Understanding cache attacks

Anne Canteaut, Cédric Lauradoux, André Seznec

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Understanding cache attacks

Anne Canteaut — Cédric Lauradoux — André Seznec

N° 5881

Avril 2006

Thèmes COM et SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5881--FR+ENG

INRIA
ROCQUENCOURT

Understanding cache attacks

Anne Canteaut [*] , Cédric Lauradoux[*] , André Seznec [†]

Thèmes COM et SYM — Systèmes communicants et Systèmes symboliques
Projets Codes et Caps

Rapport de recherche n° 5881 — Avril 2006 — 18 pages

Abstract: This paper points out that both the micro-architecture of the processor and the cache initial state impact the amount of side-channel information which is provided by analyzing the cache behaviour during a symmetric encryption. Therefore, the vulnerability of a block cipher implementation based on lookup tables highly varies with the encryption context and with the targeted platform. Our results then clarify some simulations reported by Bernstein and show that they can be reproduced only in a very particular context. However, we point out that some AES key bits can be recovered even if all lookup tables lie in the cache before each encryption, i.e., if all cache misses are avoided.

Key-words: Timing attacks, cache attacks, side-channel attacks, AES

[*] INRIA projet Codes - B.P. 105 - 78153 Le Chesnay cedex - France - `{Anne.Canteaut,Cedric.Lauradoux}@inria.fr`

[†] IRISA projet Caps - Campus de Beaulieu - 35042 Rennes Cedex - France - `seznec@irisa.fr`

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

Comprendre les attaques sur le cache

Résumé : Cet article montre que la micro-architecture du processeur et l'état initial du cache influent tous deux sur la quantité d'information fournie par l'analyse du comportement de la mémoire-cache au cours de l'exécution d'un algorithme de chiffrement symétrique. La vulnérabilité de l'implémentation d'un algorithme par blocs utilisant des tables varie donc fortement avec le contexte d'utilisation et avec la plate-forme visée. Nos résultats clarifient ainsi les simulations rapportées par Bernstein et montrent que celles-ci ne peuvent être reproduites que dans des contextes très particuliers. Toutefois, nous mettons en évidence la possibilité de retrouver certians bits de clef de l'AES même dans le cas où les tables se trouvent dans le cache avant chaque opération de chiffrement.

Mots-clés : Attaques par canaux cachés, attaques sur le cache, AES

1 Introduction

Side-channel attacks are cryptanalyses which take advantage of some physical information leaked by a cryptographic device (e.g. timing, power consumption...). Since the first work of Kocher [12], several classes of side-channel attacks like differential power analysis have been firmly established [8]. Until recently, most of these attacks are based on specific features of some software implementations of the basic transformations involved in the targeted algorithm. For instance, a timing attack against AES due to Koeune and Quisquater [13] applies if the MixColumns transformation uses a particular implementation of the multiplication in $GF(2^8)$.

But, it is usually believed that such weaknesses of AES do not appear for optimized implementations which are dedicated to 32-bit processors. In this case, both MixColumns and SubBytes transformations are implemented by lookup tables in order to decrease the encryption time. AES encryption and decryption then mainly consist of a sequence of memory accesses to these lookup tables. Unfortunately, memory accesses are not always performed in constant time. In particular, the use of cache memory has a great impact on the latency and on the power consumption of a memory access. Timing analysis or power analysis may then provide some side-channel information which can be exploited by the cryptanalyst, as mentioned by Kocher [12] and Kelsey *et al.* [11]. The first practical implementations of side-channel attacks based on the analysis of the cache behaviour for a block cipher are due to Tsunoo *et al.* [25] and Page [18]. Then, cache attacks against AES drew a lot of attention during the last three years [24, 1, 14, 2, 15].

However, all these recent works raise many open questions. Most notably, a relevant comparison between all presented cache attacks, both in terms of complexity and of practical applicability, is still missing. For instance, some of these attacks are dedicated to embedded devices, some others are limited to multi-user systems. Both the physical nature of the used side-channel information and the assumptions made on the initial state of the cache memory before encryption highly influence the applicability of a cache attack. Another open problem, which does not arise for classical side-channel attacks, is to determine which parameters in the micro-architecture of the processor impact the efficiency of the attack and whether the cryptanalysis can be mounted on all 32-bit processors or not.

In this paper, we show that cache attacks can be classified according to the conditions they require on the initial state of the cache memory. These conditions influence the practical applicability of the attacks and the type of devices on which they can be mounted. Moreover, we point out that the natures of the cache effects which are observed by the attacker highly vary with the cache initial state. Most notably, if each encryption is assumed to be performed from an empty cache or from a chosen cache initial state, then the timing or power analysis mainly detects cache misses. Due to the structure of the cache memory, such an attack against AES enables to recover the most significant bits of each key byte.

Conversely, if the only assumption on the cache initial state is that all lookup tables used for the encryption lie in the cache before each encryption, then such cache misses do not occur anymore. This hypothesis corresponds to the situation occurring in different contexts, for instance in remote timing attacks. However, even in this case, some timing variations may still be observed on superscalar processors and can be used for mounting an attack. We will show that these timing characteristics depend on the whole micro-architecture of the processor. The efficiency of the attack and the positions of the involved key bits then vary with the processor, the compiler and the implementation. This study partially clarifies some observations reported by Bernstein: some of the key bits which are recovered in the simulations presented in [1], especially the most significant bits of each key byte, are deduced from the analysis of cache misses. These cache misses are mainly due to the system calls and to some array manipulations which are performed before each encryption. Therefore, contrary to what was originally believed, the performance reported in [1] can only be achieved if the cache initial state is empty. When the cache is initially loaded with the lookup tables, most of these cache misses do not occur anymore. However, we present a variant of Bernstein's attack which applies in this case and which enables to recover some information on the other key bits. This information comes from specific features of the micro-architectures of some processors.

This paper is organized as follows. We first describe the behaviour of the cache memory and we show how it affects memory accesses. Section 3 investigates all previous work related to cache attacks including both timing attacks and simple power analyses. We propose a classification of cache attacks according to the cache effects they exploit. Among all those attacks, remote timing attacks are particularly important for cryptographic products and we discuss the possibility to mount such an attack against secret key algorithms. Then, Section 4 briefly recalls how AES is usually implemented by lookup tables. Section 5 investigates the influence of the cache initial state and of the micro-architecture on the cache timing attack described by Bernstein. A variant of Bernstein's AES timing attack is then presented in Section 6, which relies on a different assumption on the cache initial state. In view of the obtained results on the effectiveness of the attack, we conclude that it depends on the whole micro-architecture of the processor. We finally show that some classical countermeasures enable to thwart those cache timing attacks.

2 Basics on cache memory

Since the gap between the latency of memory and the speed of processors is still increasing [9], the bus bandwidth and the access speed to the main memory have become the limiting factors in the overall processor throughput. This bottleneck is overcome by cache memory. A cache is a small piece of high speed memory. It aims at keeping the CPU as busy as possible by minimizing the load/store latency to the main memory. Modern processors have two levels of on-chip cache, respectively called L1 and L2. Each of them may be dedicated to a special purpose (data or instructions), but in the so-called Harward architecture the L1 cache is usually split into an instruction cache and a data cache. The L2 cache can hold both instructions and data; it is larger but slower than the L1 cache.

A cache is divided into blocks (or lines) of fixed size; typical block sizes are 32, 64, 128 bytes. It is worth noticing that the the L1 and L2 caches may have different block sizes. The cache associativity determines how the main memory blocks map into cache blocks: an m-way associative cache is divided into sets of m blocks. A main memory block can then be mapped to any block of a given set. The block selection within a set is performed by a replacement algorithm like LRU [9]. For instance, Figure 1 describes a 4-way associative cache.

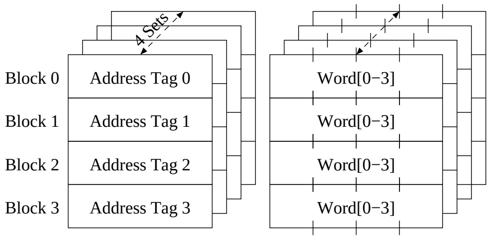

Figure 1: 4-way associative cache memory

An incoming address to the cache is split into three parts (Figure 2): a set number, an address tag and a block offset. To determine whether the accessed data is stored in the cache, its address tag is compared with the address tag of each block in the set. In the following, we consider an m-way associative cache of s_c bytes, divided into blocks, each of s_b bytes. In this case, the size of the block offset is $\lceil \log_2(s_b) \rceil$ bits and the size of the set number is $\lceil \log_2(\frac{s_c}{s_b \times m}) \rceil$.

The cache parameters affect the execution time of an algorithm that uses a data array of size S composed of elements of size s_e as follows. The array is mapped into the cache according to its alignment and to the size s_e of its elements. The alignment of a data can be considered as a constraint on its address: on most architectures the address of a data is a multiple of the size of the addressed data. The array is divided into $\lceil S/s_b \rceil$ blocks and each block can hold at most $\lceil \frac{s_b}{s_e} \rceil$ elements. Then, when an element of the array is accessed, the relative addressing can be seen as

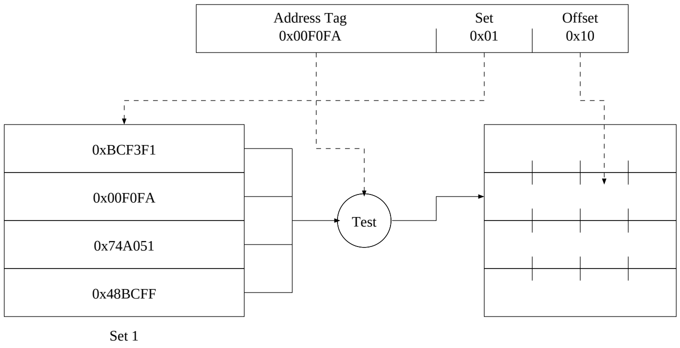

Figure 2: Addressing in a cache memory

a block selection and an offset into the block. The block offset requires $\lceil \log_2(\frac{s_b}{s_e}) \rceil$ bits and the block selection $\lceil \log_2(\frac{S}{s_b}) \rceil$-bits.

Many other memory parameters can affect the behaviour of the execution such as cache write policies, TLB.... Moreover modern processors feature very complex interactions between instructions. The microscopic execution time of any sequence of instructions also depends on these interactions. Further details on memory hierarchy can be found in [9, 10].

3 Different types of cache attacks and their applicability

When an element in a data array is accessed, two situations may occur. If this element lies in the cache memory, the access is called a hit; otherwise the block has to be loaded from the main memory, this situation is referred as a miss. The power consumption and timing of a device may highly vary depending on the memory level (L1 cache, L2 cache, main memory) where the memory accesses are performed. This means that when timing a sequence of memory accesses, the number of access hits can be successfully detected.

During the execution, we can distinguish three families of cache misses according to [10]:

- cold start misses, which arise for the first reference to a data;
- capacity misses, which occur if the size of the data array exceeds the size of the cache;
- conflict misses, which may happen only if accesses can provoke the eviction of recently accessed data.

In the rest of the paper, we will only consider cold start or conflict misses. This choice is motivated by the fact that symmetric encryption algorithms are designed to achieve a relative high speed. Therefore, all operations must be implemented by lookup tables whose total size do not exceed the cache capacity.

Under the previous assumption, the behaviour of the cache memory during an encryption (or decryption) depends both on the cache initial state and on the sequence of memory accesses which are performed. The assumptions made by the attacker on the cache initial state may highly vary according to the targeted cryptographic device. Since they strongly condition the practicability of the attack, we choose to classify cache attacks according to the corresponding requirements on the initial state of the cache:

- *Empty initial state (reset attacks)*: these attacks require that no table involved in the encryption algorithm be contained in the cache before any observation. This class of attacks is mainly based on the observation of cold start misses.

- *Forged initial state (initialization attacks)*: in these attacks the adversary must be able to trigger the cache into a known state before the encryption. This means that the attacker will generate a chosen number of cold start misses.
- *Loaded initial state (micro-architecture attacks)*: the cache already holds all the tables involved in the encryption algorithm.

How there three hypothesis realistically apply to different types of cryptographic devices will be discussed in the next sections.

For a given initial state, the sequence of memory accesses performed during the encryption can be observed by timing analysis or by power analysis as suggested in [11]. The power analysis allows the attacker to observe the encryption access by access whereas a timing attack gives a global measure of the events that occur during the encryption. As a result, a power analysis of the cache memory will obviously give more information than timing.

Another class of attacks has been proposed by Osvik *et al.* in [17]. Here, the attacker is able to directly observe the content of the cache with a software probe. In practice, this information is mainly obtained by using interactions between two data tables that share the same cache: the first array C will be the lookup table involved in the encryption algorithm; the second array D is controlled by the attacker. During the encryption the attacker performs a sequence of accesses to D. These accesses may suffer from conflict misses due to table C. Therefore, by performing chosen accesses to D, the attacker is able to generate some conflict misses which are related to the secret key. This kind of attacks can only be mounted on multi-threaded processors [26] such as the Pentium 4 HT. To carry such an attack the attacker must be able to read and trigger the cache either both before and after the encryption or during the encryption. In this sense, the practical applicability of these attacks is similar to the one of forged state attacks.

3.1 Attacks starting from empty cache

The complexity of resetting a cache memory (i.e., of a cache flush) depends on the target device. By nature, cache memory is volatile. As a consequence, simply removing the voltage supply of the device will clear the cache. This is only affordable on embedded systems, for instance on the most recent types of smart card processors [7, 19] or on Digital Signal Processors. On more complex systems like computers, the attacker needs a user account on the targeted host. Actually, the cache is reset by triggering all cache blocks with many memory accesses before any observation.

In this particular context, a cache miss occurs when the accessed element does not belong to a block which already lies in the cache. Since a cache block consists of a set of elements with the same $\lceil \log_2(\frac{s_d}{s_b}) \rceil$ most significant bits, a cache hit corresponds to a collision on the $\lceil \log_2(\frac{s_d}{s_b}) \rceil$ most significant bits of the inputs of the table. In this sense, an attack starting from an empty cache can be seen as a partial collision attack [21, 20] on the addressing function of a cache memory. For instance, on Pentium 3, the size of L1 data cache block is $s_b = 32$; thus, L1 cache misses for the AES lookup tables involve the 5 most significant bits of each key byte.

The first practical implementations of such attacks have been described against MISTY1 and DES by Page [18] and Tsunoo *et al.* [25]. In [25] Tsunoo *et al.* point out that a pair of plaintext/ciphertext which leads to a high miss ratio (i.e., to a long encryption time) provides an improbable value for the difference between some bits of the first and the last round keys. After several observations the correct partial difference can then be deduced from the least frequent value. This work also illustrates the impact of data mapping into the cache. DES Sboxes have 64 inputs of 1 byte, but better performance can be achieved when each entry is aligned on 4-byte boundaries. Unfortunately, this decreases the number of Sbox inputs per cache block, and it considerably affects the cryptanalysis as pointed out by [24]. This attack also successfully applies to MISTY1 and to the AES.

Another reset cache attack dedicated to embedded devices has been recently proposed by Lauradoux in [14]. It uses power analysis to recover linear relations on the most significant bits of all bytes of the AES secret key. The attack was demonstrated against different AES implementations and is similar to the timing attack of [24]. It exploits the fact that the first inputs of the Sbox lookup tables in AES are given by the xor between the plaintext and the secret key.

3.2 Attacks starting from initialized cache

In this class of attacks, the adversary must be able to initialize some chosen cache blocks with data from the lookup tables. This can be done by flushing the cache memory and then by performing fake encryptions with a known key in order to load certain table blocks into the cache. This requirement limits the scope of the attack to multi-users systems since an access to the cache memory is needed. With several chosen initializations and some power traces, Bertoni *et al.* [2] show how to recover the most significant bits of each key byte in AES.

This class of attacks is similar to the attacks that start from an empty cache in the sense that are based on the analysis of cache misses. The only difference is that they use conflict misses instead of cold start misses to gain some information on the most significant bits of the key bytes.

3.3 Attacks starting from loaded cache

Both previous classes of attacks can be thwarted by systematically loading the whole table before any encryption, as proposed in [2, 14]. This simple countermeasure avoids most cache misses. However, this does not imply that all timing variations have been removed, as pointed out by Bernstein [1]. For instance, Figure 3 shows that timing variations still exist for the original implementation of AES by Bosselaers, Rijmen and Barreto [3], on a Pentium 3 with a loaded cache initial state.

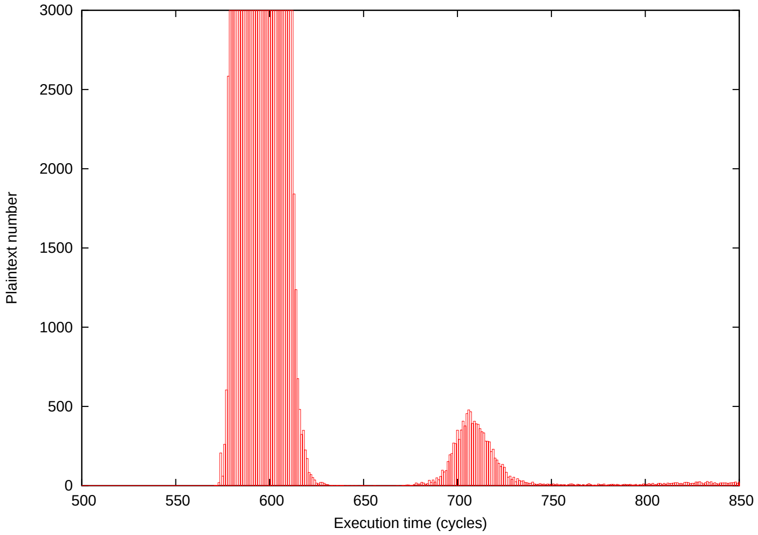

Figure 3: Distribution of timing for the original AES implementation on a Pentium 3 for 2^{20} plaintexts (the peak of the distribution is here truncated but almost reaches 2^{19})

These timing variations may be due to the fact that the cache memory on superscalar processors is designed for handling more that one access per cycle. Among all solutions which implement this feature [27, 23], the one used in X86 processors consists in splitting the cache into several independently addressed banks. Therefore, memory accesses can suffer from conflict penalties when several simultaneous concurrent references to the same bank occur. Such penalties affect and are affected by the whole micro-architecture of the processor (i.e., by pipeline, conflict banks detection logic, load/store queue...).

In some cases, the observed timing variations may also be due to the existence of some conflict misses: even if the total size of the lookup tables does not exceed the size of the L1 cache, some elements in the tables may be evicted from the cache during the execution if they lie in the particular set chosen by the replacement algorithm. Such conflict misses may potentially appear as soon as the total size of the tables exceeds the set size, but they usually occur with a non-negligible probability from half of the cache size, as we will see in Section 5. The existence of such conflict misses then depends on the size of the cache, but also on the cache associativity and on the replacement algorithm.

3.4 Remote timing attacks

An interesting extension of timing attacks is the class of remote timing attacks. Since timing attacks apply to weak implementations of a cryptographic algorithm, it is possible to transpose them to remote devices. The assumptions requested to mount a remote timing attack are completely different depending on whether we consider public or secret-key algorithms. In the case of public-key cryptography, assumptions are really weak. The attacking machine (Eve) sends a request to the targeted server (Alice) using Alice's public key (Figure 4). Then, by measuring the time to respond (TTR), Eve tries to deduce Alice's private key. This attack has been first explored by Brumley and Boneh against RSA [5].

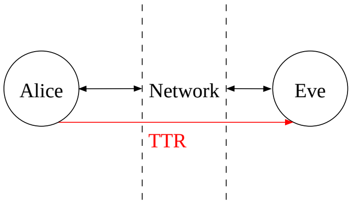

Figure 4: Remote attack against public key algorithms

On the contrary, secret-key algorithms cannot be attacked in such a way. Indeed, there is no particular reason to hope that some data encrypted with the secret key K_{AE} are voluntarily sent by Alice to Eve since Eve does not know the secret key K_{AE}. Therefore, such ciphertexts are available to the attacker only if they are eavesdropped in the context of a man-in-the-middle attack (Figure 5). For instance, Eve can probe a routing element of a communication channel between Alice and Bob. She will monitor the traffic on this communication channel to measure the response time of both parties.

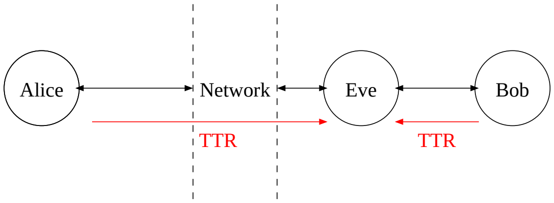

Figure 5: Remote attack against secret key algorithms

The most difficult task in all remote timing attacks is to evaluate the encryption time of the target: the attacker has to take into account the transmission time from the target to the attacking machine, but the noise added by the transmission delay to the encryption time is considerable. That is why in practice the only known remote timing attack [5] was mounted on a local network. The attack presented by Bernstein [1] bypasses this problem since it applies to a protocol in which the server's timestamp is also transmitted and available to the attacker. This feature then allows the attacker to directly access the encryption time without the noise added by the transmission through the network. This considerably weakens the security model of the target.

If we want to mount a remote cache timing attack, another important difficulty is to control the state of the cache before each encryption. A remote initialization or reset of the cache is not realistic. As a consequence, the only practical assumption in the context of remote attack is that the cache was loaded.

3.5 Previously proposed cache attacks against the AES

The previous discussion points out that the applicability of the different families of cache attacks highly depends on the type of the targeted device. For embedded devices (e.g. for Digital Signal Processors, micro-controllers...), the most natural situation is an attack starting from an empty cache. For this type of targets, power analysis is obviously preferred to timing analysis since it enables to observe the memory access sequence step by step. For non-embedded devices (e.g. for PCs), the situation is very different. Initialization attacks are obviously much more powerful but the corresponding field of applications is limited to multi-user systems. For single user-systems or in the context of remote timing attacks, the only realistic hypothesis is a loaded initial state.

The typical cache attacks depending on the type of the targeted device are summarized in Table 1.

Type of devices	Class	Cache state
embedded device	power	reset
multi-user system	memory	initialization
single user	timing	loaded

Table 1: Typical cache attacks depending on the targeted device

All previously proposed cache attacks against the AES (with a 128-bit secret key) are compared in Table 2, both in terms of efficiency and applicability. The last column in the table gives the number of information bits on the secret key recovered in the attack, as stated by the authors. Note that the assumptions on the cache initial state are not always clearly stated; our classification is derived from a careful analysis of the involved attacks. For instance, the attack described in [1] is supposed to work for any cache initial state (clear cache, loaded cache...). However, a careful examination of the program and some simulations show that its performance highly depends on the cache initial state. This influence will be discussed in Section 5.

In all these attacks, the time complexity roughly corresponds to the encryption cost of all required plaintexts, except for Bernstein's attack which performs a precomputational step with a similar time complexity.

Attack	Nature	Cache state	Complexity	Key bits
[1]	Timing	?	2^{27}	91
[14]	Power	Empty	$15 \times 2^{\lceil \log_2(\frac{S}{s_b}) \rceil}$ $(\simeq 2^9)$	$15 \times \lceil \log_2(\frac{S}{s_b}) \rceil$ (75)
[24]	Timing	Empty	2^{18}	$16 \times \lceil \log_2(\frac{S}{s_b}) \rceil$ (80)
[2]	Power	Forged	$2^{\lceil \frac{S}{s_b} \rceil} + 2^{\lceil \log_2(\frac{S}{s_b}) \rceil}$ $(\simeq 2^{32})$	$16 \times \lceil \log_2(\frac{S}{s_b}) \rceil$ (80)
[17]	Memory	Forged	2^{14}	128

Table 2: Comparison between known cache attacks against the AES (the size of each lookup table is $S = 1024$ and the results in brackets corresponds to $s_b = 32$, i.e., the block size of the L1 cache on a Pentium 3)

Initialization attacks (i.e., attacks from a forged cache state) are obviously more efficient than the other ones as shown in Table 2, but the corresponding field of applications is much more restrictive.

4 Implementation of the AES with lookup tables

The AES operates on 128-bit blocks which can be represented by a 4×4 matrix over $\mathbf{F}_{2^8}$:

$$\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}$$

The AES has been designed to be implemented on many different platforms [6]. The round function of the AES is composed of 4 basic transformations: SubBytes, ShiftRows, MixColumns, AddRoundKey that work at the byte level on the internal state. While ShiftRows and AddRoundKey are native operations even on 8-bit processors, SubBytes and MixColumns are difficult to implement. SubBytes operates on each byte of the internal state as a composition of transformations: the multiplicative inversion in $\mathbf{F}_{2^8}$ and the affine transform over $\mathbf{F}_2$ defined by:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

MixColumns considers each column as a polynomial over $\mathbf{F}_{2^8}$. This transformation corresponds to a multiplication by a fixed polynomial: $c(x) = 03\ x^3 + 01\ x^2 + 01\ x + 02$

$$\begin{pmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \\ b_{3,i} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix}$$

On 32-bit processors, lookup tables can be used to perform both operations. We can have one table of 256 words for each column of MixColumns, i.e., four tables, each of 1 KByte for the whole encryption. These tables correspond to the following relations:

$$T_0[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \bullet 03 \\ S[a] \\ S[a] \end{bmatrix} \quad T_2[a] = \begin{bmatrix} S[a] \\ S[a] \bullet 02 \\ S[a] \bullet 03 \\ S[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 02 \\ S[a] \bullet 03 \end{bmatrix}$$

wher $\bullet$ denotes the multiplication in $\mathbf{F}_{2^8}$.

Now, the j-th column of the state matrix after one AES round can be expressed as:

$$\mathrm{out}_j = T_0[\mathrm{in}_{0,j}] \oplus T_1[\mathrm{in}_{1,j-C_1}] \oplus T_2[\mathrm{in}_{2,j-C_2}] \oplus T_3[\mathrm{in}_{3,j-C_3}] \oplus RK_j$$

where the constants C_m depend on the ShiftRows transformation and RK_j is the corresponding part of the round key. An additional table of 256 words is required for the last AES round, since this last round does not involve the MixColumns operation, but only the substitution layer.

Another important point is that the first AES round depends on the xor between the secret key and the plaintext. With the previously described lookup tables, the first round of the AES can therefore be written as:

$$
\begin{aligned}
y_0 &= T_0[P_{0,0} \oplus K_{0,0}] \oplus T_1[P_{1,1} \oplus K_{1,1}] \oplus T_2[P_{2,2} \oplus K_{2,2}] \oplus T_3[P_{3,3} \oplus K_{3,3}] \oplus RK_0 \\
y_1 &= T_0[P_{0,1} \oplus K_{0,1}] \oplus T_1[P_{1,2} \oplus K_{1,2}] \oplus T_2[P_{2,3} \oplus K_{2,3}] \oplus T_3[P_{3,0} \oplus K_{3,0}] \oplus RK_1 \\
ey2 &= T_0[P_{0,2} \oplus K_{0,2}] \oplus T_1[P_{1,3} \oplus K_{1,3}] \oplus T_2[P_{2,0} \oplus K_{2,0}] \oplus T_3[P_{3,1} \oplus K_{3,1}] \oplus RK_2 \\
y_3 &= T_0[P_{0,3} \oplus K_{0,3}] \oplus T_1[P_{1,0} \oplus K_{1,0}] \oplus T_2[P_{2,1} \oplus K_{2,1}] \oplus T_3[P_{3,2} \oplus K_{3,2}] \oplus RK_3
\end{aligned}
$$

where $P_{i,j}$ are the plaintext bytes and $K_{i,j}$ are the key bytes.

5 Analysis of the AES execution on different architectures

A better understanding of the attack reported by Bernstein requires a very precise analysis of all parameters related to an AES execution. On modern processors, such an analysis can be performed with software techniques based on the performance counters. Table 3 gives the number of events that can be measured and the number of available counters (i.e., the number of events which can be measured simultaneously) for different platforms. In all following simulations, the values of these performance counters have be obtained from the `perfctr` device by the 3.2.1 papi library [4].

Processor	Number of counters	Number of events
Pentium 3	2	80
Pentium 4	18	48
Itanium	4	90
PowerPc 7450	6	200
PowerPc 750	4	40

Table 3: Characteristics of the performance counters for several architectures

All simulations have been performed with Linux (2.6.11 and 2.4.22 kernels). The compiler is *gcc* 3.4.3. The AES implementation we used is the 0.9.5 - 0.9.8 Openssl implementation [16]. We also performed some simulations with the Intel IPP 5.0 implementation, which led to very similar results. Both implementations use five lookup tables, each of 1 KByte. For instance, we are able to quantify the AES execution starting from a loaded cache initial state on a Pentium 3 by the quantities given in Table 4.

Instructions Completed	Cycles	Stall	Interrupt	L1 data accessed	L1 data misses	L2 data misses
854	525	45	0	495	20	0

L1 instruction accesses	L1 instruction misses	L2 instruction misses
450	54	0

Table 4: AES encryption on a Pentium 3 (6/8/6-10-3) from a loaded cache initial state (average over 2^{15} encryptions)

5.1 Influence of the cache initial state

The attack described in [1] is mounted in a very particular context which influences its performance. The main property is that many system calls are performed before each AES encryption because the simulations have been performed in the context of a request to a server. These system calls are of major importance since they highly influence the cache state. For instance, it is known that an interruption causes in average the replacement of 150 32-byte cache blocks in the L1 cache on a Pentium 3 processor running under Linux [22].

Moreover, Bernstein's attack performs some array manipulations before encrypting each block:

```
for (i = 0;i < 40;++i) out[i] = 0;
*(unsigned int *) (out + 32) = timestamp();
if (len < 16) return;
for (i = 0;i < 16;++i) out[i] = in[i];
for (i = 16;i < len;++i) workarea[i] = in[i];
AES_encrypt(in,workarea,&expanded);
for (i = 0;i < 16;++i) out[16 + i] = scrambledzero[i];
*(unsigned int *) (out + 36) = timestamp();
```

A consequence of these array manipulations is that some elements of the AES lookup tables are evicted from the L1 data cache. This can be checked on Table 5 which compares the number of average L1 data misses during an AES execution on a Pentium 4 when these manipulations are performed as in [1] (for `len=800`), and when they are replaced by `memcpy` instructions. It is worth noticing that, in both cases, some of these cache misses are caused by the PAPI software probe.

	Instructions Completed	Cycles	L1 Data Misses	L2 Misses
`memcpy`	824	510	65	0
`len=800`	820	534	83	0

Table 5: Average numbers of data L1 misses when the array manipulations in [1] are replaced by some `memcpy` instructions (over 2^{15} encryptions)

Therefore, it appears that the existence of systems calls and of some array manipulations in the simulations described by [1] causes the eviction from the L1 cache of many elements of the AES lookup tables before each encryption. Thus, most elements of the tables actually lie outside of the L1 cache at the beginning of each encryption. Therefore, the attack described in [1] can more or less be seen as an attack starting from an empty cache, i.e., as a reset attack, close to the attack described in [25]. This is confirmed by the simulation results given in [1, Page 10]. Indeed, we can observe that 78 key bits are recovered after 2^{25} AES encryptions on a Pentium 3. But, most of these key bits correspond to the 5 most significant bits of the key bytes, i.e., to the bits involved in cold start misses. Therefore, it seems that most of the recovered key bits in these simulations come from cold start misses.

5.2 Influence of the cache parameters

The cache parameters also influence the encryption timing. For instance, Table 6 compares the average number of cache misses during one AES encryption for a Pentium 3 and a Pentium 4. The main difference between those processors is that Pentium 3 has a 16 KByte L1 data cache and a 16 KByte L1 instruction cache, while Pentium 4 has an 8 KByte L1 data cache and a trace cache equivalent to an L1 instruction cache of 16-18 KBytes.

It clearly appears that the reduced size of the L1 cache in Pentium 4 highly increases the number of cache misses. This is due to the fact that the 5 KBytes required by the lookup tables are quite important with respect to the Pentium 4 L1 data cache size. Random accesses to these tables then create some conflict misses as explained in Section 3.3. These conflict misses do not appear anymore on the Pentium 3 since the L1 data cache is twice larger. Therefore, many cache misses can be observed during the execution of the AES on a Pentium 4 in the conditions described by Bernstein (i.e., from an almost empty cache). But, some of these cache misses are cold start misses (due to the empty cache initial state) and the other ones are conflict misses (due to the

	Instructions Completed	Cycles	Stall	L1 data accessed	L1 data misses	L2 data misses
Pentium 3	854	525	45	495	20	0
Pentium 4	824	510	12	495	65	0

Table 6: Comparison of the average numbers of cache misses between Pentium 3 and Pentium 4 (over 2^{15} encryptions), where each encryption is preceded by some `memcpy` instructions

small size of the cache). The impossibility to distinguish between both kinds of cache misses with timing information considerably reduces the performance of the attack. For this reason, the performance of the cache timing attack reported by Bernstein for a Pentium 3 cannot be achieved for a Pentium 4, as explained in [15].

6 A cache timing attack on AES starting from loaded cache

Since the existence of two different types of cache misses makes Bernstein's attack much less efficient on a Pentium 4, it clearly appears that mounting an attack in this context requires a better separation of both phenomena. This can be achieved by the following attack which avoids cold start misses since it starts from a loaded cache initial state.

This attack can be seen as a variant of Bernstein's attack in the sense that it exploits some irregularities in the timing distributions, which are typical of the micro-architecture of the targeted platform. The major difference with [1] is that our attack is a *loaded initial state attack*, i.e., the only assumption on the cache initial state is that all lookup tables lie in the cache before encryption. This situation is achieved by removing all system calls and array manipulations performed in [1].

6.1 Description of the timing attack

A timing attack can be mounted when the attacker is able to find a correlation between a certain property P on the secret key and the encryption time. In the context of cache timing attacks, the choice of Property P is related to the cache parameters. For instance, Tsunoo *et al.* in [25] consider that a long DES encryption/decryption time (i.e., a sequence of cache misses) corresponds to the fact that $K_1 \oplus K_{16} \neq E(R_0) \oplus E(R_{15})$ on some bit positions which are determined by the size of the cache blocks and the size of the used lookup tables. A similar attack on AES enables to recover the most significant bits of each key byte. For instance, for 32-byte cache blocks (e.g. on a Pentium 3), it recovers the 5 most significant bits of each byte.

For loaded cache attacks, cache misses do not occur anymore. Then, we need to find some relations between the initial state of the AES and the processor micro-architecture. Unfortunately, the whole micro-architecture of a processor is a complex system which is difficult to model. Moreover, most micro-architecture details are not documented (e.g. the structure of cache banks in the cache). Taking this fact into account, the property P that we will exploit in the attack must be identified by a precomputational step which must be performed on the same architecture as the targeted processor, as it was proposed by Bernstein. This precomputational step consists in encrypting a huge number of known plaintexts with a known key, and in characterizing the AES initial states (i.e., the XOR between the plaintext and the key) which lead to the highest encryption timings. Then, the attack consists in measuring the execution timings of different known plaintexts under the secret key and in comparing these observations and the properties learned from the precomputational step. This comparison enables to guess some key bits by comparing both results.

In the following, $A_i = K_i \oplus P_i$ denotes the i-th byte of the AES initial state. In Bernstein's attack, the precomputational step consists in determining the parameters (average and standard deviation) of the distributions of the encryption time for each value of the byte A_i. It is observed that these timing distributions may present important variations when the value of A_i

changes. Using this precomputation, the attack presented in [1] consists in computing the same parameters when some known plaintexts are encrypted with the unknown secret key. Then, the cross-correlation between both distributions may allow to recover some key bits.

In our attack, we need to modify the statistical test used in [1] since the lack of cache misses modifies the shapes of the timing distributions. Here, the precomputational step investigates the correlation between the encryption time and the values of all fixed sets of the bits of $A_i = (a_{i,0}, \dots, a_{i,7})$. For all $(a_{i,j})_{j \in J}$ with $J \subset \{0, \dots, 7\}$, we estimate the average encryption time for all $2^{|J|}$ possible values of $(a_{i,j})_{j \in J}$. Then, we deduce that, in some cases, high encryption timings always correspond to a certain value of each $a_{i,\ell}, \ell \in \{0, \dots, 7\}$, which is determined as follows.

```
$c = 0$ where $c$ is a counter which is used for determining the expected value of $a_{i,\ell}$.
For all $J \subset \{0, \dots, 7\}$ with $\ell \in J$
    Among all the $2^{|J|}$ possible values for $(a_{i,j})_{j \in J}$, determine the one for which the average
        encryption time is maximal.
    If $a_{i,\ell} = 1$ for this value of $(a_{i,j})_{j \in J}$, then increment $c$.
        otherwise, decrement $c$.
If $|c| >$ Threshold, then the expected value of $a_{i,\ell}$ is given by $sign(c) = (-1)^{a_{i,\ell}}$.
    otherwise, $a_{i,\ell}$ will be undetermined.
```

The next part of the attack, known as the non-elimination method, is described in [24]. We determine the plaintext bits which provide the highest encryption time by applying exactly the same procedure as in the precomputational step. Then, we deduce that $k_{i,\ell} = a_{i,\ell} \oplus p_{i,\ell}$ for all positions (i, ℓ) which led to a prediction during the precomputational step.

Table 7 gives the number of bits for which a value of $a_{i,\ell}$ was predicted during the precomputational step and the average error rate on the corresponding deduced key bits. For instance, with the original implementation [3] on a Pentium 3 and 2^{30} known plaintexts-ciphertexts, the attack correctly guesses 66.75 key bits in average. The main difference between the OpenSSL implementation and the original implementation by Bosselaers, Rijmen and Barreto is that the second one suffers from unaligned memory accesses as pointed out in [28]. Note that we always use the same number of encryptions both in the precomputational step and in the attack, and that the timing information corresponds to the value of the cycle counter of the processor.

Our attack is clearly different from Bernstein's attack since it does not involve the same key bits. Indeed, in the attack against the original implementation runningon a Pentium 4 (15/2/7 processor) with 2^{26} encryptions, we are able to predict 50 bits in the precomputational step. Among them, 19 (resp. 31) belong the 4 most significant bits (resp. 4 least significant bits) of the input bytes, whereas an attack based on cache misses only involve the 4 most significant bits (Pentium 4 has 64-byte cache blocks).

Implementation	Pre-computation	Number of predicted bits	Error rate
Original implementation [3]	2^{30}	75 bits	11 %
OpenSSL [16]	2^{30}	20 bits	15 %

Table 7: Result of the attack on a Pentium 3 processor

6.2 Impact of micro-architecture on cache attacks

We apply the attack on several processors of the Pentium 4 family of processors. The micro-architecture of the Pentium 4 family has evolved throught the years and we find under the name Pentium 4 some processors with completely different micro-architectures. The results given in Table 8 clearly exhibit the link between the whole micro-architecture and the efficiency of the attack.

CPU model (family/model/ stepping)	Frequency (Ghz)	predicted key bits	Error Rate	Average AES timing	Standard deviation AES timing
15/4/1	3.2	17	10 %	660	25
15/3/3	3	4	50 %	690	9
15/2/4	2.0	55	10 %	820	33
15/2/7	2.4	80	10 %	659	41
15/2/9	2.6	75	10 %	658	40
15/2/5	2.8	78	10 %	654	40

Table 8: Evaluation of the cache attack on Pentium 4 processors against the original implementation

Our observations are also confirmed by the variations of the attack performances when the compiler options are changed (Table 9).

Compiler	Processor	Option	predicted key bits	Error rate
gcc 3.2.2	Pentium 3	-	95	10 %
gcc 3.2.2	Pentium 3	-O3	80	10%
gcc 3.2.2	Pentium 3	-O9 -mcpu=pentium3 -march=pentium3	45	15%

Table 9: Evaluation of cache attacks considering different compiler options against the original implementation

7 Countermeasures

The previously described cache timing attacks against the AES can be thwarted by both following classical countermeasures.

The first one compensates the encryption timing variation. This countermeasure can be decomposed into four components:

- a cache warm-up routine which guarantees that the cache has been loaded with the tables involved in the encryption process. This element is added to remove all cold start misses and all evictions that occurs during an interruption.
- a timing probe which measures the current execution time of the AES implementation. This is achieved using the timestamp (HardClock()) of the processor.
- a compensation loop which is designed to increase the encryption time until we reach the worst case execution time (WCET). It executes several times a small piece of code with a known latency.
- an interruption or abnormal event detection: if an interrupt occurs during the encryption, some blocks of the lookup table can be evicted from the cache. This can affect the current encryption but also the following encryption. If we consider that the noise added by the interruption is too important to be exploited in an attack, we only need to reload the table before the next encryption.

Since the execution is accurately measured, this countermeasure needs to be carefully designed for each micro-architecture and operating system. This countermeasure can not defeat an adversary which is able to generate conflict misses during the encryption like in [17]. From a performance

```
if abnormal = 1 then
    Warmup()
    abnormal = 0
else
    begin = HardClock()
    AES implementation...
    end = HardClock()
    if end-begin < WCET
        Compensate()
    if end-begin > THRESHOLD
        abnormal = 1
```

Figure 6: Resistant AES implementation using compensation

point of view, the degradation is excessive since we always reach the WCET of the algorithm. As shown in Figure 3, this represents an increase by almost 100% of the average encryption cost.

Another solution is to use masking to change the table mapping. But, the main drawback of this technique is that we need to compute the permutation each time an access to a table is performed. Thus, it introduces an important overhead. The same idea can nevertheless be carried out without any important computational cost if the involved permutation is a translation. A randomly chosen mask $M = m_0 \mid m_1 \mid m_2 \mid m_3$ is applied to each word of the internal state of the AES. Then, the masked output internal state e of the AES round function is obtained from a non-masked input state a by:

$$e_j = T_0[a_{0,j} \oplus m_0] \oplus T_1[a_{1,j-C_1} \oplus m_1] \oplus T_2[a_{2,j-C_2} \oplus m_2] \oplus T_3[a_{3,j-C_3} \oplus m_3] \oplus RK_j \oplus M$$

In the previous equation, the address of each memory access is masked by a m_i. The computations of $a_{i,j} \oplus m_i$ are included into the lookup table; this means that we apply a translation on the mapping of the tables: $T'_k[x_{i,j}] = T_i[x_{i,j} \oplus m_i]$. The additional mask $RK_j \oplus M$ is considered as a mask on the round key (except for the last one): $RK'_j = RK_j \oplus M$. This operation is performed at the end of the key schedule procedure. Therefore, each AES round can be described by:

$$e_j = T'_0[a_{0,j}] \oplus T'_1[a_{1,j-C_1}] \oplus T'_2[a_{2,j-C_2}] \oplus T'_3[a_{3,j-C_3}] \oplus RK'_j$$

Then, this technique for masking the internal state of the AES allows to change the mapping of the lookup tables without any additional cost. Unfortunately, we can not change the mask for each encryption since the tables T'_k must be recomputed. This operation costs about 9000 cycles on a Pentium 4. But, the mask can be changed after 256 encryptions for instance. With this configuration the resulting overhead is only of 5 % compared to a non-masked AES implementation. Our simulations show that this countermeasure is able to thwart the previously described attack. We also believe that cache timing attacks based on cache misses, such as the one presented in [17], can also be defeated by this kind of masking.

8 Conclusions

Our study clarifies the applicability of cache attacks in the sense that it emphasizes the impacts of the cache initial state, of the whole micro-architecture of the processor, of the compiler, of the operating system... on the number of key bits which are recovered by analysing the cache behaviour. For these reasons, their performances may be hard to establish since experiments cannot always be reproduced.

References

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

[2] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. AES power attack based on induced cache miss and countermeasure. In *International Symposium on Information Technology: Coding and Computing - ITCC '05*, pages 586–591. IEEE Computer Society, 2005.

[3] Antoon Bosselaers, Vincent Rijmen, and Paulo Barreto. AES ANSI C reference code v3.0. Available for instance at http://aeslib.gcu-squad.org/.

[4] Shirley Browne, Jack Dongarra, N. Garner, Kevin S. London, and Philip Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of Supercomputing - SC 2000*. IEEE, 2000. http://icl.cs.utk.edu/papi/.

[5] David Brumley and Dan Boneh. Remote timing attack are practical. In *12th USENIX Security Symposium*, pages 1–14, 2003.

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

[7] Jean-François Dhem and Nathalie Feyt. Hardware and software symbiosis helps smart card evolution. *IEEE Micro*, 21(4):14–25, 2001.

[8] ECRYPT - European Network of Excellence in Cryptology. The side channel cryptanalysis lounge. http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html, 2005.

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

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

[11] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2/3), 2000.

[12] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology - CRYPTO '96*, Lecture Notes in Computer Science 1109, pages 104–113. Springer-Verlag, 1996.

[13] Francois Koeune and Jean-Jacques Quisquater. A timing attack against Rijndael. Technical Report CG-1999/1, UCL Crypto Group, 1999.

[14] Cédric Lauradoux. Collision attacks on processors with cache and countermeasures. In *Western European WOrkshop on Research in Cryptography - WEWoRC '05*, volume P-74 of *Lecture Notes in Informatics*, pages 76–85. Bonner Köllen Verlag, 2005.

[15] Mairéad O'Hanlon and Antony Tonge. Investigation of cache timing attacks on AES. www.computing.dcu.ie/research/papers/2005/0105.pdf, 2005.

[16] The OpenSSL project. OpenSSL-0.9.7i. http://www.openssl.org/.

[17] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[18] 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.

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

[20] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES combining side channel- and differential-attack. In *Workshop on Cryptographic Hardware and Embedded Systems - CHES '04*, Lecture Note in Computer Science 3156, pages 163–175. Springer Verlag, 2004.

[21] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A new class of collision attacks and its application to DES. In *Fast Software Encryption - FSE '03*, Lecture Note in Computer Science 2887, pages 192–205. Springer Verlag, 2003.

[22] A. Seznec and N. Sendrier. HAVEGE: a user-level software heuristic for generating empirically strong random numbers. *ACM Transactions on Modeling and Computer Simulation*, 13(4):334–346, 2003.

[23] Gurindar S. Sohi and Manoj Franklin. High-bandwidth data memory systems for superscalar processors. In *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '91*, pages 53–62. ACM Press, 1991.

[24] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES implemented on computers with cache. In *Workshop on Cryptographic Hardware and Embedded Systems - CHES '03*, Lecture Note in Computer Science 2779, pages 62–76. Springer Verlag, 2003.

[25] Yukiyasu Tsunoo, Etsuko Tsujihara, Kazuhiko Minematsu, and Hiroshi Miyauchi. Cryptanalysis of block ciphers implemented on computers with cache. In *International Symposium on Information Theory and Its Applications - ISITA '02*, pages 803–806. IEEE Information Theory Society, 2002.

[26] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture - ISCA '05*, pages 392–403. ACM Press, 1995.

[27] Kenneth M. Wilson and Kunle Olukotun. High bandwidth on-chip cache design. *IEEE Transaction Computers*, 50(4):292–307, 2001.

[28] Yamir Yunus. Optimizing performance of the AES algorithm. Technical report, Intel Corporation, 2005. `http://www.intel.com/cd/ids/developer/asmo-na/eng/dc/pentium4/knowledgebase/20250.htm`.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399