On the complexity of side-channel attacks on AES-256
- methodology and quantitative results on cache attacks -

Michael Neve and Kris Tiri

Digital Enterprise Group
Intel Corporation, JF5-254
{michael.neve,kris.tiri}@intel.com

Abstract. Larger key lengths translate into an exponential increase in the complexity of an exhaustive search. Side-channel attacks, however, use a divide-and-conquer approach and hence it is generally assumed that increasing the key length cannot be used as mitigation. Yet, the internal round structure of AES-256 and its key-scheduling seem to hinder a direct extension of the existing attacks on AES-128 and thus challenge the proposition above. Indeed two consecutives round keys are required to infer the secret key and the MixColumns operation, not present in the last round, apparently increases the key search complexity from to 2^8 to 2^{32}. Additionally, it is unclear what the impact of the different round structures is on the number of required measurements. In this paper, we explore this question and show how to attack AES-256 with a key search complexity of $O(2^8)$. This work confirms with practical experiments that AES-256 only offers a marginal increase in resistance against the attacks –both in the required number of measurements and in the required processing time. As an example, we quantify this increase for the case of cache-based side-channel attacks: AES-256 only provides an increase in complexity of 6 to 7 compared to cache-based attacks on AES-128.

Keywords: side-channel, cache attacks, AES-192, AES-256, symmetric key cipher.

1. Introduction

The cryptanalytic strength of a cipher, *i.e.* its resistance against mathematical and algebraic attacks, is highly dependent upon the length of the key material. Choosing the key length is a subtle exercise in balancing security and performance requirements. Some recommendations are available for a range of operations and utilizations of various cryptographic ciphers [1]. Based on the expected computing power and attacks, the reports convey the year until which it is *safe* to use a certain key length. A special publication of the National Institute for Standards and Technologies (NIST) attributes all versions (128-, 192- and 256-bit) of the Advanced Encryption Standard (AES [2]) with a security lifetime beyond 2030 [3]. An elder IETF report of the National Security Agency (NSA) gave symmetric ciphers with a key size of at least 256 bits until the year 2245 [4]. Recommendations from various

European institutions concur with a security lifetime of AES-256 until at least 2020 [5,6].

However, those recommendations for key length only take into account mathematical and algebraic attacks; they do not consider algorithmic nor implementation-related attacks that weaken or even bypass the mathematical complexity of a cipher. However designers are confronted with both sides in the development of security products. Side-channel analysis (SCA), which takes advantage of information leaked by the physical implementation of a cipher, represents such a class of attacks. For over a decade, this technique has been used to target cryptographic applications in a variety of devices such as smart cards (initiated by the work of Kocher [10]), and setup boxes. Yet the interaction between software and a micro-architecture optimized for performance leads to similar vulnerabilities and in recent times, SCAs have been demonstrated that successfully attack software implementations of symmetric and public key encryption algorithms running on high-end microprocessors. In this manuscript, we use the cache attacks, which exploit side-channel information that is leaked by a microprocessor's cache, to illustrate how AES-256 can be attacked, but it is straightforward to extend to other applications.

As side-channels use a divide-and-conquer approach it is expected that increasing the key length does not provide an exponential relationship between attack complexity and key length as is the case for a brute force attack. To find the secret key of AES-256, however, two consecutive round keys are required and the MixColumns operation, not present in the last round, apparently increases the key search complexity to 2^{32}. Additionally, it is unclear what the impact of the different round structures is on the number of required measurements. In this work, we present the methodology to attack AES-256, which overcomes the apparent 2^{32} complexity, and quantify the exact increase in resistance. To be more precise, cracking AES-256 using time-driven cache attacks requires only 7 times the measurements and the processing time of an attack on AES-128. Access-driven attacks on AES-256 require 6 times the measurements and processing time.

The next section introduces the key concepts of AES. Section 3 details how the cache behavior is leaking information about the AES cipher key. We describe in section 4 our attack strategy for AES-192 and AES-256 and present its results in section 5 for both time-driven and access-driven attacks. We share our conclusions in section 6.

2. Overview of AES

Interested readers are referred to [9] for a complete description of the Advanced Encryption Standard (AES). In this section, we only introduce the concepts required for the understanding of the paper.

AES is an iterated symmetric key cipher with fixed block length of 128 bits and with key length of 128, 192 or 256 bits. Each round transformation i combines a 16-byte input X^i and a 16-byte round key K^i, and produces a 16-byte output X^{i+1}. The number of rounds N_r depends on the key length, *i.e.* 10, 12 or 14 rounds for 128-,

192- or 256-bit key. We denote with L the cipher key length, in bytes (following notations of [9], $L=4\cdot N_k$, where N_k is the number of 32-bit words in the cipher key).

A round is comprised of four operations: SubBytes, ShiftRows, MixColumns and AddRoundKey. Popular software implementations of AES (like OpenSSL) optimize these operations by using four precomputed tables T_0, T_1, T_2, T_3, mapping a 1-byte input to a 4-byte output. With x^i_j being the j^{th} byte of X^i and k^i_j being the j^{th} byte of K^i, *i.e.*

$$X^i = \{x^i_0|x^i_1|\ldots|x^i_{15}\} \text{ and } K^i = \{k^i_0|k^i_1|\ldots|k^i_{15}\},$$

one round of AES is computed by

$$\begin{aligned} X^{i+1} = \{ & T_0[x^i_0] \oplus T_1[x^i_5] \oplus T_2[x^i_{10}] \oplus T_3[x^i_{15}] \oplus \{k^i_0|k^i_1|k^i_2|k^i_3\} \mid \\ & T_0[x^i_4] \oplus T_1[x^i_9] \oplus T_2[x^i_{14}] \oplus T_3[x^i_3] \oplus \{k^i_4|k^i_5|k^i_6|k^i_7\} \mid \\ & T_0[x^i_8] \oplus T_1[x^i_{13}] \oplus T_2[x^i_2] \oplus T_3[x^i_7] \oplus \{k^i_8|k^i_9|k^i_{10}|k^i_{11}\} \mid \\ & T_0[x^i_{12}] \oplus T_1[x^i_1] \oplus T_2[x^i_6] \oplus T_3[x^i_{11}] \oplus \{k^i_{12}|k^i_{13}|k^i_{14}|k^i_{15}\} \}. \end{aligned}$$

The last round of AES, however, is slightly different as it does not include the MixColumns operation. Therefore another table T_4 is often defined and used for the last round (*i.e.* here $i=N_r$):

$$\begin{aligned} X^{i+1} = \{ & T_4[x^i_0] \oplus T_4[x^i_5] \oplus T_4[x^i_{10}] \oplus T_4[x^i_{15}] \oplus \{k^i_0|k^i_1|k^i_2|k^i_3\} \mid \\ & T_4[x^i_4] \oplus T_4[x^i_9] \oplus T_4[x^i_{14}] \oplus T_4[x^i_3] \oplus \{k^i_4|k^i_5|k^i_6|k^i_7\} \mid \\ & T_4[x^i_8] \oplus T_4[x^i_{13}] \oplus T_4[x^i_2] \oplus T_4[x^i_7] \oplus \{k^i_8|k^i_9|k^i_{10}|k^i_{11}\} \mid \\ & T_4[x^i_{12}] \oplus T_4[x^i_1] \oplus T_4[x^i_6] \oplus T_4[x^i_{11}] \oplus \{k^i_{12}|k^i_{13}|k^i_{14}|k^i_{15}\} \}. \end{aligned}$$

Figure 1 gives a block representation of the two final rounds of any AES encryption irrespectively of the key size. The next to last round (N_r-1) is identical to all previous ones, while the last one (N_r) does not include a MixColumns operation.

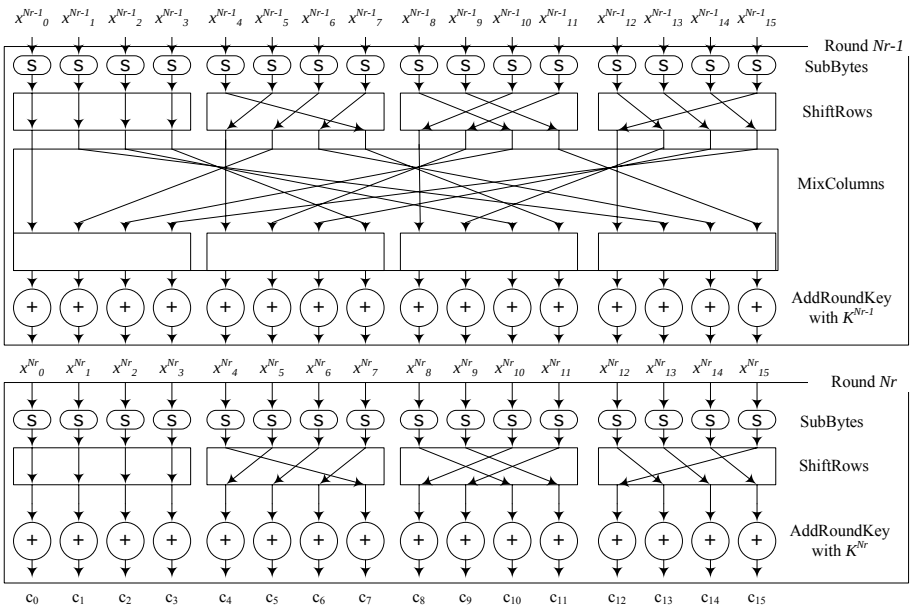

Fig. 1. Block view of the two final state transformations for any AES encryption.

Note that any round i can be inverted with the knowledge of the output and the round key, *i.e.* X^i can be computed from X^{i+1} and K^i.

Two functions compose the AES key scheduling. First, the cipher key is expanded into a key array of $16 \cdot (N_r+1)$ bytes. Then, the successive (16-byte) round keys are derived from this array. There is one more round key than there are rounds because there is an initial key addition (AddRoundKey) to whiten the plaintext.

In this paper, we will be considering key lengths of 128, 192 and 256 bits. The key expansion iterates over the last L bytes to produce L new bytes of the key array. Figure 2 illustrates the last iterations of the key schedule for key size higher than 128-bit. It is important to understand what can be deduced by the last round key, *i.e.* whether or not it is possible to compute the cipher key from the last round key – we refer to this as the *roll back* (of the last round key to the cipher key). The roll back is only possible for L=16; in other cases (Fig. 2.a and 2.b), knowledge of parts ($k^{11}_{8..11}$ and $k^{11}_{12..15}$ for AES-192) or all (for AES-256) of the next to last round key is also required to compute the cipher key.

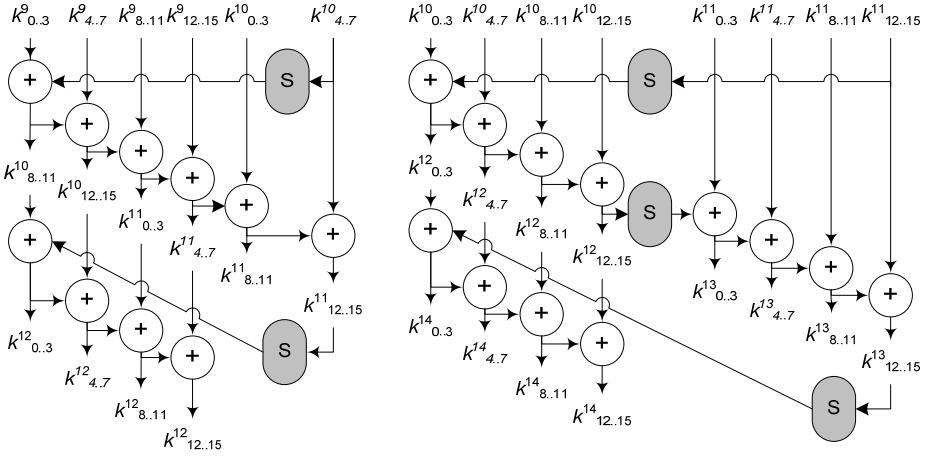

Fig. 2. Details of the last rounds of the AES key expansion for (a) 192-bit and (b) 256-bit cipher key. In opposition to the case of 128-bit long key, the cipher key can be rolled back from the last round key; while it is not the case in (a) and (b).

3. Cache-based attacks

The CPU data cache –simply called cache in the remainder of this manuscript– is a fast memory that ideally stores the next data that will be requested by the CPU. This is done to overcome the latency penalty associated with fetching data from a higher level of the memory architecture. In practice, the cache holds a copy of the recently used data and its contiguous data in a cache line, because they will likely be requested again soon.

If the next CPU request is present in the cache (this event is called a cache-hit), then the access time is short; otherwise (cache-miss), the data must be fetched, with an additional latency, from a higher memory location, *i.e.* a larger cache (slower) or from main memory (even slower). The data involved in the cache-miss would then be stored (with its contiguous data, *i.e.* to fill a complete cache line) in the lower cache levels to better accommodate probable future requests of this data (and/or its contiguous data).

However, since the cache has a limited size, the new cache line would evict a line of previously stored data. So different data that are mapped to the same cache line will collide and compete between each other. These evictions have been used to mount side-channel attacks based on the analysis of the cache behavior: on one hand *access-driven attacks* infer information from the evolution in time of the state of the cache, while *time-driven attacks* analyze the differences in the overall execution time.

There has been a significant amount of research on cache side-channel vulnerabilities since early 2002. Page delivered the first theoretical work predicting

how a microprocessor's cache would leak information [15]. Tsunoo was the first to report a practical attack [18] in which some key material was recovered from DES. Later, Bernstein presented a *time-driven* attack on AES-128 leading to full key recovery with 2^{27} 400-byte plaintexts and encryption time pairs [20]. In [22], Percival disclosed an *access-driven* attack on RSA targeting the cache and leveraging the hardware parallelism of execution pipelines in simultaneous multithreading processors. At the same time, [23] developed similar techniques targeting AES-128. Later, Neve extended Bernstein's work to infer full key recovery from the first and second round of AES-128 [24] and also adapted Osvik's work to single-threaded processors by tweaking the OS-scheduling and to the last round of AES-128 [25]. [21] attacked the last round of AES-128 through collisions in the cache. [16,17,18] presented the *trace-driven* attack, which infers information from the order in which some cache events occur. However, to the best of our knowledge, no practical trace-driven attacks have been demonstrated so far.

A range of countermeasures have been put forward to protect against the different cache attacks, often by the authors of the attacks. [26] covers software mitigations that have been proposed by Intel Corporation and that have been incorporated into cryptographic tools and libraries (*e.g.* OpenSSL [27]).

3.1 Access-driven attacks

Access-driven attacks are generally implemented as follows. Two processes are being executed on the same processor; one is the target process performing a cryptographic operation (this process is usually referred to as *crypto*) and a second process that tries to steal crypto's secret key (this process is usually referred to as *spy*). To observe crypto's footprint in the cache, *spy* consistently evicts a portion of the cache in order to determine which cache lines (out of this portion) are used by *crypto*. With the corresponding ciphertext, *spy* can then infer information regarding *crypto*'s secret key. For instance, by knowing which cache lines are not accessed during the last AES round, it is possible to eliminate the key byte candidates that given the ciphertext would map onto those cache lines [19,25].

3.2 Time-driven attacks

In time-driven attacks, the spy process observes variations in the execution time of the crypto process and derives key material from estimating the number of key-dependent memory accesses that result in cache misses (and thus corresponding latency increases). For instance, by estimating cache collisions (which are memory accesses to memory locations stored in the same cache line) of the last AES round based on known ciphertext and a guess on a key byte, a statistical analysis that compares the estimate with the execution time will return the most likely key byte candidate [21].

4. Attacks strategy

Until now, all results on AES attacks were demonstrated with 128-bit keylength. This is due to certain extents because, as seen in Section 2, it is straightforward to roll back the 128-bit cipher key from the last round key. However, as far as we know no results have been presented on higher key lengths, despite the increasing number of devices and applications using 192- and 256-bit keys. Although classical cryptanalysis suggests an increase of resistance (at least against brute-force) with the number of key bits, no publication studies their resistance from the side-channel standpoint, would the channel be the cache behavior or any other. Therefore, in this following, we analyze and answer the following question: *What is the gain in security when increasing the cipher key length, with respect to cache-based side-channel attacks?*

As observed in section 2, only a 128-bit cipher key can be directly rolled back from the last round key; otherwise, two consecutive round keys (K_{i-1} and K_i) are required to roll back the cipher key for higher key size, *i.e.* for 192- and 256-bit key lengths.

Based on the results of previous works, we consider that the last round key K_{Nr} can be recovered from either access- or time-driven side-channel attacks using the techniques put forward in published works. Given the last round key K_{Nr} and the ciphertext, the last round is easily inverted in order to find the output X_{Nr} of round N_{r-1}. The side-channel, *i.e.* information about the cache behavior, can subsequently be used to find K_{Nr-1} similarly as used in the first part of the attack to find K_{Nr}. However, two facts make this second part more complex; both are due to the presence of the MixColumns operation in all but the last round:

- As observed in section 3, software implementations might use table T_4 for the last round. Some attacks (principally access-driven ones) on the last round focus their analysis on the behavior of this table, because it is only accessed in the last round. This is not the case for the other rounds, which are based on all the other tables T_0, T_1, T_2 and T_3. Therefore, the effect of the next to last round on those tables' accesses, which the attacker tries to observe, is mixed with the *noise* induced by accesses to the tables in the preceding rounds.
- Due to the absence of MixColumns in the last round, any byte of the ciphertext C and the corresponding byte of the last round key can lead to the value of one byte of X^{Nr}:

$$x^{Nr}{}_i = \text{SubBytes}^{-1}(\text{ShiftRows}^{-1}(x^{Nr+1}{}_j \oplus k^{Nr}{}_j)).$$

Therefore, the minimum key search space is 2^8 to estimate the input $x^{Nr}{}_i$ to the table T_4. Due to the MixColumns operation in the next to last round, however, 4 bytes of X^{Nr} (and 4 bytes of K^{Nr-1}) are involved in the computation of one byte of X^{Nr-1}:

$$x^{Nr-1}{}_i = \text{SubBytes}^{-1}(\text{ShiftRows}^{-1}(\text{MixColumns}^{-1}(y_j)),$$
$$\text{with } y_j = \{x^{Nr}{}_j | x^{Nr}{}_{j+1} | x^{Nr}{}_{j+2} | x^{Nr}{}_{j+3}\} \oplus \{k^{Nr-1}{}_j | k^{Nr-1}{}_{j+1} | k^{Nr-1}{}_{j+2} | k^{Nr-1}{}_{j+3}\}.$$

Therefore, the key search space to estimate the input to the tables T_0, T_1, T_2 and T_3 is (at first sight) is 2^{32}.

The former fact is particular to some implementations of AES and does not present a major issue, as successful attacks have been demonstrated on an AES-128 implementations not using T_4 in the last round. Contrarily, the latter fact, which implies a minimum key search space of 2^{32}, might be seen as making the attacks impractical or unrealistic. For instance, a time driven attack that would require 41,000 ($>2^{15}$) observations to see statistical differences and to select the correct key value, would result in a problem with complexity over $O(2^{15+32})$.

The simple trick used here is to leverage the fact that MixColumns is a linear operation, *i.e.* for any linear transformation $F : x \rightarrow y = F(x)$, it holds by definition that $F(a \oplus k) = F(a) \oplus F(k)$. The MixColumns can therefore easily be inverted and the previous attacks would result in a next to last round key *altered* by MixColumns. Indeed, the channel model can use

$$x^{Nr-1}{}_i = \text{SubBytes}^{-1}(\text{ShiftRows}^{-1}(x'^{Nr}{}_j \oplus k'^{Nr-1}{}_j))$$

with $x'^{Nr}{}_j$ a byte of $\text{MixColumns}^{-1}(\{x^{Nr}{}_j | x^{Nr}{}_{j+1} | x^{Nr}{}_{j+1} | x^{Nr}{}_{j+1}\})$ to find $k'^{Nr-1}{}_j$ where $k'^{Nr-1}{}_j$ is a byte of $\text{MixColumns}^{-1}(\{k^{Nr-1}{}_j | k^{Nr-1}{}_{j+1} | k^{Nr-1}{}_{j+1} | k^{Nr-1}{}_{j+1}\})$ with a key search space of 2^8. Once all the $k'^{Nr-1}{}_j$ are retrieved, the MixColumns operation can be applied to find K^{Nr-1}. Hence, the complexity is reduced to some factors of 2^8.

This observation permits us to state the main observation: *The security increase provided by larger key lengths is marginal for AES, in term of complexity and computation costs.*

We give quantitative data in the next section to prove the negligibility of the complexity increase. However, we conclude this section by stressing that this statement does not only apply in the case of cache-based attacks: other side-channels like power or electromagnetic attacks can leverage this observation to attack AES-192 and AES-256. The number of measurements to find both round keys depends on the precise implementation of the two rounds. For instance, when using masked outer rounds 29], one would need a significantly smaller number of measurements to find the next-to-last round key than to find the last round key.

5. Results: Attack complexity for AES-192 and AES-256

We consider here a software implementation based on 5 precomputed lookup tables each of size 1024 bytes running on a processor with cache lines of size 64 bytes. Extensions to other parameters are straightforward.

5.1 Access-driven

As previously observed [25], the probability p of a single cache line not being loaded into the cache after k accesses to l cache lines is $(1-1/l)^k$. As a result, the expected number of cache lines that are not loaded becomes $l \cdot (1-1/l)^k$. The last round key is inferred from analyzing the accesses to table T_4 that occupies 16 cache lines (we assume the data are aligned). Hence, the probability p for a specific cache line corresponding to table T_4 not to be accessed after 16 accesses to the table is

$$p = (1\text{-}1/16)^{16} = 0.356$$

The expected number E of cache lines corresponding to table T_4 that are not accessed during the last round is given by

$$E = 16 \cdot p = \ 5.70$$

Since each cache line contains 16 table elements, each not-accessed cache line leads to 16 *wrong* key-byte candidates that can be eliminated. However, those numbers represent the ideal case as they do not integrate the presence of noise in the measurements, *i.e.* cache lines evicted due to other processes sharing the cache. Therefore, a practical attack needs to take the effect of noise in account. Here, we use a simple voting process: after more than T votes as *wrong*, a key-byte candidate is discarded. The threshold T is linked to the noise level and should be adapted accordingly. The number of not-rejected key-byte candidates $\#_K$ can be expressed as a function of the number of observations n by the following function with threshold T as parameter where C_n^t is the binomial coefficient expressing the number of combinations of t items that can be selected from a set of n items.

$$\#_K = 256 \cdot \sum_{t=0}^{T} C_n^t \cdot p^t \cdot (1-p)^{n-t}$$

Figure 3 shows the number of remaining key-byte candidates obtained through the function above and obtained through experimental results of an access driven cache attack for different values of the threshold T. The function closely matches the empirical data. The figure shows that very few observations are required to find the last round key. The figure also shows that increasing the threshold, and thus increasing the confidence level that the right key has been chosen, does not increase the number of observations significantly.

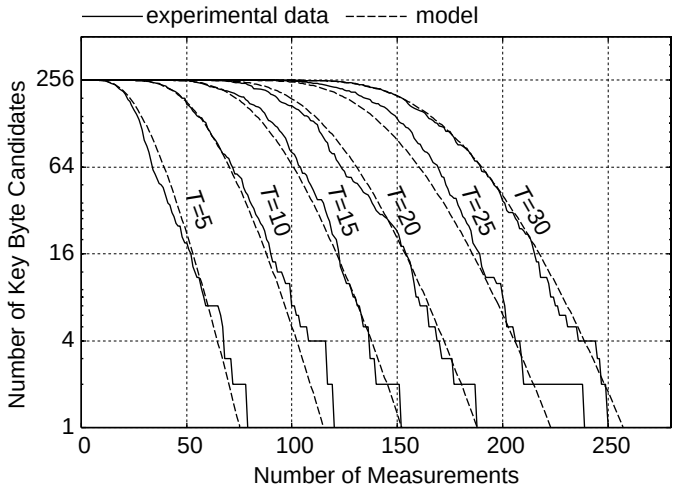

Fig. 3. Required number of measurements to eliminate incorrect last round key byte values.

When attacking the next to last round, the probability for a cache line to not be accessed decreases as the number of accesses k to the table of interest increases to 44 and 52 for AES-192 and AES-256 respectively. As a result, p becomes 0.058 and 0.035 and E becomes 0.94 and 0.56 respectively. The function above still holds and approximates the empirical data, as confirmed by figure 4 which shows the required number of measurements to eliminate incorrect next to last round key byte values of AES-256 obtained through the function above and obtained through experimental results of an access driven cache attack for different values of the threshold T.

The experimental results show that the required number of measurements N to successfully find the cipher key is set by finding the key of the next to last round, which requires 5 times as much measurements for AES-256 than to find the last round key of any AES implementation. Since the computational complexity of the attack is proportional to the minimum key search space and the number of measurements and must account for the fact that both K^{Nr} and K^{Nr-1} must be found, AES-256 is 6 times harder to attack than AES-128.

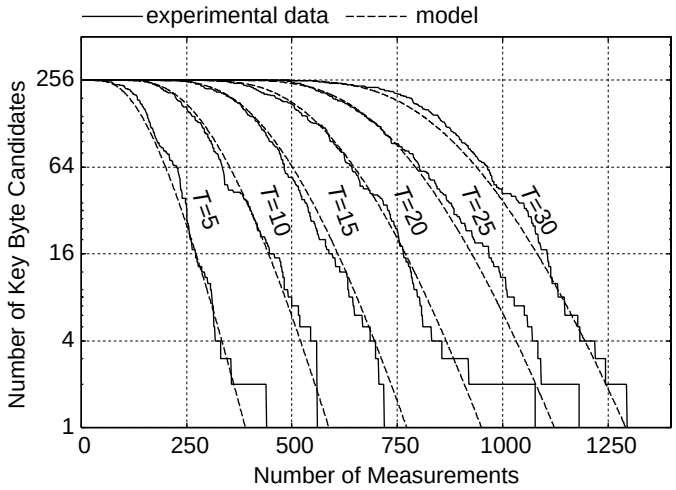

Fig. 4. Required number of measurements to eliminate incorrect next to last round key byte values of AES-256.

5.2 Time-driven

A recent publication provides an accurate analytical model predicting the minimum number of measurements to infer the cipher key in a time-driven attack, with respect to the cache parameters and the precomputed lookup tables sizes in software implementations [28].

With this model, it can be calculated that the required number of observations N to mount a successful time-driven attack, which analyses cache collisions in the last round using the cache line estimation model, is equal to 6,592 for AES-128 given that per encryption there are 36 (*i.e.* $4 \cdot (N_r-1)$) accesses to each of the tables T_0, T_1, T_2 and T_3 and 16 accesses to table T_4.

In case of AES-192, there are 44 accesses to each of the tables T_0, T_1, T_2 and T_3 and 16 accesses to table T_4. As a result, there are 5,071 observations required when analyzing cache collisions of table T_4 to find the last round key and 18,850 observations when analyzing cache collisions in T_0, T_1, T_2, T_3 to find the next to last round key.

In case of AES-256, there are in 52 accesses to each of the tables T_0, T_1, T_2 and T_3 and 16 accesses to table T_4. As a result, there are 3,917 observations required when analyzing cache collisions of table T_4 to find the last round key and 40,912 observations when analyzing cache collisions in T_0, T_1, T_2, and T_3 to find the next to last round key.

The reason that less measurements are required to find K^{Nr} for AES-192 and AES-256 then for AES-128 is that there are more accesses to T_0, T_1, T_2, and T_3 and as a result there is less noise due to a different number of cache misses to these tables. The time variations being analyzed are thus mainly due to cache misses in the table of interest T_4.

Since the computational complexity of the attack is proportional to the minimum key search space and the number of measurements and also must account for the fact that both K^{Nr} and K^{Nr-1} must be found, AES-256 is 7 times harder to attack than AES-128. Note also that the measurements used to find K^{Nr} can be reused to find K^{Nr-1}.

6. Conclusions

In this paper, we addressed side-channel attacks on AES-256: we demonstrated with practical results that the complexity (*i.e.* resistance) increase with the number of key bits is virtually non-existent. In particular, for the cache based attacks, an attack on AES-256 is only 6 to 7 times as hard as an attack on AES-128 both in the required computing power as in the required number of observations. We used the cache side-channel as an example side-channel, but the methodology presented in this work can be applied to leverage any other channel and attack AES-256.

Acknowledgements

The authors would like to thank Gary Graunke for a fruitful discussion.

References

1. Cryptographic Key Length Recommendations, http://www.keylength.com/ <online>
2. National Institute of Standards and Technologies, “Advanced Encryption Standard (AES)”, FIPS 197, November 2001.
3. National Institute of Standards and Technologies, “Recommendation for Key Management”, NIST Special Publication 800-57 Draft, 05/2006.
4. National Institute of Standards and Technologies, “Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher”, NIST Special Publication 800-67 Version 1, 05/2004
5. European Network of Excellence in Cryptology, “Yearly Report on Algorithms and Keysizes (2005)”, D.SPA.16 Rev. 1.0, IST-2002-507932 ECRYPT, 01/2006
6. DCSSI “Mécanismes cryptographiques - Règles et recommandations standard”, Rev. 1.02, 11/2004.
7. J. T. Trostle. “Timing attacks against trusted path”. In Proceedings of the IEEE Symposium on Security and Privacy, May 1998.
8. J.F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, “A practical implementation of the timing attack” In CARDIS '98: Proceedings of The International Conference on Smart Card Research and Applications, pp. 167-182, Springer-Verlag.
9. Joan Daemen and Vincent Rijmen. *The design of Rijndael, AES - The Advanced Encryption Standard*. Information Security and Cryptology. Springer, 2001.
10. P. C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems” CRYPTO, LNCS 1109, pp. 104–113, 1996.

11. Paul C. Kocher and Joshua Jaffe and Benjamin Jun, “Differential Power Analysis” M. Wiener *(ed.)*, Advances in Cryptology – CRYPTO '99, Springer-Verlag, 1999 , LNCS , 1999 , 388-397
12. Karine Gandolfi and Christophe Mourtel and Francis Olivier, “Electromagnetic Analysis: Concrete Results”. In Ç Koç and D. Naccache and C. Paar *(ed.)*, Cryptographic Hardware and Embedded Systems - CHES 2001, Springer-Verlag, 2001 , LNCS 2162 , 251-261.
13. Dan Boneh and Richard A. DeMillo and Richard J. Lipton, “On the Importance of Checking Cryptographic Protocols for Faults”, In Lecture Notes in Computer Science, 1233, pp. 37-51, 1997.
14. Auguste Kerckhoffs. “La cryptographie militaire”. Journal des sciences militaires, IX(1):5–38, January 1883.
15. D. Page, “Theoretical use of cache memory as a cryptanalytic side-channel”, Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002.
16. G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo, “AES Power Attack Based on Induced Cache Miss and Countermeasure”, International Symposium on Information Technology: Coding and Computing (ITCC 2005), pp. 586-591 April 2005.
17. C. Lauradoux, “Collision attacks on processors with cache and countermeasures”, Western European Workshop on Research in Cryptology (WEWoRC 2005), pp. 76-85, July 2005.
18. O. Acıiçmez, and Ç. K. Koç.. Trace driven cache attack on AES. IACR Cryptology ePrint Archive, Report 2006/138, April 2006.
19. Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, “Cryptanalysis of DES Implemented on Computers with Cache”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS 2779, pp. 62-76, September 2003.
20. D. J. Bernstein, “Cache-timing attacks on AES”, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf <online>, April 2005.
21. J. Bonneau and I. Mironov, “Cache-Collision Timing Attacks against AES”, in Louis Goubin and Mitsuru Matsui (eds), Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2006, Lecture Notes in Computer Science 4249, Springer.
22. C. Percival, “Cache missing for fun and profit”, BSDCan 2005, http://www.daemonology.net/papers/htt.pdf <online>, May 2005.
23. D. A. Osvik, A. Shamir, and E. Tromer, “Cache Attacks and Countermeasures: The Case of AES”, Cryptographers’ Track - RSA Conference (CT-RSA 2006), LNCS 3860, pp. 1-20, February 2006.
24. Michael Neve, Jean-Pierre Seifert, and ZhenghongWang. “A refined look at Bernstein’s AES side-channel analysis”. In Proceedings of AsiaCCS 2006.
25. M. Neve and J.-P. Seifert, “Advances on Access-driven Cache Attacks on AES”, Selected Areas of Cryptography (SAC 2006), August 2006.
26. E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert, “Software mitigations to hedge AES against cache-based software side channel vulnerabilities” Cryptology ePrint Archive, 2006.
27. OpenSSL Project, http://www.openssl.org/ <online>.
28. Kris Tiri, O. Acıiçmez, M. Neve, F. Andersen, “A Mathematical Model for Time-Driven Cache Attacks”, In Proceedings of Fast Software Encryption 2007..
29. J. Coron, and L. Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis", CHES 2000, LNCS 1965 , pp. 231-237.