Attacking AES Using Bernstein's Attack on Modern Processors

Hassan Aly* and Mohammed ElGayyar

Department of Mathematics, Faculty of Science,
Cairo University, Giza 12613, Egypt
{hassan.aly,melgayar}@sci.cu.edu.eg

Abstract. The Advanced Encryption Standard (AES) was selected by NIST due to its heavy resistance against classical cryptanalysis like differential and linear cryptanalysis. Even after the appearance of the modern side-channel attacks like timing and power consumption side-channel attacks, NIST claimed that AES is not vulnerable to timing attacks. In 2005, Bernstein [6] has successfully attacked the OpenSSL AES implementation on a Pentium III processor and completely retrieved the full AES key using his cache timing side-channel attack. This paper reproduces Bernstein's attack on Pentium Dual-Core and Core 2 Duo processors. We have successfully attacked the AES implemented in the latest OpenSSL release 1.0.1c using the most recent GCC compiler 4.7.0 running on both Windows and Linux in some seconds by sending 2^{22} plaintexts at most. We improved Bernstein's first round attack by using 2 way measurements. Instead of using only the above average timing information, we added the above minimum timing information which significantly improved the results.

Keywords: AES, timing attack, Bernstein's attack, cache memory attack, side-channel attack, cryptanalysis.

1 Introduction

For a long time, attacking cryptographic systems was relying only on its mathematical basis like the case in differential and linear cryptanalysis. To conduct such attacks you have to know either a number of ciphertexts or pairs of ciphertexts and plaintexts. Nowadays, several attacks are based on the information revealed from the encryption devices. Since this information is not the ciphertext or the plaintext, so it is often called side-channel information [5]. This information may be revealed by measuring the power consumption, heat consumption, cache access or time elapsed during processing.

The timing attacks can be considered as the most popular attacks that have greatly developed during the last ten years. Measuring the time taken to access cache memory helps identifying cache hits and misses which in turn is considered

* Current address: Department of Computer Science and Information, College of Science, Majmaah University, AzZulfi 11932, P.O. Box 1712, Kingdom of Saudi Arabia.

A. Youssef, A. Nitaj, A.E. Hassanien (Eds.): AFRICACRYPT 2013, LNCS 7918, pp. 127–139, 2013.

a dangerous information to be revealed. NIST has stated in [20] (section 3.6.2) that "Table lookup: not vulnerable to timing attacks; relatively easy to effect a defense against power attacks by software balancing of the lookup address.". However, the most powerful timing attacks on AES depend on measuring table lookup access times, which reveals most of the AES key bits.

2 AES Implementation Background

AES is the most widely used secret key block cipher. It is a substitution-permutation network (SPN), it has an iterative structure that works in rounds. That is, both encryption and decryption consists of number of iterations, called rounds, each round consists of a fixed set of operations, namely (SubBytes, ShiftRows, MixColumns, and AddRoundKey). The operations are applied to the input plaintext block, called state. After iterating a predefined number of rounds, (10, 12, and 14 rounds for 128-bit, 192-bit, and 256-bit keys, respectively), the final state is the ciphertext block.

Although the optimized ANSI C code submitted with Rijndael proposal uses 5 T-tables [31], high performance implementations like OpenSSL [25], uses only four 256-entry 32-bit T-tables T_0, T_1, T_2 and T_3 during encryption. This implementation precomputes original S-boxes and stores every lookup vector in a different T-table each of 1024-byte size. Each round state word $(x_{4i}^{(r)}, x_{4i+1}^{(r)}, x_{4i+2}^{(r)}, x_{4i+3}^{(r)}), i = 0, 1, 2, 3$, is generated as:

$$(x_0^{(r)}, x_1^{(r)}, x_2^{(r)}, x_3^{(r)}) = T_0[x_0^{(r-1)}] \oplus T_1[x_5^{(r-1)}] \oplus T_2[x_{10}^{(r-1)}] \oplus T_3[x_{15}^{(r-1)}] \oplus (k_0^{(r-1)}, k_1^{(r-1)}, k_2^{(r-1)}, k_3^{(r-1)})$$

$$(x_4^{(r)}, x_5^{(r)}, x_6^{(r)}, x_7^{(r)}) = T_0[x_4^{(r-1)}] \oplus T_1[x_9^{(r-1)}] \oplus T_2[x_{14}^{(r-1)}] \oplus T_3[x_3^{(r-1)}] \oplus (k_4^{(r-1)}, k_5^{(r-1)}, k_6^{(r-1)}, k_7^{(r-1)})$$

$$(x_8^{(r)}, x_9^{(r)}, x_{10}^{(r)}, x_{11}^{(r)}) = T_0[x_8^{(r-1)}] \oplus T_1[x_{13}^{(r-1)}] \oplus T_2[x_2^{(r-1)}] \oplus T_3[x_7^{(r-1)}] \oplus (k_8^{(r-1)}, k_9^{(r-1)}, k_{10}^{(r-1)}, k_{11}^{(r-1)})$$

$$(x_{12}^{(r)}, x_{13}^{(r)}, x_{14}^{(r)}, x_{15}^{(r)}) = T_0[x_{12}^{(r-1)}] \oplus T_1[x_1^{(r-1)}] \oplus T_2[x_6^{(r-1)}] \oplus T_3[x_{11}^{(r-1)}] \oplus (k_{12}^{(r-1)}, k_{13}^{(r-1)}, k_{14}^{(r-1)}, k_{15}^{(r-1)})$$

Here, T_0, T_1, T_2, T_3 are four lookup tables with 1 byte input and 4 bytes output and $(k_{4i}^{(r)}, k_{4i+1}^{(r)}, k_{4i+2}^{(r)}, k_{4i+3}^{(r)}), i = 0, 1, 2, 3$ is the i-th word of the r-th round key.

3 Cache Based Side Channel Attacks Background

Cache based side channel attacks, cache attacks, can be classified into three major types of attacks: time-driven, access-driven, and trace-driven.

In a trace-driven attack, the attacker should be able to monitor and collect the cache activity including every memory access during an encryption. This data collection process is meant to create a profile (trace) of cache hits and misses for a single encryption. The quality of the attack depends on how many traces are needed to begin the analysis phase, a better attack requires less traces.

In a time-driven attack, the attacker do not need to have the ability of collecting data of every memory access. Instead, the attack relies on a value that can be used to describe or approximate the total number of cache hits and misses. In most of time-driven attacks, the attacker collects only the total execution time of

an encryption as many times as needed, then use mathematical tools to analyze the collected data. The quality of the attack depends on how many encryptions are needed to begin the analysis phase, a better attack requires less encryptions.

In an access-driven attack, the attacker can detect which cache sets were modified by the encryption process, which leads the attacker to know which lookup table entries are accessed during the encryption. Then, uses elimination and non-elimination techniques to detect the right key candidates. For more information about microarchitectural attacks, consult [2].

4 Related Work

In this section, we introduce the history of cache based side channel attacks and timing attacks related to Bernstein's attack.

In 1996, Kocher [16] first introduced timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In 1998, Kelsey et al. [15] mentioned the "attacks based on cache hit ratio in large S-box ciphers" prospect. In 2002, Page [27] expanded the idea proposed by Kelsey et al., of cache memory being used as a side-channel which leaks information during the run of DES. As well as describing and simulating the theoretical attack. He discussed how hardware and algorithmic alterations can be used to defend against such techniques in [28] in 2003. Later on, Tsunoo et al. [34,35] implemented the first practical cache timing cryptanalysis of DES, 3DES, MISTY1, Camellia and AES.

In 2003, Brumley and Boneh [10] devised and implemented a remote timing attack against unprotected OpenSSL implementation of RSA over a local area network.

In 2005, Acıiçmez et al. [4] improved the efficiency of Brumley and Boneh timing attack on unprotected SSL implementations of RSA-CRT by a factor of more than 10. Earlier in the same year, Bertoni et al. [7] introduced the first trace-driven attack on AES based on induced cache misses. Also they proposed a simple countermeasure against the attack.

Bernstein [6] derived an attack on AES which depends only on calculating the encryption time information caused by cache memory hits and misses then comparing timing data using statistical methods. Bernstein's attack is a time-driven cache attack. It is performed as a template attack where at first a profile under a known key is generated with the same platform as the later attacked one. The real attack is performed in a second phase where a profile of an unknown key is generated. Those two profiles are then correlated and the key space for the unknown key is reduced. In a last phase for full key recovery, a brute-force of the remaining key space is performed. This attack is not affected by cache architecture or active manipulation, it only depends on the similarity between reference and target machines. Later, Percival [29] was the first to use access-driven attack against RSA, and demonstrated that the shared access to cache memory provides an easily used covert channel between threads, allowing in many cases for theft of cryptographic keys.

Osvik et al. [26,33] led the work on attacking AES using access-driven cache memory attacks, and described several software side-channel attacks based on

inter-process leakage through the state of the CPU's cache memory. The authors discussed an attack called *synchronous attack*, which requires knowledge of either the plaintext or the ciphertext. The synchronous attacker can operate synchronously with the encryption on the same processor. Moreover, they demonstrated an extremely strong type of attack called *asynchronous attack*, which does not require any knowledge of plaintexts or ciphertexts. The asynchronous attacker will execute his own program on the same processor as the encryption program without any explicit interaction, depending on the knowledge of the non-uniform distribution of the plaintexts or ciphertexts. They also experimentally demonstrated their applicability to real systems, such as OpenSSL and presented a variety of countermeasures which can be used to mitigate such attacks. However, they did not give a description of how to perform a full asynchronous attack. Lauradoux [17] proposed some countermeasures against these attacks, and Canteaut et al. [11] followed him in 2006.

In 2006 also, Neve et al. [23,22] presented a thorough analysis of Bernstein's attack, reproducing the attack and demonstrating results of important experiments practically. They answered a lot of open questions about the attack like, what if there is no learning phase? Can this attack be a real remote threat or not? and more. Then they extended the attack with a second round attack to reveal other key bits that could not be revealed by Bernstein's first round attack.

In the same year, Neve et al. [21] introduced an access-driven attack on AES, by demonstrating how a spy process running on the same single threaded CPU can measure the number of accessed cache lines by another process running on the same CPU.

Another cache memory attack was introduced in 2006 by Bonneau et al. [9], in this cache collision attack, they aimed to predict cache collisions timing variation using a simplified cache model. Their most powerful attack recovered a full 128-bit AES key with an improvement of almost four orders of magnitude over Bernstein's attack.

A cache based remote timing attack followed by Acıiçmez et al. [3], they described an expanded second round attack that can be used to obtain secret keys of remote cryptosystems. Their attack requires hyper threading enabled system with a large enough workload. In 2006, Acıiçmez et al. [1] presented a trace-driven attack on AES. They described a first two rounds attack and a last round attack as well. At the end of their work, they show the trade-off between the online and offline cost of the attack in details. Later in the same year, Bonneau [8] described a final round trace-driven attack on AES, building off of previous work by Acıiçmez and Koç [1]. Bonneau introduced an algorithm that reduces the problem of attacking AES given a small set of cache traces, to a simple constraint satisfaction problem.

In 2007, Tiri et al. [32] have proposed an analytical model for time-driven cache attacks. They presented a tool to help us evaluate the security of symmetric key ciphers against against such attacks.

In 2008, Zhao et al. [37] introduced a first two rounds access-driven attack on AES. Introducing the elimination technique in guessing the key bytes.

They succeeded in recovering the full 128-bit AES key through the first round attack using about 350 samples, and two rounds attack using about 80 samples in a few seconds.

In 2010, Rebeiro et al. [30] justified that cache timing attacks on AES are unable to force hits in the third round and concluded that a similar third round cache timing attack does not work. Hence, protecting only the first two AES rounds prohibits cache based timing attacks. Zhao and Wang [36] presented an improved trace-driven attack on AES and CLEFIA by considering S-box misalignment, and due to this feature, about 200 samples are enough to obtain full 128-bit AES key within seconds. Bogdanov et al. introduced a novel differential collision attack based on the MDS properties of AES on embedded CPUs. Their experiments show that efficient attacks on embedded systems implementing AES are not theoretical any more.

In 2011, Gallais et al. [12] introduced an improved adaptive plaintext, and presented a new known plaintext trace-driven cache-collision attacks against embedded AES implementations. Their experiments show that with approximately 30 known plaintexts, the key space of AES 128-bit is reduced to 2^{30}. Gullasch et al. [13] improved over prior work [26,21,33] by providing a first practical access-driven cache attack on AES in the asynchronous model. They introduced a novel approach by using neural networks to handle noise surrounding key candidates. Their experiments shows that performing only 100 encryptions is enough to find the key in average of 3 minutes including key search phase. They mentioned a way to transfer the offline phase to another machine by downloading 62.5 KB only per attack.

In 2012, Mowery et al. [19] proved that any cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail.

5 Bernstein's Attack on AES

Bernstein's attack is a first round cache timing attack, it consists of two online phases, namely profiling and attacking phases, and two offline phases, namely correlation and key search phase. During profiling and attacking phases, it measures and collects total execution time of a single encryption, thousands or millions of times. Then a mathematical correlation phase matches between results of the earlier online stages and generates a list of possible key candidates. The last phase is the key search phase, it is a brute force attack, searching for the unknown key in the reduced key space generated by the correlation phase.

The biggest advantage of this attack, it is a generic attack and it can be performed mostly without any knowledge about many processor details.

Bernstein succeeded to attack an OpenSSL implementation of AES, which makes use of four T-tables only, and utilizes a total of four kilobytes (4096 bytes) of memory. The idea is that for the first round, the table lookup indices $x_i^{(0)}$ are each related to only one key byte $k_i^{(0)}$ and one plaintext byte p_i:

$$x_i^{(0)} = p_i \oplus k_i^{(0)},\ i = 0, 1, 2, ..., 15.$$

So, at the profiling phase we know the i-th key byte $k_i^{(0)}$ and the i-th plaintext byte p_i, which leads directly to the table lookup index $x_i^{(0)}$. On the other hand, at the attacking phase we know both p_i and $x_i^{(0)}$, which reveals the unknown i-th key byte $\hat{k}_i^{(0)}$:

$$\hat{k}_i^{(0)} = p_i \oplus x_i^{(0)}, i = 0, 1, 2, ..., 15.$$

Osvik et al. [26,33] mentioned a lot of important shortcomings about Bernstein's attack:

- it requires reference measurements of encryption under known key in an identical configuration, and these are often not readily available (e.g., a user may be able to write data to an encrypted file system, but creating a reference file system with a known key is a privileged operation).
- it relies on timing the encryption and thus, it seems impractical on many real systems due to excessively low signal-to-noise ratio.
- even when this attack works, it requires high number of analyzed encryptions.

To work around these shortcomings, Neve et al. [23,22] suggested that instead of using another reference machine, attacking two different keys on the same machine might recover some bits by comparing similar byte signatures. Also, we tried to compare between two different attacking stages for the same key, but we failed to get any good results.

However, in some cases like a shared computer with different users accounts, the attacker has access to his own account on the machine in which he can collect required information about his own known key.

6 Our Work

Bernstein in [6] presented his attack against the OpenSSL 0.9.7a of AES implementation on an 850MHz Pentium III desktop computer running FreeBSD 4.8. O'Hanlon and Tonge [24] failed to collect any useful data about the key by attacking a Pentium IV running GCC 4.0.0 and OpenSSL 0.9.7f. They succeeded with a Pentium III running GCC 2.95.3 against the MIRACL [18] implementation of AES. Followed by Canteaut et al., they attacked a Pentium IV processor [11] trying to modify the cache state before the attack by removing system calls. Also recently, Jayasinghe et al. [14] succeeded to implement Bernstein's attack on the original configuration used by Bernstein at 2005. Table 1 outlines all these implementation of the attack.

While through our experiments we tested over 100 random keys, we decided to attack a fixed key $k = \{2b, a8, 62, a3, 4d, 42, e2, 44, 27, 89, a4, 4a, c6, 7e, cd, eb\}$, through the rest of this paper.

After testing the attack on Ubuntu 9.10, OpenSSL version 0.9.8g and GCC 4.4.1 on a Pentium Dual-Core processor as shown in Table (2), we noticed that smaller packet sizes are giving better results. Our best results were achieved by sending 2^{27} plaintexts of size 100 bytes.

[1] This attack was performed on a 32-bit Windows7 Ultimate sp1.
[2] This attack was performed on a 64-bit Windows7 Home Premium sp1.

Table 1. This table shows attackers hardware and software configurations

Attacker	CPU model	GCC	AES software	#Packets
Bernstein[6]	Pentium III	2.95.4	OpenSSL 0.9.7a	2^{27}
O'Hanlon et al.[24]	Pentium III	2.95.3	MIRACL	$2^{30}/3$
Canteaut et al.[11]	Pentium III	3.2.2	Original [31]	2^{30}
	Pentium IV	3.2.2	Original [31]	2^{26}
Jayasinghe et al.[14]	Pentium III	2.95.4	OpenSSL 0.9.7a	2^{27}
Our attack	Pentium Dual-Core	4.4.1	OpenSSL 0.9.8g	2^{26}
	Pentium Dual-Core[1]	4.7.0	OpenSSL 1.0.1c	2^{20}
	Pentium Core 2 Duo[2]	4.7.0	OpenSSL 1.0.1c	2^{20}

Table 2. This table shows the attacked processor model and cache size

CPU model	Level	Cache line size	Cache sets	Associativity	Total size
Pentium Dual-Core T2060	L1	64 B	2×64	8	2×32 KB
	L2	64 B	4096	4	1 MB
Pentium Core 2 Duo P7550	L1	64 B	2×64	8	2×32 KB
	L2	64 B	4096	12	3 MB

After testing the attack on a similar configuration to Bernstein's, we started porting the attack to test on Windows7 32-bit and 64-bit operating systems. We chose to port the attack to MinGW, not Cygwin, so the attack is not restricted to machines running Cygwin only. Porting the attack to MinGW was a hard job, due to the differences between sockets implemented in Linux and Windows.

Performing the first attack on Windows 32-bit, OpenSSL version 0.9.8g and GCC 4.7.0 on a Pentium Dual-Core processor as shown in Table (2) was successful, and some key candidates appeared after sending 2^{26} samples of size 100. After this we followed Neve et al. [22] by removing the network delay from the attack, since the execution time is measured on the server. We merged the original study and server programs to create the new ServerNoNetwork program which runs locally. Also, we fixed the samples length to 16 as Neve et al. advised, since only the first 16 bytes are encrypted even if the packet size is 1000.

Performing the attack again with the new attack program on our testing platforms lead to better results in less time. Our best results began to appear at 2^{25} samples. This attack took less than 30 minutes and the results included 4 accurate peaks for k_1, k_5, k_9, k_{13}. By repeating the attack several times with the same configuration, the same 4 peaks kept appearing and no other candidates were observed.

Since the main concept used by Bernstein depends on touching the cache memory before and after AES encryption, we restored back the ability to send different sizes of samples again. At the first glance, we tried to speedup the attack by removing unused code like calling rand() function, which fills the whole packet array while we need only the first 16 entries to be randomized, also we added a command line argument for the limit to stop whenever it reaches that limit.

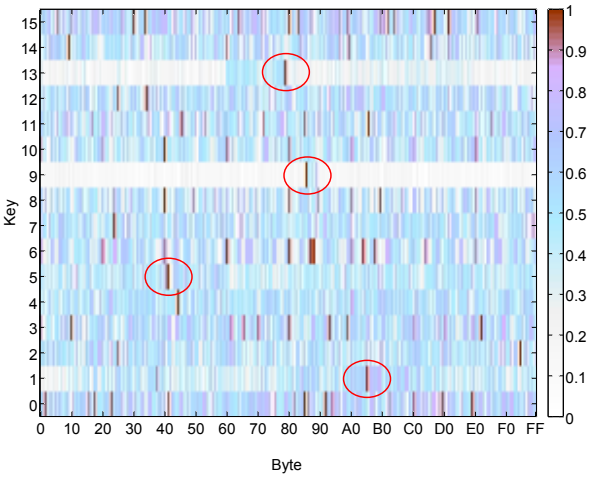

Fig. 1. Single peaks for group $g_1 = \{a8, 42, 89, 7e\}$ are marked with red. x-axis represents byte values in hexadecimal and y-axis represents the 16 key bytes

By repeating the attack ten times, we discovered that results are mostly appearing as single peaks on this configuration see Fig. 1. Another interesting behavior is the relation between results of the same attack, it appears that results are grouped in 4 groups, g_0, g_1, g_2, and g_3, each group contains 4 key candidates: $g_i = \{k_i, k_{i+4}, k_{i+8}, k_{i+12}\}, i = 0, 1, 2, 3..$ The first group appeared while sending samples of size 16 was g_1, other groups appeared later with larger samples.

After improving and optimizing the speed of the merged program, a successful attack on Windows7, GCC 4.71, and OpenSSL 1.0.1c, required sending only 1M of samples with different sizes, and we recovered the full key without a brute force attack in less than 20 seconds. We didn't need to use the brute force attack, since all key candidates were found as single peaks.

The reason behind this grouping is how OpenSSL implemented AES and how the MinGW GCC compiler assembles the implementation on Windows. Every group corresponds to a set of certain table lookup indices, i.e g_0 corresponds to T_0 lookup indices and so on, recalling that:

$$\left(x_{4i}^{(r)}, x_{4i+1}^{(r)}, x_{4i+2}^{(r)}, x_{4i+3}^{(r)}\right) = T_0\left[x_{4i}^{(r-1)}\right] \oplus T_1\left[x_{4i+5}^{(r-1)}\right] \oplus T_2\left[x_{4i+10}^{(r-1)}\right] \oplus T_3\left[x_{4i+15}^{(r-1)}\right] \oplus \left(k_{4i}^{(r-1)}, k_{4i+1}^{(r-1)}, k_{4i+2}^{(r-1)}, k_{4i+3}^{(r-1)}\right)$$

where $i = 0, 1, 2, 3$ and all sum operations are done modulo 16.

While analyzing the results in Table 3, we found that peaks are changing approximately every extra 300-350 bytes to the sample. It appears that the attacked operating system and/or MinGW GCC compiler are partitioning the T-tables into 256 byte chunks (4 cache lines per table), and due to misalignment of the tables in cache, an extra cache line is used which means $256 + 64 = 320$ bytes in cache.

Following this theory, we found that 320, 640, 960, and 1280 bytes are very special sample sizes that reveals more than 8 key candidates at once in some

Table 3. This table shows the relation between revealed groups and sample size by sending only 2^{20} samples

Sample Size in Bytes	Recovered Key Indices	Group	Time in Seconds	Improved Time
16	$1, 5, 9, 13$	g_1	1.6	0.9
100-300	$5, 9, 13$	g_1	2-10	1.1-1.5
350	$0, 4, 8, 12$	g_0	12	1.6
400-600	$0, 4, 8$	g_0	13-19	1.7-2.0
650	$3, 7, 11, 15$	g_3	21	2.1
700-950	$7, 11, 15$	g_3	22.5-30	2.2-2.7
1000	$2, 6, 10, 14$	g_2	32.5	2.8

cases. Figure 2 shows the evolution of the correct key candidate against the number of samples, the figure shows the results of measuring 1M samples of size 320 Byte, as you can notice, the correct key candidates were very clear at the level of 256K and in some cases at 64K samples only.

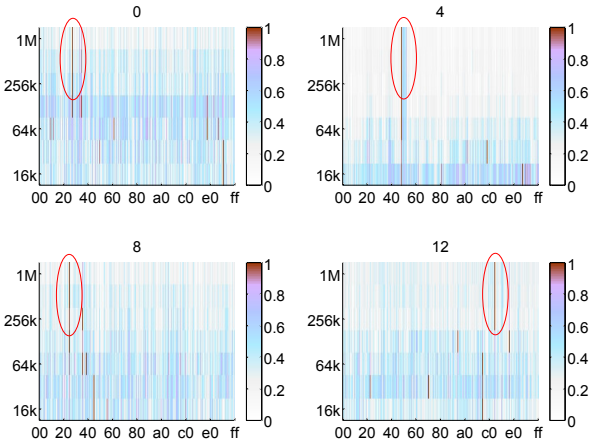

Fig. 2. Single peaks for group $g_0 = \{2b, 4d, 27, c6\}$ are marked with red. In this grouped correlation graph, the x-axis represents the byte value and the y-axis represents number of samples.

The last part of our work is the most exciting part. We tried to modify the core of the attack by replacing the main measurement criteria by another one. The core measurement criteria in the attack is $u[j][b] - taverage$ where

$$taverage := \frac{\sum_{i=0}^{total\#packets} timing_i}{total\#packets}$$

is the total overall average time and

$$u[j][b] := \frac{\sum_{i=0}^{tnum[j][b]} timing[j][b]_i}{tnum[j][b]},$$

$tnum[j][b]$ is the total number of samples for the j-th key candidate and the plaintext byte b. We aimed to replace this complicated formula with a simpler one to help gain more speed and accuracy in our attack program, our formula is $umin[j][b] - tmin$ where

$$tmin := min_i(timing_i)$$

is the overall minimum timing and

$$umin[j][b] := min_i(timing[j][b]_i)$$

is the local minimum for the j-th key candidate and the plaintext byte b.

We were not surprised when this new measure succeeded to recover the same groups as the original measure with the same configurations, since the minimum is used in access-driven attacks to eliminate wrong key candidates. Calculating minimum timing information for each candidate eliminates all low value candidates easily because candidates with cache hits have a lower timing information than candidates with cache misses. Calculating the minimum also eliminates the noise; if a particular candidate is affected by noise and hence had a high timing value, using minimum means that at the first instant of noise absence, the real timing information is recorded and will never be raised again even if the noise is back.

Figure 3 shows the results of the first group with the new measurement, which looks more clear than the original measure.

Our experiments show that the reasons behind our success in recovering the full 128-bit key so fast is a combination of the simple structure of GCC 4.7.0 for MinGW, optimizing the attack program, removing all redundant code, and using two measurement criterions instead of only one.

After succeeding with the new measure, we kept both measurements, so the attacker can choose which measure to use. At the correlation stage, both measurements data are kept together in the file for double checking.

At this point, using the merged program ServerNoNetwork, with the original and new measurements, we succeeded to recover the full AES key, without a brute force search in less than 20 seconds, using less than 1M random plaintexts.

7 Future Work

After attacking OpenSSL successfully we plan to attack other cryptographic libraries. Our first trial was to attack MIRACL version 5.5.4 [18] which implements AES in a small slow implementation and another high performance one, using 5 lookup T-tables: 4 for all rounds and one for the final round. Attacking this implementation failed to extract more than 2 or 3 key bytes for the small implementation, while we succeeded to get a better chance with the fast one.

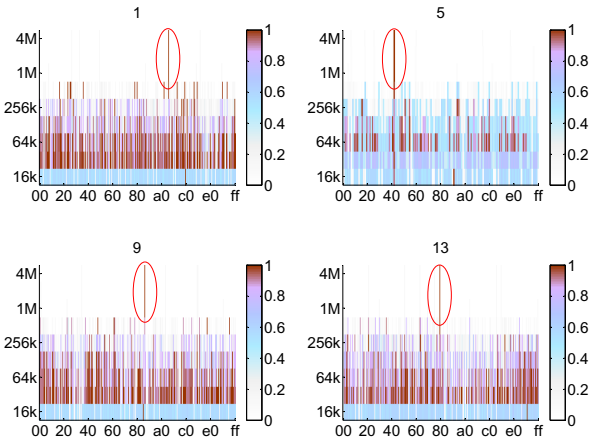

Fig. 3. Single peaks for group $g_1 = \{a8, 42, 89, 7e\}$ are marked with red. In this grouped correlation graph, the x-axis represents the byte value and the y-axis represents number of samples.

8 Conclusion

We succeeded to attack the latest OpenSSL implementation of AES using Bernstein's cache timing attack on a different testing environment from those used earlier. We replaced the original "above average" measure with a simple "above minimum" one. Our experiments shows that GCC 4.7.0 for MinGW might be the reason behind recovering the 128-bit key in seconds using either the original or new measurements.

References

1. Acıiçmez, O., Koç, Ç.: Trace-driven cache attacks on AES (short paper). Information and Communications Security, 112–121 (2006)
2. Acıiçmez, O., Koç, K.: Microarchitectural attacks and countermeasures. Cryptographic Engineering, 475–504 (2009)
3. Acıiçmez, O., Schindler, W., Koç, Ç.K.: Cache based remote timing attack on the AES. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 271–286. Springer, Heidelberg (2006)
4. Acıiçmez, O., Schindler, W., Koç, Ç.: Improving Brumley and Boneh timing attack on unprotected SSL implementations. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, pp. 139–146. ACM (2005)
5. Bar-El, H.: Introduction to side channel attacks, vol. 43. Discretix Technologies Ltd. (2003)
6. Bernstein, D.: Cache-timing attacks on AES (2005), `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`

7. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: AES power attack based on induced cache miss and countermeasure. In: International Conference on Information Technology: Coding and Computing, ITCC 2005, vol. 1, pp. 586–591. IEEE (2005)
8. Bonneau, J.: Robust final-round cache-trace attacks against AES. Tech. rep., Citeseer (2006)
9. Bonneau, J., Mironov, I.: Cache-collision timing attacks against AES. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 201–215. Springer, Heidelberg (2006)
10. Brumley, D., Boneh, D.: Remote timing attacks are practical. In: Proceedings of the 12th Conference on USENIX Security Symposium, vol. 12, p. 1. USENIX Association (2003)
11. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding cache attacks (2006)
12. Gallais, J., Kizhvatov, I., Tunstall, M.: Improved trace-driven cache-collision attacks against embedded AES implementations. Information Security Applications, 243–257 (2011)
13. Gullasch, D., Bangerter, E., Krenn, S.: Cache games–bringing access-based cache attacks on AES to practice. In: 2011 IEEE Symposium on Security and Privacy (SP), pp. 490–505. IEEE (2011)
14. Jayasinghe, D., Fernando, J., Herath, R., Ragel, R.: Remote cache timing attack on Advanced Encryption Standard and countermeasures. In: 2010 5th International Conference on Information and Automation for Sustainability (ICIAFs), pp. 177–182. IEEE (2010)
15. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side channel cryptanalysis of product ciphers. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 97–110. Springer, Heidelberg (1998)
16. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
17. Lauradoux, C.: Collision attacks on processors with cache and countermeasures. In: Western European Workshop on Research in Cryptology WEWoRC, vol. 5, pp. 76–85 (2005)
18. MIRACL: Multiprecision Integer and Rational Arithmetic C/C++ Library. Shamus Software Ltd., Dublin, `http://www.shamus.ie`
19. Mowery, K., Keelveedhi, S., Shacham, H.: Are AES x86 cache timing attacks still feasible? In: Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop, pp. 19–24. ACM (2012)
20. Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., Roback, E.: Report on the development of the Advanced Encryption Standard (AES). Journal of Research of the National Institute of Standards and Technology 106(3) (2001), `http://archive.org/details/jresv106n3p511`
21. Neve, M., Seifert, J.-P.: Advances on access-driven cache attacks on AES. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 147–162. Springer, Heidelberg (2007)
22. Neve, M., Seifert, J., Wang, Z.: Cache time-behavior analysis on AES. In: Selected Area of Cryptology (2006)
23. Neve, M., Seifert, J., Wang, Z.: A refined look at Bernstein's AES side-channel analysis. In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications security. pp. 369–369. ACM (2006)
24. O'Hanlon, M., Tonge, A.: Investigation of cache timing attacks on AES. School of Computing, Dublin City University (2005)

25. OpenSSL: The open source toolkit for SSL/TLS, http://www.openssl.org
26. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
27. Page, D.: Theoretical use of cache memory as a cryptanalytic side-channel. Tech. rep., Citeseer (2002)
28. Page, D.: Defending against cache-based side-channel attacks. Information Security Technical Report 8(1), 30–44 (2003)
29. Percival, C.: Cache missing for fun and profit. In: BSDCan 2005 (2005)
30. Rebeiro, C., Mondal, M., Mukhopadhyay, D.: Pinpointing cache timing attacks on AES. In: 23rd International Conference on VLSI Design, VLSID 2010, pp. 306–311. IEEE (2010)
31. Rijmen, V., Bosselaers, A., Barreto, P.: Optimised ANSI C code for the Rijndael cipher (now AES). Public domain software (2000), http://fastcrypto.org/front/misc/rijndael-alg-fst.c
32. Tiri, K., Acıiçmez, O., Neve, M., Andersen, F.: An analytical model for time-driven cache attacks. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 399–413. Springer, Heidelberg (2007)
33. Tromer, E., Osvik, D., Shamir, A.: Efficient cache attacks on AES, and countermeasures. Journal of Cryptology 23(1), 37–71 (2009)
34. Tsunoo, Y.: Cryptanalysis of block ciphers implemented on computers with cache. In: Preproceedings of ISITA 2002 (2002)
35. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of DES implemented on computers with cache. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 62–76. Springer, Heidelberg (2003)
36. Zhao, X., Wang, T.: Improved cache trace attack on AES and CLEFIA by considering cache miss and S-box misalignment. Tech. rep., Cryptology ePrint Archive, Report 2010/056 (2010)
37. Zhao, X., Wang, T., Dong, M., Yuanyuan, Z., Zhaoyang, L.: Robust first two rounds access driven cache timing attack on AES. In: 2008 International Conference on Computer Science and Software Engineering, vol. 3, pp. 785–788. IEEE (2008)