Phase – Offline Phase

This phase can be performed away from the victim system and will consume the information acquired in the previous phases in order to uncover the victim's secret key.

1st and 2nd round equations from the Rijndael specification, tell the attacker the relation between 3 types of information: key, plaintext, table element. So, in the 1st round attack phase, the 1st round equations are used for uncovering some victim's key bits. Next, the 2nd round attack phase uses the 2nd round equations for uncovering the remaining key bits. In a single sentence the 1st round attack phase only serves to decrease the complexity of the 2nd round attack phase. The more key information discovered in the 1st phase the faster the 2nd phase becomes.

EVALUATION AND RESULTS

We then proceed in listing a set of criteria vectors that will quantify the success rate of a given attack containing certain characteristics.

Criteria vectors are:

1. Number of halves of key bytes discovered
2. The right/wrong association between T-box elements and respective L1-D mapping lines

There are some attack variables whose definition can only be extracted empirically, which are:

1. It, I represent the number of line fills in the measurements (in other words, they regulate the timing resolution) from 1st attack and 2nd attack phase, respectively.
2. Nt, N represent the number of measurements performed in the 1st attack and 2nd attack phase, respectively.

Each test consists in executing different attacks with different characteristics and extract the respective score according to our criteria. The tests are:

1. Test 1 – Performs different attacks varying Nt and It variables value and acknowledge each resultant vector 2 score.
2. Test 2 – Performs different attacks varying N and I variables value and acknowledge each resultant vector 1 score.

We also plot these test results, in order to capture an idea of the success evolution according to the different attacks performed.

CONCLUSION

In this work, we implemented an enhanced cache side-channel attack targeting a T-box based AES implementation on the L1-D cache level, based on the Prime + Probe strategy. Additionally, we tested several attacks with the purpose to understand the attack's limitations in terms of number of samples required (number of measurements) and other technical variables. Plus, several concepts were recovered related to the attack. Concepts in the scope of the cache functionality and structure, cryptographic algorithms, cryptographic implementations, types of cache side-channel attacks on AES.

REFERENCES

1. M. Kowarschik and C. Weiß, "An overview of cache optimization techniques and cache-aware numerical algorithms," in Algorithms for Memory Hierarchies — Advanced Lectures, volume 2625 of Lecture Notes in Computer Science. Springer, 2003, pp. 213–232
2. G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo, "Aes power attack based on induced cache miss and countermeasure," in Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume I - Volume01, ser. ITCC'05.USA: IEEE Computer Society, 2005, p. 586-591.
3. D. J. Bernstein, "Cache-timing attacks on aes," Tech. Rep., 2005.
4. D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case ofaes," in Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics65 in Cryptology, ser. CT-RSA'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 1–20.
5. D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware Software Interface ARM Edition, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.
6. N. Tuck and D. M. Tullsen, "Initial observations of the simultaneous multithreading pentium 4 processor," in Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques,

ser. PACT ’03.USA: IEEE Computer Society, 2003, p. 26.

7. W. Bao, C. Hong, S. Chunduri, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, and P. Sadayappan, “Static and dynamic frequency scaling on multicore cpus,”ACM Trans. Archit. Code Optim., vol. 13,no. 4, Dec. 2016.
8. J. Daemen and V. Rijmen, The Design of Rijndael. Berlin, Heidelberg: Springer-Verlag, 2002.
9. D. Gruss, C. Maurice, K. Wagner, and S. Mangard, “Flush+flush: A fast and stealthy cache attack,” in Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721, ser. DIMVA 2016. Berlin, Heidelberg: Springer-Verlag,2016, p. 279–299.
10. A. Canteaut, C. Lauradoux, and A. Seznec, “Understanding cache attacks,” INRIA, Research Report RR-5881, 2006
11. Z. He and R. Lee, “How secure is your cache against side-channel attacks?” in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO-50 ’17.New York, NY, USA: Association for Computing Machinery, 2017, p. 341–353.