# Cache-timing Attacks on AES

Wei Liu, Ariel Di Segni, Ye Ding, Teng Zhang, *Polytechnic Institute of New York University*

***Abstract*—This paper presents several tests to demonstrate the effectiveness of cache-timing attacks against AES on a newest processor: Intel Atom N2800. Cache-timing attack is a type of side channel attacks that takes advantage of the information leaked by encryption devices. We also implement the attack on other two Intel processors: Intel Core i5 and Intel Xeon E5410 to compare the immunity level of the Intel Atom N2800.**

***Index Terms*—side channels, cache-timing attacks, AES**

## I. Introduction

In this project, we give a demonstration of cache-timing attacks against Advanced Encryption Standard (AES).

Timing attacks are a type of side channel attacks, a new class of attacks that was developed with the realization that encryption devices give the attacker more information than previously thought. Side channel attacks use information from the physical implementation of a cryptosystem, such as its timing information, power consumption, electromagnetic leaks, or even sound, which are exploited to break the system. Timing attacks watch the data movement in and out of the CPU, or of the memory (or cache) and determine a secret key (such as an AES key) by analyzing the variations in the time taken to execute cryptographic algorithms.

The information that can be gathered with these attacks is referred to as side channel information, which is defined as "information 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] Side channel attacks can provide a number of valuable assumptions, such as information about the hardware, the message that is sent, and more. Therefore, it is important to study them and try to diminish the ability to use the information leakage that comes with encryption devices to find the keys that have been used for the encryption [2].

The existence of side channel attacks was already known and taken into consideration when AES was developed to replace the old Data Encryption Standard by the National Institute of Standards and Technology (NIST) [3]. The AES candidates that were chosen, after the evaluation of their security, all relied on large table lookups to improve the performance of their algorithm. However, this allowed specific timing attacks to be carried out against systems that use table lookups, taking advantage of the information leakage caused by cache misses. This work, along with the research that is currently being done in this field, attempts to offer results that will help finding ways to mitigate this type of attack.

21 May 2013

Wei Liu, Department of Electrical Engineering, Polytechnic Institute of New York University, Brooklyn, NY 11201 USA (e-mail: wl964@students.poly.edu).

Ariel Di Segni, Department of Computer Engineering, Polytechnic Institute of New York University, Brooklyn, NY 11201 USA (e-mail: ads606@students.poly.edu).

Ye Ding, Department of Computer Engineering, Polytechnic Institute of New York University, Brooklyn, NY 11201 USA.

Teng Zhang, Department of Computer Engineering, Polytechnic Institute of New York University, Brooklyn, NY 11201 USA (e-mail: tz537@students.poly.edu).

## II. Advanced Encryption Standard

AES is short for Advanced Encryption Standard, which was in 2001. It is a specification for the encryption of electronic data. It is becoming very popular during recent years and has even been adopted by the U.S. government.

AES has different round keys which are derived from a cipher key. The deriving process uses a schedule called Rijndael's key schedule. In the initial round, every byte of the plaintext does a xor operation with the corresponding round key. By the end of every round, AES will add a round key to the next round.

For each round, there exist three steps: SubBytes, ShiftRows and MixColumns. In the SubBytes step, every byte is substituted according to the lookup table. Then AES implements ShiftRows, in which every row of the state is shifted cyclically by several columns. The first row keeps no change. The second row is shifted by 1 column, the third row by 2 and the fourth row by 3. After that, AES implements the MixColumns step. Every byte in a column is taken as a coefficient of a polynomial over $GF(2^8)$. AES then multiplies this polynomial with another polynomial $c(x)= 0x03 \cdot x3 + x2 + x + 0x02$. The coefficients of the result make up the new column of the state. It shall be mentioned here that the final round takes no MixColumns step.

## III. Cache-timing Attacks on AES

Although the encryption method of AES is complex, it can still be used for information leakage and consequently as an attack to break a system. Bernstein [1] has found that many block ciphers, specifically AES, may leak timing information during cache hits or misses. Then he put forward an attack named cache-based timing attack, which can reconstruct the key by observing the data flow of different levels of cache. The AES algorithm employed four 1024-byte tables: T0, T1, T2, and T3. In order for AES to get enough speed, those 4KB information should be stored in the cache. However, even for a large enough cache, some other processes will eventually cause the cache misses. Those misses are the basic reason that causes the encryption to occur at a variable rate.

We choose the method of cache-timing attack to handle AES. As mentioned above, two constant 256-byte tables S and S' and four constant 1024-byte lookup tables T0, T1, T2, and T3 expanded from S and S' are used for AES implementation. The basic idea of AES is utilize the round key which is generated from T0, T1, T2 and T3 we have mentioned to do the XOR with input. So change the input bit may lead to different timing of encryption. That is the basic idea of Bernstein's attack. We believe that the total AES time is related to time of array lookup and lookup table indices dependent on secret key material.

We will attack from the array lookup T [plaintext xor round key]. We will try to find out the value of k[0] by observing the AES timing and deducing k[0] as a function of plaintext n[0]. If this approach works on k[0] and n[0], we will continue to find out k[1], k[2], k[3] and all others as function of n[1], n[2], n[3] and so on. So the most important work we need to do is to watch the time spent on n's. For example, if we try to find out k[4] which is the $5^{th}$ byte of key, we will watch the total AES times for every n[4] where n[4] is $5^{th}$ byte of plaintext. Then we will find out maximum AES time among these n[4]s when n[4]=x. Then we will use another computer using same CPU and same AES the key of which is known by us. We will try to get the maximum value of AES time when k[4] XOR n[4]=y. At last, we deduce k[4] equals x XOR y. By this method, we will continue to find out k[1], k[2], k[3] and all others.

Generally, we can divide the attack as three steps: preparation or profiling, attack, and analysis. First, this attack begins with collecting a lot of timing information to build the baseline level of timing information, just like a profile. So during this step, we will send random 400, 600, 800 byte packets to a sever using the AES encryption algorithm. And in this step, we just set the AES key in sever is all zeros, so that the information is recorded to a file to determine the profile of the system as it reacts to the different size packets. Then, we do the same thing like in the profiling step but this time we set a random secret key instead of a known key. Then we begin to send many 400, 600 and 800 bytes packet to collect the timing variations. The number of packets used by Bernstein [1], was $2^{27}$ 400-byte packets, $2^{25}$ 600-byte packets, and $2^{25}$ 800-byte packets.

The final step is getting the correlation of the baseline profile to the date that we collected with the unknown key. This analysis outputs potential keys that produce large correlations and combined with output from the sever, we can get the whole potential key.

## IV. Related Work

Cache-timing attacks were first introduced by Bernstein [1] in 2005, who showed how to complete an AES key recovery from known-plaintext timings of a network server on another computer. He used a Pentium III as targeted server and conjectured that the same attack could be carried out against software running on many other CPUs. In the last part of his paper, he encourages future work by applying the same attack to other CPUs and other severs. He implies that with the difficulty of writing constant-time high speed AES software, it would be surprising if any AES-based server were immune.

Salembier [2] verified Bernstein's attack with serial experiments using a different environment from the original one and successfully extracted the AES key. Also, after a thorough and in-depth analysis of potential real-world uses of this attack, he provided ideas for starting points in prevention of the attack. He proposed that the final software solution by BGNS appeared to be the best. When coupled with the idea to incorporate the process with the operating system, it appeared to be the finest solution.

More recently, Mowery [3] showed that the AES cache-timing attack is no longer feasible now, for several reasons. First, the AES-NI instruction set moves AES data structures out of the cache. Secondly, multicore processors with per-core L1 and L2 caches are now used. He suggests that any data-cache timing attack against x86 processors that does not somehow subvert the pre-fetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail, to say nothing of the difficulties imposed by multicore processors and hardware AES implementations.

Following Bernstein's suggestion to try his attack on different processors and taking into account Mowery's criteria for cache-timing attack prevention, we tested the attack on a different processor, the Intel Atom N2800, in order to determine whether it is immune against cache-timing attacks.

## V. Our Work

We tested cache-timing attacks on different AES-based servers, by attempting to extract a complete AES key from a server on another computer. The attack was demonstrated on three different servers so that we were able to compare the results among them. The goal of this demonstration was to show the level of cache-timing attack immunity of the Intel Atom N2800, and we used the other two processors to compare different levels of immunity. A deep understanding of the underlying reasons behind the discrepancy in the results obtained on different platforms can shed light on the vulnerabilities to cache-timing attacks and, as a result, on how the new processors can ensure an enhanced protection against them.

### *A. Testing Environments*

#### *1) Test 1 – Intel Core i5*

Server:
Mac OS X
Intel Core i5 2.5 GHz, 4 GB RAM
L2 Cache per core: 256 KB, L3 Cache: 3MB

gcc version 4.2.1
OpenSSL 1.0.1e 2 Feb 2013

Attacker:
Mac OS X
Intel Core 2 Duo 2.26GHz, 2GB RAM

gcc version 4.2.1
OpenSSL 0.9.8r 8 Feb 2011

**Figure 1. Testing Environment of Test 1**

*2) Test 2 – Intel Xeon E5410*

Server:
Linux Gnome 2.16.0
Intel Xeon E5410 2.33GHZ, 8GB RAM
L2 Cache Size 6MB

gcc version 4.1.2
OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008

Attacker:
Linux Gnome 2.16.0
Intel Xeon E5410 2.33GHZ, 8GB RAM
L2 Cache Size 6MB

gcc version 4.1.2
OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008

**Figure 2. Testing Environment of Test 2**

*3) Test 3 – Intel Atom N2800*

Server:
Linux Ubuntu 11.10 Desktop
Intel Atom N2800 1.86GHZ, 2GB RAM
L1 Cache (per core): 56 KB
L2 Cache (per core): 512 KB

gcc version 4.6.1
OpenSSL version 1.0.0e 6 sep 2011

Attacker:
Linux Gnome 2.16.0
Intel Xeon E5410 2.33GHZ, 8GB RAM
L2 Cache Size 6MB

gcc version 4.1.2
OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008

**Figure 3. Testing Environment of Test 3**

*B. Overview of Tests*

*1) Test 1*

In the first test of the timing attack on AES, we used a 2.26GHz Intel Core 2 Duo laptop computer running Mac OS X as the attacker and a 2.5 GHz Intel Core i5 laptop computer running Mac OS X as the targeted server. Following Bernstein's description, each of our tests was composed of three parts: preparing the attack, carrying out the attack, and analyzing the attack.

In the preparation, $2^{16}$ packets were sent to the server using a known AES key, that is, all zeros. After the preparation (about 160 minutes), the program printed many lines such as the following:

**14 400 250 65545 3106.435 525.100 -0.316 2.051**

The above line indicates that the server was sent 65545 (that is, around $2^{16}$) 400-byte packets having $n[14] = 250$. Those packets were handled in 3106.435 cycles on average, with a deviation of 525.100 cycles. Compared to the average over all choices of $n[14]$, the average for $n[14] = 250$ was 0.316 cycles lower. The number 2.051 is an approximate estimate of the deviation in this difference. The other lines show similar information for other choices of $n[14]$ from 0 to 255. For example, the top cycle for $n[14] = 8$ was 1.382; for $n[14] = 25$ it was 3.035; for $n[14] = 120$ it was 1.330.

Figure 4 shows these numbers for all choices of $n[14]$.

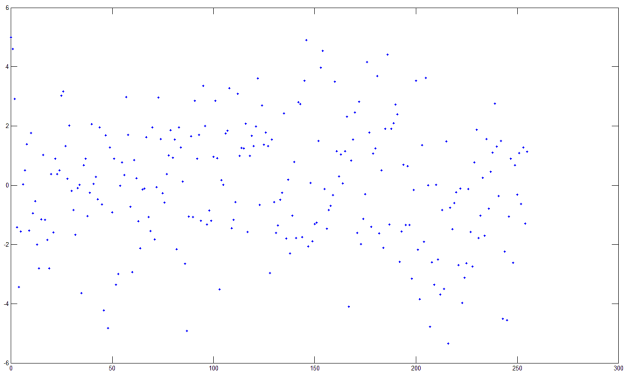

**Fig. 4**. How $n[14]$ affects OpenSSL AES timings for $k = 0$ on an Intel Core i5 inside the targeted server (with operating system Mac OS X). The horizontal axis is the choice of $n[14]$, from 0 through 255. The vertical axis is the average number of cycles for about $2^{16}$ 400-byte packets with that choice of $n[14]$ minus the average number of cycles for all choices of $n[14]$.

The attack was carried out in a similar fashion as the preparation, by sending a secret 16-byte key taken from the cryptographic pseudorandom number generator of the server's operating system instead of a known AES key (all zeros).

After $2^{16}$ random packets – about 160 minutes – we analyzed the attack by correlating the lines obtained from the preparation (study.400) and from the attack (attack.400) 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. For this test, 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 was not successful.

*2) Test 2*

In the second test, we used two 2.33GHZ Intel Xeon E5410 desktop computers running Linux Gnome 2.16.0, one as attacker and one as server. The preparation, attack, and analysis were carried out as in test 1. During the same time as test 1 – 160 minutes for the preparation and 160 minutes for the attack – we were able to send a much bigger number of packets, $2^{20}$. After the preparation, the program printed many lines such as the following:

**14 400 250 1050900 1587.038 115.648 -0.120 0.113**

The above line indicates that the server was sent 1050900 (that is, around $2^{20}$) 400-byte packets having $n[14] = 250$.

Figure 5 shows the values of the choice of $n[14]$, from 0 through 255, and the top cycles (the average number of cycles

for about $2^{20}$ 400-byte packets with that choice of $n[14]$ minus the average number of cycles for all choices of $n[14]$.

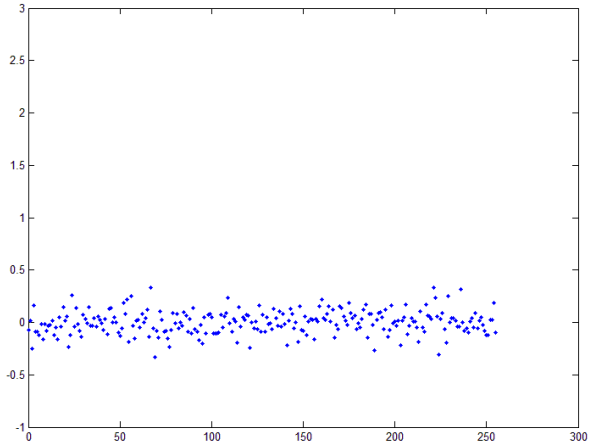

**Fig. 5**. How $n[14]$ affects OpenSSL AES timings for $k = 0$ on a Intel Xeon E5410 inside the targeted server (with operating system Linux Gnome 2.16.0). The horizontal axis is the choice of $n[14]$, from 0 through 255. The vertical axis is the average number of cycles for about $2^{20}$ 400-byte packets with that choice of $n[14]$ minus the average number of cycles for all choices of $n[14]$.

For this test, the range of possibilities was 256 for all the correlations like test 1, which indicates that the correlation was not extracting any useful information about the secret key, showing that the attack to the Intel Xeon E5410 was not successful as well.

*3) Test 3*

In the third test of the timing attack on AES, we used a 2.33GHZ Intel Xeon E5410 desktop computer running Linux Gnome 2.16.0 as the attacker and a 1.86GHZ Intel Atom N2800 desktop computer running Linux Ubuntu 11.10 Desktop as the targeted server. The preparation, attack, and analysis were carried out as in test 1 and 2. During the same time as the previous two tests – 160 minutes for the preparation and 160 minutes for the attack – we were able to send $2^{17}$ packets. After the preparation, the program printed many lines such as the following:

**14 400 250 131426 6193.608 122.453 -2.823 0.338**

The above line indicates that the server was sent 131426 (that is, around $2^{17}$) 400-byte packets having $n[14] = 250$.

Figure 6 shows the values of the choice of $n[14]$, from 0 through 255, and the top cycles (the average number of cycles for about $2^{17}$ 400-byte packets with that choice of $n[14]$ minus the average number of cycles for all choices of $n[14]$.

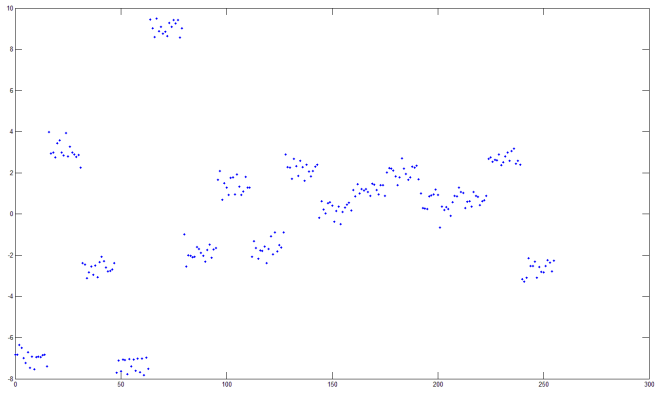

**Fig. 6**. How $n[14]$ affects OpenSSL AES timings for $k = 0$ on an Intel Atom N2800 inside the targeted server (with operating system Ubuntu 11.10 Desktop). The horizontal axis is the choice of $n[14]$, from 0 through 255. The vertical axis is the average number of cycles for about $2^{17}$ 400-byte packets with that choice of $n[14]$ minus the average number of cycles for all choices of $n[14]$.

For this test, 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.

### *C. Timing of Tests*

Tables 1(a), (b), and (c) show the time that it took us to prepare and carry out the attacks for each test.

*1) Test 1 – Intel Core i5*

| | study.400 | attack.400 | 9.5h |
|---|---|---|---|
| | Time (min) | Time (min) | Total time (min) |
| $2^{14}$ packets | 40 | 40 | 80 |
| $2^{15}$ packets | 80 | 80 | 240 |
| $2^{16}$ packets | 160 | 160 | 560 |

**Table 1(a).** Timing for test 1.

*2) Test 2 – Intel Xeon E5410*

| | study.400 | attack.400 | 9.5h |
|---|---|---|---|
| | Time (min) | Time (min) | Total time (min) |
| $2^{18}$ packets | 40 | 40 | 80 |
| $2^{19}$ packets | 80 | 80 | 240 |
| $2^{20}$ packets | 160 | 160 | 560 |

**Table 1(b).** Timing for test 2.

*3) Test 3 – Intel Atom N2800*

| | study.400 | attack.400 | 9.5h |
|---|---|---|---|
| | Time (min) | Time (min) | Total time (min) |
| $2^{15}$ packets | 40 | 40 | 80 |
| $2^{16}$ packets | 80 | 80 | 240 |
| $2^{17}$ packets | 160 | 160 | 560 |

**Table 1(c).** Timing for test 3.

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Intel Xeon E5410 | 256 | 13 | c3 | 55 | 00 | 36 | bc | 96 | a1 | 02 | b3 | 40 | 50 | 48 | f9 | 70 | 6c | 78 | ... |
| Intel Core i5 | 256 | 13 | 38 | 05 | e5 | 9f | 82 | 12 | 18 | 1f | 7f | 73 | b3 | 08 | 03 | 1d | 71 | e3 | ... |
| Intel Atom N2800 | 112 | 13 | 33 | 3c | 3e | 3b | 31 | 36 | 3d | 30 | 32 | 38 | 34 | 37 | 3f | 39 | 35 | 3a | ... |

**Table 2.** Correlation results.

## VI. Results

Table 2 shows the correlation results of the three tests that we demonstrated. Based on these results, we know that the secret *k*[13] was one of 256 in the Intel Xeon E5410 and in the Intel Core i5 and one of 112 in the Intel Atom N2800. This shows that the cache-timing attack on AES does not seem to be successful against two Intel processors, Intel Core i5 and Intel Xeon E5410, and is somewhat successful against the Intel Atom N2800, even though the attacker still managed to narrow the range of possibilities of a secret key that was trying to extract by a very limited number (from 256 to 112) and taking a very long time to do so. All the attacks that we tested took a much longer time than Bernstein's original attacks. Based on this number, we can estimate the level of the immunity of the Intel Atom N2800 as 96% compared to the Intel Core i5, which appears to be completely immune to the attack.

## VII. Explanations

The final result of our project seems to show that the cache timing attack is not successful on Intel Core i5 and Intel Xeon E5410, and it is somewhat successful on the Intel Atom N2800, which still has a relatively higher level of immunity against cache timing attack compared to the Pentium III used by Bernstein. We have two explanations for this result: multicore processors and AES-NI instruction.

The first reason is multi-core processors, in other words, that the more immune processors have a more complicated cache architecture. As we know, in order to increase the performance, most modern microprocessor adopt multi-core architecture. For instance, the Atom N2800 that we used in this project has two cores. The mere inclusion of multiple cores complicates cache attacks immensely [3] since the attacker should be aware of the processor-specific multi-core cache behavior, like the eviction policy. For example, the Intel Core i5 we used has separate L1 and L2 cache, and shared L3 cache, so when the attacker implements the attack, he needs to know whether the data can remain in the L2 cache when it is evicted from L3 cache by other cores. Also, there could be a rescheduling of the attack thread from one core to the other core, so that may make the timing information more inaccurate.

Moreover, some cache parameters also changed very lot in past years. For example, the L1 cache of Pentium III used by Bernstein was only 16 KB, and the whole look up table was 8KB, which means the data in look up table were easily evicted and finally lead to the timing fluctuation. However, for example, the L1 cache in the Atom N2800 is 56 KB, which is much larger than the look up table's size, which means the look up table can entirely fit into the cache without being evicted, rendering the techniques powerless.

The other explanation is the new AES instruction set named AES-NI, a set of new microprocessor instructions that implements some of the sub-steps of the AES algorithms directly in hardware, so it reduces data manipulation (lookup in cache or memory) and lowers the risk of cache timing attack [5].

In order to verify the effectiveness of AES-NI to against cache timing attack, we tried to disable the AES-NI in Intel Core i5 by clearing some of the bits of the runtime environment variable OPENSSL_ia32cap, with a command that disables the use of AES-NI, PCLMULQDQ, and SSSE3 optimizations for x86-64:

```
> OPENSSL_ia32cap=~0x200000200000000
```

After we ran the test again, one line of the correlation result showed less than 256 possibilities, specifically 16 (as shown Table 3). Although it is only one line, it still proves that after we disabled the AES-NI, the processor became vulnerable to the cache-timing attack.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Intel Core i5 with disabled AES-NI | 16 | 13 | 93 | 91 | 90 | 92 | 9e | 94 | 9a | 9c | 96 | 95 | 97 | 98 | 9f | 9b | 99 | 9d |

**Table 3.**correlation results after disable the AES-NI.

## VIII. Conclusion

In this project, we tried to implement a cache-timing attack against AES, as proposed by Bernstein, on an Intel Atom N2800, to test the levels of immunity of Atom N2800 against cache timing attack. In order to do so, we tried the cache timing attack on three different processors: Intel Core i5, Intel Xeon E5410 and Intel Atom N2800. The results show that the Intel Core i5 and Intel Xeon E5410 are totally immune from the cache timing attack, while the level of immunity of the Atom N2800 was lower. Since this attack was proposed eight years ago, the change of processor's architecture, AES instruction and cache make this kind attack hard to implement on current processors. Future work could be done by demonstrating more tests on the Atom N2800 to get a more accurate estimate of its immunity level and try to change the ways to implement the attack and see whether there is any possible way to make cache timing attack successful again.

## References

[1] Daniel J. Bernstein. Cache-timing attacks on AES. April 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[2] R. Salembier. Analysis of cache timing attacks against AES. Scholarly Paper, ECE Department, George Mason University, Virginia; available from:http://ece.gmu.edu/courses/ECE746/project/F06_Project_resources /Salembier Cache Timing Attack.pdf, May 2006.

[3] K. Mowery, S. Keelveedhi, and H. Shacham. “Are AES x86 Cache Timing Attacks Still Feasible?” (Short Paper). In S. Capkun and S. Kamara, eds. In *Proceedings of CCSW 2012*. ACM Press, Oct. 2012.

[4] http://www.mentby.com/Group/openssl-users/can-aes-ni-be-disabled.html.

[5] http://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard--aes-/data-protection-aes-general-technology.html