Collision attacks on processors with cache and countermeasures

Cédric Lauradoux

INRIA Rocquencourt, Projet CODES
www-rocq.inria.fr/codes/
cedric.lauradoux@inria.fr

Abstract: Implementing cryptographic algorithms is a difficult problem since additional secret information can be recovered from some physical characteristics of a cryptographic device. Among all side-channel attacks, collision attacks and cache attacks are the most recent ones. The first technique uses side-channel information to detect internal collisions related to the algorithm. The second one exploits timing or power consumptions related to the memory accesses. This paper presents a new attack on the first round of AES based on power analysis, which combines both collision attacks and cache attacks. It provides many linear relations between the secret key bits from the encryption of a few chosen plaintexts. For instance, for a classical implementation using 4 lookup tables on a processor with 64-byte cache blocks, 48 linear relations involving half of the key bits are derived. Some countermeasures which defeat such attacks are also presented.

C. Wolf, S. Lucks, P.-W. Yau (Eds.): WEWoRC 2005, LNI P-74, pp. 76–85, 2005.

1 Introduction

Recent works [SWP03, SLFP04, LMV04] have introduced collision attacks against implementations of cryptographic primitives. The underlying issue raised by those attacks is the following: side-channel analysis provides a way to detect internal collisions. Such collision attacks were originally described by H. Dobbertin [Dob98]. The power of collision attacks have been demonstrated on smart-card processors [SLFP04, SWP03].

But, the next generation of smart-card processors [DF01, Phi] will embed cache memory in order to improve memory instruction latency. This mechanism can be used by the attacker to recover some information on the secret embedded in the device as pointed out in [Pag02, TSS$^+$03]. Actually, timing and power consumption are affected depending on whether the memory accesses are performed inside or outside the cache. Such attacks on AES have been described in [Ber05, BZB$^+$05].

The aim of this paper is to identify how an attacker can exploit the power variations related to memory accesses in the AES. We show that the attacks described in [Ber05, BZB$^+$05]

actually correspond to partial collision attacks. Moreover, we describe a feasible cache attack based on power analysis which provides an important number of linear relations between the secret key bits from a few chosen plaintexts. Indeed the actual cache attacks [BZB+05, OST05] can be considered as invasive side channel attack since the attacker influence the behavior of the cache in some way (Hyperthreading...). We present an attack that does not require this assumption. This obviously reduces the amount of information collected. But in another hand the number of encryptions required to recover information on the secret key is considerably reduces.

2 AES implementation

Several implementations of the AES are described in [DR02]. They target different platforms and make use of some tables to accelerate computation. The AES round function is composed of 4 basic transformations: ByteSub, ShiftRow, MixColumn, AddRoundKey. The implementation of ShiftRow and AddRoundKey is direct on 8-bit platforms, but MixColumn and ByteSub require to use two lookup tables of 256 bytes. On 32-bit platforms, the round transformation can be expressed as:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-C_1}] \oplus T_2[a_{2,j-C_2}] \oplus T_3[a_{3,j-C_3}] \oplus k_j \quad (1)$$

where the output e and the round key k are decomposed into four 32-bit words denoted by e_j and k_j respectively. The $(a_{i,j})_{i,j}$ correspond to the byte decomposition of the internal state of the AES. Further details on AES implementation can be found in [DR02]. Tables T_0 to T_3 are defined as follows:

$$T_0[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \bullet 03 \\ S[a] \\ S[a] \end{bmatrix}$$

$$T_2[a] = \begin{bmatrix} S[a] \\ S[a] \bullet 02 \\ S[a] \bullet 03 \\ S[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 02 \\ S[a] \bullet 03 \end{bmatrix}$$

Where $\bullet$ is the multiplication over GF(2^8). From these definitions, Daemen and Rijmen [DR02] propose another possible round implementation with a single table T_0 only:

$$e_j = k_j \oplus T_0[a_{0,j}] \oplus (T_0[a_{1,j-C_1}] \oplus (T_0[a_{2,j-C_2}] \oplus (T_0[a_{3,j-C_3}] \lll 8)) \lll 8) \lll 8 \quad (2)$$

We obtain 4 tables of 256 words for implementing (1) and a single table of 256 words for implementing (2). Evaluating the performance of these implementations on a target with cache is difficult. The location of the tables in the memory hierarchy has actually a great impact on the efficiency of the implementations of Formulas (1) and (2). This can

be pointed out by the following simple experiment: we perform several AES encryptions where the cache is flushed from all the tables before each encryption. The corresponding timing observation is given on Figure 1. The execution irregularities come from the

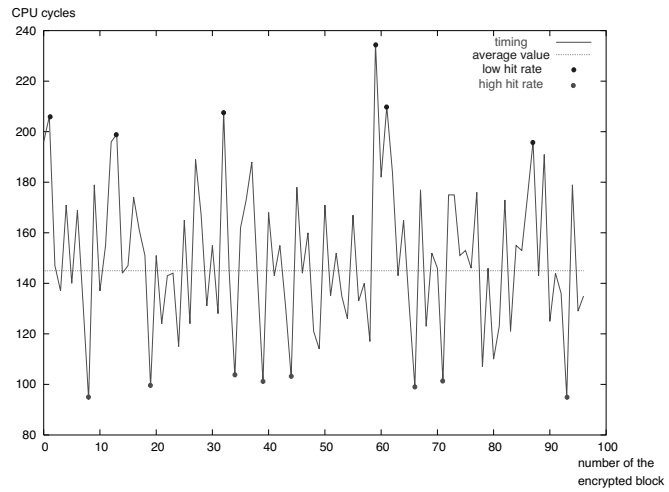

Figure 1: Simple timing observation on a PowerPC 7457

memory accesses. Since the gap between processor memory and high speed memory is important [HP96], processors embed cache memory to reduce memory accesses latency. However the amount of data stored in the cache is limited. Then, additional penalties may occur when large arrays of data are involved. If the accessed data still lies in the cache, we have a cache hit, otherwise it is a cache miss. Another well-known effect of the cache on computation time is the Cold start miss [Hil87]. This effect is also known as the anticipation miss: at the beginning of the execution of a process, data and instructions are outside the cache. The first memory accesses of a process are then heavily affected by memory latency. Moreover, many other parameters may interfere in the efficiency of the cache ([HP96, Hil87]): size of line, associativity, size of cache, victim buffer, pre-fetching engine... In this paper we will always consider that the data involved by the AES algorithm fit the cache. This mean that there is no Capacity misses or Conflict misses ([Hil87]). This assumption is important for performance achievement. An AES implementation with data that would not fit the cache would not be fast. This is why this important assumption is not always clearly detailed in all the recent works. Figure 2 illustrates the difficulty to observe the timing of events occurring in a processor. We first observe a sequence of cache flushes and encryptions. When the execution time is low, we may deduce that the memory accesses were fast. However, when we observe the execution time obtained by encrypting the same sequence of plaintexts several times and then plot the average value for each plaintext, we see that only some irregularities remain and that other events seem to be more sensitive to the execution context. It points out that many parameters may affect the execution time but some events appear to be constant. It clearly appears from these simulations that a single timing observation is not enough to obtain a good understanding

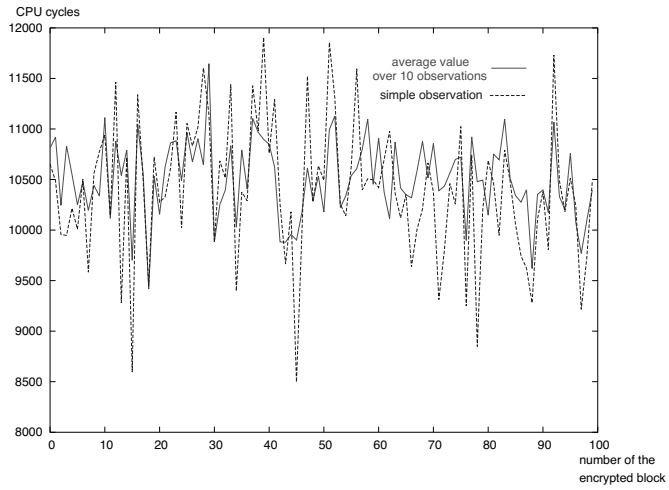

Figure 2: Average value versus simple observation on a Pentium 4

of the phenomena with a high confi dence rate. Several executions are required to give an accurate view of what happens within the processor. It follows that chosen plaintext attacks are much more powerful in this context since the same plaintexts can be encrypted several times, providing many observations of the same phenomenon. In known plaintext attacks where only a single observation of each encryption is available, the observation time must be much longer. Actually, as described in [BZB+05], power analysis offers a much better perspective than timing analysis.

3 Simple power analysis of the first round of AES

As described in [SWP03, SLFP04, LMV04], power analysis or timing analysis enables to detect internal collisions within a function f, where f is either the round function of a block cipher or a subfunction of the round. In the context of an implementation on a micro-controller, we say that a partial collision occurs if for an input word w the Hamming distance with the resulting word $w' = f(w)$ is low. This defi nition is due to the fact that power consumption is very similar in this particular case.

We now defi ne the addressing function for a cache memory and the notion of collisions on this addressing function. The internal structure of a cache is composed of block B of size s_b. To access an element e of size s_e at address d in a data set S, an addressing function g selects a block and performs an offset into the block to output e. The internal offset and the block selection respectively require $\lceil \log_2(\frac{s_b}{s_e}) \rceil$-bits and $\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil$-bits where n is the number of elements of size s_e in S. For instance, for 64-byte cache blocks, each cache block contains 16 elements of a table of AES. The 4 most signifi cant bits of the

address select the cache block and the 4 less significant correspond to the offset. Those assumptions hold only if the first element of the table correspond with the beginning of a cache block. Actually, a cache block may contain some data from another data set. In this case, our assumptions remain true if we simply add an external offset l to the address before the block selection and the internal offset.

A cache collision occurs when the same block is accessed for two addresses d and d'. This definition is motivated by the fact that power consumption and timing highly vary depending on whether the memory accesses are performed in the same cache block or not. Indeed cache collisions can be directly observed using power consumption analysis [BZB+05]. We will now focus our attention on the addressing function and on the block selection for the first round of the AES.

3.1 Implementation involving 4 tables

We will now demonstrate the ability of an attacker to recover linear relations between the $\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil$ most significant bits of each key byte using power analysis. The internal state of the AES before the first access to a lookup table is obtained by a simple xor between the secret key and the plaintext. The explicit formulas describing the first encryption round are given in Figure 3, where $p_{i,j}$ and $K_{i,j}$ are respectively the plaintext bytes and the encryption key bytes.

$$
\begin{aligned}
e_0 &= T_0[p_{0,0} \oplus K_{0,0}] \oplus T_1[p_{1,1} \oplus K_{1,1}] \oplus T_2[p_{2,2} \oplus K_{2,2}] \oplus T_3[p_{3,3} \oplus K_{3,3}] \oplus k_0 \\
e_1 &= T_0[p_{0,1} \oplus K_{0,1}] \oplus T_1[p_{1,2} \oplus K_{1,2}] \oplus T_2[p_{2,3} \oplus K_{2,3}] \oplus T_3[p_{3,0} \oplus K_{3,0}] \oplus k_1 \\
e_2 &= T_0[p_{0,2} \oplus K_{0,2}] \oplus T_1[p_{1,3} \oplus K_{1,3}] \oplus T_2[p_{2,0} \oplus K_{2,0}] \oplus T_3[p_{3,1} \oplus K_{3,1}] \oplus k_2 \\
e_3 &= T_0[p_{0,3} \oplus K_{0,3}] \oplus T_1[p_{1,0} \oplus K_{1,0}] \oplus T_2[p_{2,1} \oplus K_{2,1}] \oplus T_3[p_{3,2} \oplus K_{3,2}] \oplus k_3
\end{aligned}
$$

Figure 3: The first round of the AES implemented with four tables

The most relevant property of the first AES round is that the addressing function depends on the plaintext and on the secret key. Since each cache block contains $\frac{s_b}{s_e}$ elements of a lookup table, the index of the cache block accessed is given by:

$$\frac{(p_{i,j} \oplus K_{i,j}) \times s_e}{s_b}.$$

During the first round, we perform 4 accesses to each table which involve 4 different key bytes. For each table, we may observe miss or hit on the cache using simple power analysis. We consider that the cache does not contain any part of the tables. This assumption is very easy to assert since the cache memory is volatile: if we just remove the supply voltage of a smart card, we completely discard the contents of the cache. We may obtain the following trace (Figure 4) for cache accesses.

The first 4 accesses always correspond to cache misses since they are Cold start misses. For the remaining accesses, we obtain 8 possible scenarios for each lookup table. If we observe

access	T_0	T_1	T_2	T_3
1	miss	miss	miss	miss
2	miss/hit	miss/hit	miss/hit	miss/hit
3	miss/hit	miss/hit	miss/hit	miss/hit
4	miss/hit	miss/hit	miss/hit	miss/hit

Figure 4: Trace of memory access

a hit on any of the second accesses to T_0 for instance, we deduce that the $\log_2(\frac{n \times s_e}{s_b})$ most significant bits of $(p_{0,0} \oplus K_{0,0})$ and of $(p_{0,1} \oplus K_{0,1})$ are equal. A hit on the third access is related either to the first access or to the second one, and a hit on the last access can be related to any of the previous accesses. It is then obvious that the attacker only needs to modify the values of the most significant bits of $p_{0,0}$, $p_{1,1}$, $p_{2,2}$, $p_{3,3}$ to obtain a hit for each access. Moreover, all 4 parts of the plaintext can be changed simultaneously, since the observation on the 4 tables are independent from each other. Therefore, the attack obtains $\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil \times 12$ linear relations which involve $\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil \times 16$ bits of the secret key. These relations can be obtained from $2^{\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil}$ encryptions. In the case of a 64-byte cache block, 16 encryptions lead to 48 linear relations on 64 key bits. The involved bits correspond to the most significant part of each key byte. The obtained linear relations can be directly exploited for decreasing the complexity of another attack. They lead for instance to an exhaustive search with complexity 2^{80}. Obtaining the exact value of the 64 involved key bits by some side-channel information is still an open problem.

3.2 Implementation involving a single table

Things seem more difficult in the case of the implementation using a single lookup table since we only access one table during the full first round. For convenience, we give in Figure 5 the description of the first round with a single table implementation.

$$e_0 = k_0 \oplus T_0[p_{0,0} \oplus K_{0,0}] \oplus (T_0[p_{1,1} \oplus K_{1,1}] \oplus (T_0[p_{2,2} \oplus K_{2,2}] \oplus (T_0[p_{3,3} \oplus K_{3,3}] \lll 8)) \lll 8) \lll 8 \quad (3)$$
$$e_1 = k_1 \oplus T_0[p_{0,1} \oplus K_{0,1}] \oplus (T_0[p_{1,2} \oplus K_{1,2}] \oplus (T_0[p_{2,3} \oplus K_{2,3}] \oplus (T_0[p_{3,0} \oplus K_{3,0}] \lll 8)) \lll 8) \lll 8 \quad (4)$$
$$e_2 = k_2 \oplus T_0[p_{0,2} \oplus K_{0,2}] \oplus (T_0[p_{1,3} \oplus K_{1,3}] \oplus (T_0[p_{2,0} \oplus K_{2,0}] \oplus (T_0[p_{3,1} \oplus K_{3,1}] \lll 8)) \lll 8) \lll 8 \quad (5)$$
$$e_3 = k_2 \oplus T_0[p_{0,3} \oplus K_{0,3}] \oplus (T_0[p_{1,0} \oplus K_{1,0}] \oplus (T_0[p_{2,1} \oplus K_{2,1}] \oplus (T_0[p_{3,2} \oplus K_{3,2}] \lll 8)) \lll 8) \lll 8 \quad (6)$$

Figure 5: The first round of the AES implemented with a single table

The attacker may first apply the previous attack to obtain linear relations between the most significant parts of the bytes involved in the first equation. He will only change $p_{0,0}$ to guess the equations related to $K_{0,0}$, $K_{1,1}$, $K_{2,2}$, $K_{3,3}$. This requires $2^{\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil}$ encryptions. Those trials involved at least a case with 2 hits for the first equation. From this case, the attacker seek a case with 3 hits for the first equation. This cost $2^{\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil + 1}$. Having this done, he can recover relation on the secret key related to Equation (4). Indeed

the attacker want to generate a case with one miss and 15 hits. He will perform an exhaustive search on equations (4), (5), (6) to generate this case. Compared to the attack on the four tables's implementation, this only increases the number of required encryptions to $15 \times 2^{\lceil \log_2(\frac{n \times s_e}{s_b}) \rceil}$ for obtaining the same amount of information on the secret key. With 240 encryptions on a 64-byte cache block we obtain 60 linear equations on 64 key bits. Then the cost of exhaustive search is reduced to 2^{68}.

4 Countermeasures

The first obvious countermeasure consists in replacing lookup tables by operations. This countermeasure is simply not effective since the collisions attack described in [SLFP04] still applies. Actually, in presence of cache memory, classical collisions attacks are still possible. Another important reason to avoid this solution is the heavy cost for execution time. Now, we will detail several suitable solutions for reducing the impact or defeating those attacks.

- Dummy accesses to a table can increase the noise in order to decrease the ability of the opponent to distinguish the activity related to the key from other . This technique was proposed in [Pag03]. When the dummy array is twice larger than the cache, then the hit probability is $\frac{1}{2}$. For each access to the lookup tables we make an access to the dummy array. This technique has two drawbacks:
  - the random number generator must be fast and unbiased. A biased generator does not counter the attack.
  - dummy data may interact with the lookup tables. This can create conflict memory access and important CPU idling.
- Warming up the processor may guarantee the presence of the tables in cache by performing several accesses in the table before the computation. This solution was proposed in [BZB+05, Pag02]. This can be efficient only for small tables and if the size of the data are not too big to affect the miss ratio. If an interrupt occurs during the execution and flushes the cache then the warm-up effect is annihilated. Warm-up does not guarantee the location in the cache of the data.
- Increasing the size of cache block provide a way to reduce the information leak by the device.
- A collision is the relation between 2 elements that affects the timing or the power consumption of the algorithm. On processors, it corresponds to several accesses on the same cache line. Clearly, this can not be avoided. But, collision attacks can be defeated by masking the relationship between the data involved in a collision. This can be viewed as an encryption or randomization of the addressing function of the cache memory. This can be achieved by adding several implementation secret keys (these implementation keys are local parameters and must not be transmitted since

they do not influence the ciphertext). Here, each key is a parameter in a permutation of the inputs of the lookup table. Then, we access the table through the permutation. Those permutations can be implemented efficiently using another lookup table or in a different way. With those permutations, the attacker still detects the collisions but he is not able to find the underlying relation if he does not know the permutation table. He has to perform $\binom{n}{k}$ trials where n is the number of elements in the table and k the number of elements per cache line, for recovering the composition of each line of the lookup table. For the AES table, the number of trials is more than 2^{135} (256 elements per table and 32 elements per L2 data cache line of a Pentium 4). But now the opponent can try to find collisions on the permutation if we use a table. It is more difficult because the size of the permutation table is smaller: n elements of size $\log_2(n)$ and we can use other methods to implement the permutation.

- Prefetching seems to be an interesting solution since we can improve the performance of the algorithm and confuse the attacker since it would not be easy to distinguish a collision from something else. We try to hide memory latencies by anticipating accesses. This is very similar to warm-up technique but the cost of prefetching is lighter so it can be included inside the encryption loop. When any interruptions occur, we have a Cold start miss. By using prefetch we will reduce the penalties of Cold start miss.

5 Conclusion

We demonstrate how an opponent can take advantage of the optimization mechanism of a processor. The addressing function of a cache memory allows an attacker to obtain many linear relations between the secret key bits in the AES. This also holds for any cryptographic algorithm which uses some memory tables in order to increase the implementation performance. One important assumption not mentioned in our paper is that all tables lie at the same time in the cache. If the capacity of the cache memory is not sufficient to contain the tables then additional misses will occur. They are referred as conflict misses in processor architecture [Hil87]. If Conflict misses occur the effect of cache parameters (replacement policies, associativity...) may be important. This field has not been explored yet.

Partial collision attacks seem quite powerful as they apply in different contexts, in simple power analysis [SWP03, SLFP04] directly on a cryptographic function or on some related functions like the cache addressing functions.

Recent works [Ber05] on cache timing attacks are also related to the first round on the AES. But, the feasibility of such a timing attack is still questionable.

We give several possibilities to defeat those attacks. All proposed countermeasures have a computational cost. Among all these countermeasures the most interesting one is the permutation solution because it is based on a rigorous security argument.

Acknowledgment

The author wants to thank Anne Canteaut and the anonymous referees for their comments.

References

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES, 2005.

[BZB+05] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. In *ITCC (1)*, pages 586–591, 2005.

[DF01] Jean-François Dhem and Nathalie Feyt. Hardware and Software Symbiosis Helps Smart Card Evolution. *IEEE Micro*, 21(4):14–25, 2001.

[Dob98] Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[Hil87] Mark Hill. *Aspect of cache memory and instruction buffer performance*. PhD thesis, University of California, Berkeley, 1987.

[HP96] John Hennessy and David Patterson. *Computer Architecture: A quantitative approach*. Morgan Kaufmann Publisher, Inc, 1996.

[KSWH00] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side Channel Cryptanalysis of Product Ciphers. *Journal of Computer Security*, 8(2/3), 2000.

[LMV04] Hervé Ledig, Frédéric Muller, and Frédéric Valette. Enhancing Collision Attacks. In *CHES 2004, LNCS 3156*, pages 176–190. Springer, 2004.

[OST05] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and Countermeasures: the Case of AES. Cryptology ePrint Archive, Report 2005/271, 2005. `http://eprint.iacr.org/`.

[Pag02] 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.

[Pag03] D. Page. Defending Against Cache Based Side-Channel Attacks. *Information Security Technical Report*, 8(1):30–44, April 2003.

[Per05] Colin Percival. Cache Missing For Fun And Profit. In *BSDCan*, 2005.

[Phi] Philips. HiPerSmart - 32 bit high performance smart card Ics.

[SLFP04] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A Collision-Attack on AES combining Side Channel- and Differential-Attack. In *CHES 2004, LNCS 3156*, pages 163–175. Springer-Verlag, 2004.

[SWP03] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In *FSE 2003, LNCS 2887*, pages 192–205, 2003.

[TSS+03] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In *CHES 2003, LNCS 2779*, pages 62–76, 2003.