# Cache Based Remote Timing Attack on the AES

Onur Acıiçmez[1], Werner Schindler[2], and Çetin K. Koç[1,3]

[1] Oregon State University, School of EECS
Corvallis, OR 97331, USA
{aciicmez,koc}@eecs.oregonstate.edu
[2] Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185–189, 53175 Bonn, Germany
Werner.Schindler@bsi.bund.de
[3] Information Security Research Center, Istanbul Commerce University
Eminönü, Istanbul 34112, Turkey
koc@cryptocode.net

**Abstract.** We introduce a new robust cache-based timing attack on AES. We present experiments and concrete evidence that our attack can be used to obtain secret keys of remote cryptosystems if the server under attack runs on a multitasking or simultaneous multithreading system with a large enough workload. This is an important difference to recent cache-based timing attacks as these attacks either did not provide any supporting experimental results indicating if they can be applied remotely, or they are not realistically remote attacks.

**Keywords:** Cache Attack, Remote Attack, AES, Timing Analysis, Side Channel Analysis.

## 1 Introduction

The implementations of cryptosystems may leak information through so-called side channels due to the physical requirements of the device, e.g., power consumption, electromagnetic emanation and/or execution time. In side-channel attacks, the information obtained from one or more side-channels is used to reveal the key of a cryptosystem. Power, electromagnetic, and timing attacks are well-known types of side-channel attacks (c.f. [15,13,16,4,25]). Side-channel analysis of computer systems has recently attracted increasing attention (for a discussion see [3]). In this paper, we focus on a type of side channel cryptanalysis that takes advantage of the information leaks through the cache architecture of a CPU.

The feasibility of the cache based side channel attacks, abbreviated to "cache attacks" from here on, was first mentioned by Kocher and then by Kelsey et al. in [15,14]. D. Page described and simulated a theoretical cache attack on DES [21]. Cache attacks were first implemented by Tsunoo et al. [27,26]. They developed different attacks on various ciphers, including MISTY1 [26], DES and Triple-DES [27]. The recent efforts have unleashed the actual power of cache attacks [2,5,24,20,6,17,19,28,8].

M. Abe (Ed.): CT-RSA, LNCS 4377, pp. 271–286, 2007.
© Springer-Verlag Berlin Heidelberg 2007

None of the mentioned papers, except [5], considered whether a remote cache attack is feasible. Although Bernstein claimed that his attack could reveal a full AES key remotely, his experiments were purely local [5] and he did not present sufficient evidence to support those claims. Furthermore, a thorough analysis of this attack showed that it could not compromise remote systems and could only recover the key partially in a local attack [18]

Despite of [7] the vulnerability of software systems against remote timing attacks was not taken into account until Brumley and Boneh performed an attack on unprotected SSL implementations over a local network ([10]). An improved version of this attack can be found in [1].

In this paper, we present a robust effective cache attack, which can be used to compromise remote systems, on the AES implementation described in [11] for 32-bit architectures. Although our basic principles can be used to develop similar attacks on other implementations, we will only focus on the particular implementation stated above.

Our paper is organized as follows: we will cover the basics of cache attacks in the next section. In Section 3, we will introduce our new cache attack on the AES. The results of the experiments will be presented along with the implementation details in Section 4. The paper ends with concluding remarks.

## 2 Basics of a Cache Attack

A cache is a small and fast storage area used by the CPU to reduce the average time to access main memory. It stores copies of the most frequently used data.[1] When the processor needs to read a location in main memory, it first checks to see if the data is already in the cache. If the data is already in the cache (a cache hit), the processor immediately uses this data instead of accessing the main memory, which has a longer latency than a cache. Otherwise (a cache miss), the data is read from the memory and a copy of it is stored in the cache. The minimum amount of data that can be read from the main memory into the cache at once is called a cache line or a cache block, i.e., each cache miss causes a cache block to be retrieved from a higher level memory.

Cache attacks exploit the cache hits and misses that occur during the encryption / decryption process of the cryptosystem. Even if the same instructions are executed for all (plaintext, cipherkey) pairs the cache behavior during the execution may cause variations in the program execution time and power consumption. Cache attacks try to exploit such variations to narrow the exhaustive search space of secret keys.

Theoretical cache attacks were first described by Page in [21]. Page characterized two types of cache attacks, namely trace-driven and time-driven. In trace-driven attacks (e.g. [2,6,17]), the adversary is able to obtain a profile of the cache activity of the cipher. This profile includes the outcomes of every memory access the cipher issues in terms of cache hits and misses. Therefore,

[1] Although it depends on the particular data replacement algorithm, this assumption is true almost all the time for current processors.

the adversary has the ability to observe (e.g.) if the $2^{nd}$ access to a lookup table yields a hit and can infer information about the lookup indices, which are key dependent. This ability gives an adversary the opportunity to make inferences about the secret key.

Time driven attacks, on the other hand, are less restrictive since they do not rely on the ability of capturing the outcomes of individual memory accesses [5,27,8]. The adversary is assumed to be able to observe the total execution time of the cipher, i.e. the aggregate profile, which at most gives hint to an approximative number of cache hits and misses, or to input-dependent correlations of particular operations. Time-driven attacks are based on statistical inferences, and therefore require much higher number of samples than trace-driven attacks.

We have recently seen another type of cache attacks that can be named as "access-driven" attacks [20,24,19]. In these attacks, the adversary can determine the cache sets that the cipher process modifies. Therefore, she can understand which elements of the lookup tables or S-boxes are accessed by the cipher. Then, the candidate keys that cause an access to unaccessed parts of the tables can be eliminated.

## 3 A New Remote Cache Attack on AES

All of the proposed cache attacks, except [5], either assume that the cache does not contain any data related to the encryption process prior to each encryption or explicitly force the cache architecture to replace some of the cipher data. The implementations of Tsunoo et al. accomplish the so-called 'cache cleaning' by loading some garbage data into the cache to clean it before each encryption [27,26]. The need of cleaning the cache makes an attack impossible to reveal information about the cryptosystems on remote machines, because the attacker must have an access to the computer to perform cache cleaning. They did not investigate if this attack could successfully recover the key without employing explicit cache cleaning on certain platforms.

Attacks described in [20] replace the cipher data on the cache with some garbage data by loading the content of a local array into the cache. Again, the attacker needs an access to the target platform to perform these attacks. Therefore, none of the mentioned studies could be considered as practical for remote attacks over a network, unless the attacker is able to manipulate the cache remotely.

In this paper, we show that it is possible to apply a cache attack without employing cache cleaning or explicitly aimed cache manipulations when the cipher under the attack is running on a multitasking system, especially on a busy server. In our experiments we run a dummy process simultaneously with the cipher process. Our dummy process randomly issues memory accesses and eventually causes the eviction of AES data from the cache. This should not be considered as a form of intentional cache cleaning, because we use this dummy process only to imitate a moderate workload on the server. In presence of different processes

that run on the same machine with the cipher process, the memory accesses that are issued by these processes automatically evict the AES data, i.e., cause the same effect of our dummy process on the execution of the cipher.

Multitasking operating systems allow the execution of multiple processes on the same computer, concurrently. In other words, each process is given permission to use the resources of the computer, not only the processor but also the cache and other resources. Although it depends on the cache architecture and the replacement policy, we can roughly say that the cache contains most recently used data almost all the time. If an encryption process stalls for enough time, the cipher data will completely be removed from the cache, in case of the presence of other processes on the machine. In a simultaneous multithreading system, the encryption process does not even have to stall. The data of the process, especially parts of large tables, is replaced by other processes' data on-the-fly, if there is enough workload on the system.

The results of our experiments show that the attack can work in such a case on a simultaneous multithreading environment. The reader should note that our results also point the vulnerability of remote systems against Tsunoo's attack on DES, as well.

In this section we outline an example cache attack on AES with a key size of 128 bits. In our experiments we consider the 128-bit AES version with a block length of 128 bits. Our attack can be adjusted to AES with key length 192 or 256 in a straight-forward manner (cf. Subsect. 3.4).

The basic attack consists of two different stages, considering table-lookups from the first and second round, respectively. The basic attack may be considered as an adaption of the ideas from the earlier cache attack works to a timing attack on AES since similar equations are used. Our improved attack variant is a completely novel approach. It employs a different decision strategy than the basic one and is much more efficient. It does not have different parts and falls into sixteen independent 8-bit guessing problems.

The differences of our approaches from the earlier works are the followings. First of all, we exploit the internal collisions, i.e., the collisions between different table lookups of the cipher. Some of the earlier works (e.g. [20,19,24,6]) exploits the cache collisions between the memory accesses of the cipher and another process. Exploiting such external collisions mandates the use of explicit local cache manipulations by (e.g.) having access to the target machine and reading a local data structure. This necessity makes these attacks unable to compromise remote systems. On the other hand, taking advantage of internal collisions removes this necessity and enables one to devise remote attacks as will be shown in this paper. The idea of using internal collisions is employed in some of the previous works, e.g. in [26,27,17]. The earlier timing attacks that rely on internal collisions perform the so-called cache cleaning, which is also a form of explicit local cache manipulations. These works did not realize the possibility of automatic cache evictions due to the workload on the system, and therefore could not show the feasibility of remote attacks.

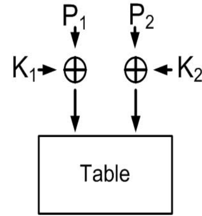


**Fig. 1.** Two different accesses to the same table

### 3.1 Basic Attack Model

We will use Figure 1 to explain the basic attack model. Assume there are two accesses to the same table. Let $P_i$ and $K_i$ be the $i^{th}$ byte of the plaintext and cipherkey, respectively. In this paper, each byte is considered to be either an 8-digit radix-2 number $\in \{0,1\}^8$, that can be added in $GF(2^8)$ using a bitwise exclusive-or operation, or an integer in [0, 255] that can be used as an index. For the rest of this section, we assume that each plaintext consists of a single 16-byte message block.

The structure shown in the figure uses different bytes of the plaintext and the cipherkey as inputs to the function that computes the index of each of these two accesses. If both of them access to the same element of the table the latter should find the target data in the cache, resulting a cache hit; which should reduce the execution time. Then the key byte difference $K_1 \oplus K_2$ follows immediately from the values of plaintext bytes $P_1$ and $P_2$ using the equation

$$P_1 \oplus K_1 = P_2 \oplus K_2 \Rightarrow P_1 \oplus P_2 = K_1 \oplus K_2 \ .$$

In trace-driven attacks, we assume that the adversary can directly understand if the latter access results a hit, thus can directly obtain $K_1 \oplus K_2$. This goal is more complicated in time-driven attacks. We need to use a large sample to realize an accurate statistics of the execution. When sampling different plaintext pairs with the corresponding execution time we would expect that the plaintext byte difference $P_1 \oplus P_2$ that causes the shortest average execution time gives the correct key byte difference, assuming a cache hit decreases the overall execution time.

However, in a real environment the situation is more complicated. Even if the latter access is to a different element other than the target of the former access, a cache hit may still occur. Any cache miss results the transfer of an entire cache line, not only one element, from the main memory. Therefore, if the former access has a target, which lies in the same cache line of the previously accessed data, a cache hit will occur. In that case, we can still obtain the key byte difference partially as follows:

Let $\delta$ be the number of bytes in a cache line and assume that each element of the table is $k$ bytes long. Under this situation, there are $\delta/k$ elements in each line, which means any access to a specific element will map to the same line with

$(\delta/k-1)$ different other accesses. If two different accesses to the same array read the same cache line, the most significant parts of their indices, i.e., all of the bits except the last $\ell = \log_2(\delta/k)$ bits, must be identical.[2] Using this fact, we can find the difference of the most significant part of the key bytes using the equation

$$\langle P_1 \rangle \oplus \langle P_2 \rangle = \langle K_1 \rangle \oplus \langle K_2 \rangle \ ,$$

where $\langle A \rangle$ stands for the most significant $(8-\ell)$ bits of $A$.

Indices of table lookups are driven by the outputs of usually more complex functions of the plaintext and the cipherkey than only bitwise exclusive-or of their certain bytes. The structure of these functions determines the performance of the attack, i.e., the amount of reduction in the exhaustive search space. The basic idea presented above can be adapted to any such function in order to develop successful attacks.

The attack model discussed so far is partially correct, except the lack of counting the fact that two different accesses to the same cache line may even increase the overall execution time. We realized during our experimentation that an internal collision, i.e. cache hit, at a particular AES access either shortens or lenghtens the overall execution time. The latter phenomenon may occur if a cache hit occurs from a logical point of view but the respective cache line has not already been loaded, inducing double work. Thus, if we gather a sample of messages and each of these messages generates a cache hit during the same access, then the execution time distribution of this sample will be significantly different than that of a random sample. We consider this fact to develop our attacks on the AES.

### 3.2 First Round Attack

The implementation we analyze is described in [11] and it is widely used on 32-bit architectures. To speed up encrytion all of the component functions of AES, except AddRoundKey, are combined into lookup tables and the rounds turn to be composed of table lookups and bitwise exclusive-or operations. The five lookup tables T0, T1, T2, T3, T4 employed in this implementation are generated from the actual AES S-box value as the following way:

$$T0[x] = (2 \bullet s(x), s(x), s(x), 3 \bullet s(x)), \quad T1[x] = (3 \bullet s(x), 2 \bullet s(x), s(x), s(x)),$$

$$T2[x] = (s(x), 3 \bullet s(x), 2 \bullet s(x), s(x)), \quad T3[x] = (s(x), s(x), 3 \bullet s(x), 2 \bullet s(x)),$$

$$T4[x] = (s(x), s(x), s(x), s(x)) \ ,$$

where s(x) and $\bullet$ stand for the result of an AES S-box lookup for the input value $x$ and the finite field multiplication in $GF(2^8)$ as it is realized in AES, respectively. The round computations, except in the last round, are in the form:

$$(S^{(r+1)}_{(4*i)},\, S^{(r+1)}_{(4*i+1)},\, S^{(r+1)}_{(4*i+2)},\, S^{(r+1)}_{(4*i+3)}) := (RK^r_{(4*i)},\, RK^r_{(4*i+1)},\, RK^r_{(4*i+2)},\, RK^r_{(4*i+3)}) \oplus$$

[2] We assume that lookup tables are aligned in the memory, which is the case most of the time. If they are not aligned, this will indeed increase the performance of the attack as mentioned in [20].

$$T0[S^r_{(4*i)}] \oplus T1[S^r_{(4*i+5 \ mod \ 16)}] \oplus T2[S^r_{(4*i+10 \ mod \ 16)}] \oplus T3[S^r_{(4*i+15 \ mod \ 16)}] ,$$

where $S^r_i$ is the byte $i$ of intermediate state value that becomes the input of round $r$, $RK^r_i$ is the $i^{th}$ byte of the $r^{th}$ round key and $i \in \{0, .., 3\}$.

The first 4 references to the first table, T0, are:

$$P_0 \oplus K_0, P_4 \oplus K_4, P_8 \oplus K_8, P_{12} \oplus K_{12} .$$

If any two of these four references are forced to map to the same cache line for a sample of plaintext then we know that this will affect the average execution time. For example, if we assign the value $\langle P_0 \oplus K_0 \oplus K_4 \rangle$ to $\langle P_4 \rangle$, i.e.,

$$\langle P_4 \rangle = \langle P_0 \oplus K_0 \oplus K_4 \rangle$$

for a large sample of plaintexts the timing characteristics of this sample will be different than that of a randomly chosen sample. We can use this fact to guess the correct key byte difference $\langle K_0 \oplus K_4 \rangle$.

Using the same idea we can find all key byte differences $\epsilon_{i,j} = \langle K_i \oplus K_j \rangle$ with $i, j \in \{0, 4, 8, 12\}$. For properly selected indices $(i_1, j_1), (i_2, j_2), (i_3, j_3)$, i.e. if the $GF(2)$-linear span of $\{K_{i_1} \oplus K_{j_1}, K_{i_2} \oplus K_{j_2}, K_{i_3} \oplus K_{j_3}\}$ contains all six XOR sums $K_0 \oplus K_4, K_0 \oplus K_8, \ldots, K_8 \oplus K_{12}$, then each $\epsilon_{i,j}$ follows immediately from $\epsilon_{i_1,j_1}, \epsilon_{i_2,j_2}$ and $\epsilon_{i_3,j_3}$. We can further reduce the search space by considering the accesses to other three tables T1, T2 and T3. In general, we can obtain $\langle K_i \oplus K_{4*j+i} \rangle$ for $i, j \in \{0, 1, 2, 3\}$. Since $(8 - \ell)$ is the size of the most significant part of a table entry in terms of the number of bits the first round attack allows us to reduce the search space by $12 * (8 - \ell)$ bits. The parameter $\ell$ depends on the cache architecture. For $\ell = 0$, which constitutes the theoretical lower bound, the search space for a 128 bit key becomes only 32 bits. For $\ell = 4$ the search space is reduced by 48 bits yielding an 80 bit problem.

On widely used processors the search space typically reduces to 56, 68, or 80 bits for 128-bit keys. In the environment where we performed our experiments the cache line size of the L1 cache is 64 bytes, i.e. the most significant part of a key byte difference is 4 bits long. In other words, we can only obtain the first 4 bits of $K_i \oplus K_{4*j+i}$ and the remaining 4 bits have to be searched exhaustively unless we use a second round attack.

### 3.3 Second Round Attack – Basic Variant

Using the guesses from the first round a similar guessing procedure can be applied in the second round to obtain the remaining key bits. We briefly explain an approach that exploits only accesses to $T_0$, i.e., the first table. To simplify notation we set $\Delta_i := P_i \oplus K_i$ in the remainder of this section. In the second round the encryption accesses four times to T0, namely to obtain the values

$$2 \bullet s(\Delta_8) \oplus 3 \bullet s(\Delta_{13}) \oplus s(\Delta_2) \oplus s(\Delta_7) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus K_8 \oplus 0x01 \quad (1)$$

$$2 \bullet s(\Delta_0) \oplus 3 \bullet s(\Delta_5) \oplus s(\Delta_{10}) \oplus s(\Delta_{15}) \oplus s(K_{13}) \oplus K_0 \oplus 0x01 \quad (2)$$

$$2 \bullet s(\Delta_4) \oplus 3 \bullet s(\Delta_9) \oplus s(\Delta_{14}) \oplus s(\Delta_3) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus 0x01 \quad (3)$$

$$2 \bullet s(\Delta_{12}) \oplus 3 \bullet s(\Delta_1) \oplus s(\Delta_6) \oplus s(\Delta_{11}) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus K_8 \oplus K_{12} \oplus 0x01 \quad (4)$$

where s(x) and $\bullet$ stand for the result of an AES S-box lookup for the input value $x$ and the finite field multiplication in $GF(2^8)$ as it is realized in AES, respectively. If the first access $(P_0 \oplus K_0)$ touches the same cache line as (1) for each plaintext within a sample, i.e. if

$$\langle P_0 \rangle = \langle 2 \bullet s(\Delta_8) \oplus 3 \bullet s(\Delta_{13}) \oplus s(\Delta_2) \oplus s(\Delta_7) \oplus s(K_{13}) \oplus K_4 \oplus K_8 \oplus 0x01 \rangle \quad (5)$$

the expected average execution time will be different than for a randomly chosen sample. If we assume that the value $\langle K_4 \oplus K_8 \rangle$ has correctly been guessed within the first round attack this suggests the following procedure.

1. Phase: Obtain a sample of $N$ many (plaintext, execution time) pairs.
2. Phase: Divide the entity of all (plaintext, execution time) pairs into $2^{32}$ (overlapping) subsets, one set for each candidate $(\widetilde{K_2}, \widetilde{K_7}, \widetilde{K_8}, \widetilde{K_{13}})$ value. Put each plaintext into all sets that correspond to candidates $(\widetilde{K_2}, \widetilde{K_7}, \widetilde{K_8}, \widetilde{K_{13}})$ that satisfy the above equation. Note that a particular plaintext should be contained in about $N/2^{8-\ell}$ different subsets.
3. Phase: Calculate the timing characteristics of each set, i.e., the average execution time in our case. Compute the absolute difference between each average and the average execution time of the entire sample. There will be a total of $2^{4 \cdot 8}$ timing differences, each from a different absolute value of $(\widetilde{K_2}, \widetilde{K_7}, \widetilde{K_8}, \widetilde{K_{13}})$. The set with the largest difference should point to the correct values for these 4 bytes.

Hence, we can search through all candidates for $(K_2, K_7, K_8, K_{13}) \in GF(2)^{32}$ to guess the true values. Applying the same idea to (2) to (4) we can recover the full AES key. Note that in each of the consecutive steps only $4 \cdot 4 = 16$ key bits have to be guessed since $K_i$ and the most significant bits from some other $K_j$ follow from the first step and $\epsilon_{ij}$ from the first round attack (cf. Sect. 3.2) where $i$ is a suitable index in $\{2, 7, 8, 13\}$.

The bottleneck is clearly the first step since one has to distinguish between $2^{32}$ key hypotheses rather than between $2^{16}$. Experimental results are given in Sect 4. In the next subsection we introduce a more efficient variant that saves both samples and computations.

### 3.4 A More Efficient, Universally Applicable Attack

In the previous subsection we explained a second round attack where 32, resp. 16, key bits have to be guessed simultaneously. In this section we introduce another approach that allows independent search for single key bytes. It is universally applicable in the sense that it could also be applied in any subsequent round, e.g. to attack AES with 256 bit keys.

We explain our idea at (1). Our goal is to guess key byte $K_8$. Recall that access to the same cache line as for $(P_0 \oplus K_0)$ is required in the second round iff (5) holds. If we fix the four plaintext bytes $P_0$, $P_2$, $P_7$, and $P_{13}$ then (5) simplifies to

$$\langle c \rangle = \langle 2 \bullet s(\Delta_8) \rangle \quad (6)$$

with an unknown constant $c = c(K_0, K_2, K_4, K_7, K_8, K_{13}, P_0, P_2, P_7, P_{13})$. We observe encryptions with randomly selected plaintext bytes $P_i$ for $i \notin \{0, 2, 7, 13\}$ and evaluate the timing characeristics with regard to all 256 possible values of $P_8$. For the most relevant case, i.e. $\ell = 4$, there are 16 plaintext bytes ($2^\ell$ in the general case) that yield the correct (but unknown) value $< 2 \bullet s(\Delta_8) >$ that meets (5). Ideally, with regard to the timing characteristics, these 16 plaintext bytes should be ranked first, pointing at the true subkey $K_8$; i.e. to a key byte that gives identical right-hand sides $< 2 \bullet s(\Delta_8) >$ for all these 16 plaintext bytes. The ranking is done similar as in Subsect. 3.2. To rank the 256 $P_8$-bytes one calculates for each subset with equal $P_8$ values the absolute difference of its average execution time with the average execution time of all samples. The set with the highest difference is ranked first and so on. In a practical attack our decision rule says that we decide for that key byte candidate $\widetilde{K}_8$ for which a maximum number of the $t$ (e.g. $t = 16$) top-ranked plaintext bytes yield identical $\langle 2 \bullet s(\Delta_8)\rangle$ values. If the decision rule does not clearly point to one subkey candidate, we may perform the same attack with a second plaintext $P_0'$ for which $\langle P_0\rangle \neq \langle P_0'\rangle$ while we keep $P_2, P_7, P_{13}$ fixed (changing $\langle c\rangle$ to $\langle c'\rangle := \langle c\rangle \oplus \langle P_0 \oplus P_0'\rangle$). Applying the same decision rule as above, we obtain a second ranking of the subkey candidates.

Clearly, if $P_8$ and $P_8'$ meet (6) for $P_0$ and $P_0'$, resp., then

$$\langle P_0 \oplus P_0'\rangle = \langle 2 \bullet s(P_8 \oplus K_8)\rangle \oplus \langle 2 \bullet s(P_8' \oplus K_8)\rangle. \tag{7}$$

Equation (7) may be used as a control equation for probable subkey candidates $\widetilde{K}_8$. From the ranking of $\widetilde{P}_8$ and $\widetilde{P}_8'$, we derive an order for the pairs $(\widetilde{P}_8, \widetilde{P}_8')$, e.g. by adding the ranks of the components or their absolute distances from the respective means. For highly ranked pairs $(\widetilde{P}_8, \widetilde{P}_8')$ we substitute $(\widetilde{P}_8, \widetilde{P}_8', \widetilde{k})$ into control equation (7) where $\widetilde{k}$ is a probable subkey candidate from the 'elementary' attacks.

We note that the attack described above can be applied to exploit the relation between any two table-lookups. By reordering a type (5)-equation one obtains an equation of type (6) whose right-hand side depends only on one key byte (to be guessed) and one plaintext byte. The plaintext bytes that affect the left-hand side are kept constant during the attack. The whole key could be recovered by 16 independent one-key byte guessing problems. We mention that the (less costly) basic first round attacks might be used to check the guessed subkey candidates $\widetilde{K}_0, \ldots, \widetilde{K}_{15}$.

**Comparison with the Basic Second Round Attack from Subsect 3.3.** For sample size $N$ the 'bottleneck' of the basic second round attack, the 32 bit guessing step, requires the computation of the average execution times of $2^{32}$ sample subsets of size $\approx N/2^{8-\ell}$. In contrast, each of the 16 runs of the improved attack variant only requires the computation of the average execution times of 256 subsets of size $\approx N_I/256$ (with $N_I$ denoting the sample size for an individual guessing problem) and sorting two lists with 256 elements (plaintexts and key byte candidates). Even more important, $16N_I$ will turn out to be clearly smaller than $N$ (cf. Sect. 4).

The only drawback of the improved variant is that it is a chosen-input attack, i.e., it requires an active role of the adversary. In contrast, the basic variant explained in the previous section is principally a known-plaintext attack, which means an adversary does not have to actively interfere with the execution of the encryption process, i.e., the attack can be carried out by monitoring the traffic of the encryption process. However, this is only true for the (less important) so-called innerprocess attacks (cf. Section 4 for details). For 'real' attacks (interprocess and remote attacks) the basic variant is performed as a chosen-input attack, too, since the attacker needs to choose the plaintext to be encrypted as the concatenation of several identical 128 bit strings in order to increase the signal-to-noise ratio.

## 4 Experimental Details and Results

We performed two types of experimental attacks that we call innerprocess and interprocess attacks to test the validity of our attack variants. In innerprocess attack we generated a random single-block of messages and measured their encryption times under the same key. The encryption was just a function that is called by the application to process a message. The execution time of the cryptosystem was obtained by calculating the difference of the time just before the function call and immediately after the function return. Therefore, there was minimum noise and the execution time was measured almost exactly.

For the second part of the experiments, i.e., interprocess attack, we implemented a simple TCP server and a client program that exchange ASCII strings during the attack. The server reads the queries sent by the client, and sends a response after encrypting each of them. The client measures the time between sending a message and receiving the reply. These measurements were used to guess the secret key. The server and client applications run on the same machine in this attack. There was no transmission delay in the time measurements but network stack delays were present.

Brumley and Boneh pointed out that a real remote attack over a network was principally able to break a remote cipher, when the interprocess version of the same attack worked successfully. Furthermore, their experiments also showed that their actual remote attack required roughly the same number of samples used in the interprocess version [10]. Therefore, we only performed interprocess experiments. Applying an interprocess attack successfully is a sufficient evidence to claim the actual remote version would also work with (most likely) a larger sample size.

We performed our attack against OpenSSL version 0.9.7e. All of the experiments were run on a 3.06 GHz. HT-enabled Xeon machine with a Linux operating system. The source code was compiled using the gcc compiler version 3.2.3 with default options. We used random plaintexts generated by rand() and srand() functions available in the standard C library. The current time is fed into srand() function, serving as seed for the pseudorandom number generator. We measured time in terms of clock cycles using the cycle counter.

For the experiments of innerprocess attack, we loaded 8 KB garbage data into the L1 cache before each encryption to remove all AES data from the first level

cache. We did not employ this type of cache cleaning during the experiments of interprocess attack. Instead, we wrote a simple dummy program that randomly accesses an 8 KB array and run this program simultaneously with the server in order to imitate the effect of a workload on the computer.

We used two parameters in our experiments.

1. Sample Size ($N$): This is the number of different (plaintext, execution time) pairs collected during the first phase of the attacks. We have to use a large enough sample of queries to obtain accurate statistical characteristic of the system. However, a very large sample size causes unnecessary increase in the cost of the attack.
2. Message Length ($L$): This is the number of message blocks in each query. We concatenated a single random block $L$ many times with one another to form the actual query. $L$ was 1 during the innerprocess attack, i.e., each query was a single block, whereas it was 1024 in the interprocess attack. This parameter is used to increase the signal-to-noise ratio in the case of having network delays in the measurements.

We performed our attacks on the variant of AES that has 128-bit key and block sizes. The cache line size of L1 cache is 64 bytes, which makes $\ell = 4$ bits. The cipher was run on ECB mode. In our experiments, we performed all second round guessing problems for the basic attack with only $2^{12}$ different key hypotheses, one of them being the correct key combination. Our intention was to demonstrate the general principle but to save many encryptions. In this way, we reduced the complexity of 'bottleneck' exhaustive search by even more than a factor of $2^{20}$ since less samples are sufficient for the reduced search space.

For the innerprocess attack, collecting $2^{18}$ samples was enough to find the correct value of the key. Since we only considered $2^{12}$ different key hypotheses in second round guessing problems, the required sample size would be more than $2^{18}$ for a real scale innerprocess attack. In fact, statistical calculations suggest that $4*2^{18}$ samples should be sufficient for $2^{32}$ key hypotheses although in a strict sense (13) only guarantees an error probability of at most $2\epsilon/(1-c)-\epsilon^2/(1-c)^2 > 2\epsilon - \epsilon^2$ (cf. Example 1 in the appendix). (The right-hand side denotes the error probability for the reduced search space while $c$ is unknown.) However, (11) is a (pessimistic) lower bound we may expect that the true error probability is indeed significantly smaller, possibly after increasing the sample size somewhat.

The key experiment is the interprocess attack, which shows the vulnerability of remote servers against such cache attacks. In our experiments, we collected 50 million random but known samples and applied our attack on this sample set. This sample size was clearly sufficient to reveal the correct key value among $2^{12}$ different key hypotheses. Again, the same heuristic arguments indicate the sufficiency of 200 million samples in a real-scale attack. We also estimated the number of required samples in a remote attack over a local network. Rough statistical considerations indicate that increasing the sample size of the interprocess attack by a factor of less than 6 should be sufficient to successfully apply the attack on a remote server.

We tested our improved variant on the same platform with the same settings. The only difference was the set of the plaintexts sent to the server. We only performed interprocess attack with this new decision strategy. Our experimental results indicate a clear improvement over the basic attack. We could recover a full 128-bit AES key by encrypting slightly more than 6.5 million samples in average per each of the 16 guessing problems and a total of 106 million queries, each containing $L = 1024$ message blocks. Recall the further advantage of the improved variant, namely the much lower analysis costs.

We want to mention that all of these results correspond to the minimum number of samples from which we got the correct decision from our decision strategy. In a real-life-attack an adversary clearly has to collect more samples to be confident on her decisions in a real attack. More sophisticated stochastic models that are tailored to specific cache strategies certainly will improve the efficiency of our attack.

Our client-server model does not perfectly fit into the behavior of an actual security application. In reality, encrypting/decrypting parties do not send responses immediately and perform extra operations, besides encryption and decryption. However, this fact does not nullify our client-server model. Although, the less signal-to-noise ratio in actual attacks increases the cost, it does not change the principle feasibility of our attacks. We want to mention that timing variations caused by extra operations decrease the signal-to-noise ratio. If a security application performs the same operations for each processed message, we expect the "extra timing variations" to be minimal, in which case the decrease in the signal-to-noise ratio and thus the increase in the cost of the attack also remains small.

## 5 Conclusion

We have presented a new cache-based timing attack on AES software implementations. Our experiments indicate that cache attacks can be used to extract secret keys of remote systems if the system under attack runs on a server with a multitasking or multithreading system and a large enough workload. Although a large number of measurements are required to successfully perform a remote cache attack, it is feasible in principle. In this regard, we would like to point the feasibility of such cache attacks to the public, and recommend implementing appropriate countermeasures. Several countermeasures [21,5,20,22,23,9] have been proposed to prevent possible vulnerabilities and develop more secure systems.

## References

1. O. Acıiçmez, W. Schindler, Ç. K. Koç. Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations. ACM CCS'05, C. Meadows, P. Syverson, editors, 139-146, Virginia, 2005.
2. O. Acıiçmez and Ç. K. Koç. Trace-Driven Cache Attacks on AES. Cryptology ePrint Archive, Report 2006/138, 2006. Available at: http://eprint.iacr.org/2006/138

3. O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. Predicting Secret Keys via Branch Prediction. Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, to appear.
4. D. Agrawal, B. Archambeault, J. R. Rao, P. Rohatgi. The EM Side-Channel(s). CHES'02, B. S. Kaliski, Ç. K. Koç and C. Paar, editors, 29-45, Springer, LNCS 2523, Berlin 2003.
5. D. J. Bernstein. Cache-timing attacks on AES. April, 2005. Available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf
6. G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. ITCC'05, 586 - 591, IEEE Computer Society, 2005.
7. D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. CRYPTO 98, H. Krawczyk, editor, 1-12, Springer, LNCS 1462, Berlin 1998.
8. J. Bonneau and I. Mironov. Cache-Collision Timing Attacks against AES. CHES'06, Springer, LNCS , Berlin 2006.
9. E. Brickell, G. Graunke, M. Neve, J.-P. Seifert. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. Cryptology ePrint Archive, Report 2006/052, 2006. Available at: http://eprint.iacr.org/2006/052
10. D. Brumley, D. Boneh. Remote Timing Attacks are Practical. Proceedings of the $12^{th}$ Usenix Security Symposium, 1-14, 2003.
11. J. Daemen, V. Rijmen. "The Design of Rijndael". Springer, Berlin 2001.
12. W. Feller. An Introduction to Probability Theory. $3^{rd}$ edition, revised printing, Wiley, New York 1970.
13. K. Gandolfi, C. Mourtel, F. Olivier. Electromagnetic Analysis: Concrete Results. CHES'01, Ç. K. Koç, D. Naccache, and C. Paar, editors, 251-261, Springer, LNCS 2162, Berlin 2001.
14. J. Kelsey, B. Schneier, D. Wagner, C. Hall. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, vol.8, 141-158, 2000.
15. P. C. Kocher. Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems. CRYPTO '96, N. Koblitz, editor, 104-113, Springer, LNCS 1109, Berlin 1996.
16. P. C. Kocher, J. Jaffe, B. Jun. Differential Power Analysis. CRYPTO '99, M. Wiener, editor, 388-397, Springer, LNCS 1666, Berlin 1999.
17. C. Lauradoux. Collision attacks on processors with cache and countermeasures. WEWoRC'05, C. Wolf, S. Lucks, and P.-W. Yau, editors, 76-85, KÏlen, LNI P-74, Bonn 2005.
18. M. Neve, J.-P. Seifert, Z. Wang. A refined look at Bernstein's AES side-channel analysis. ASIA CCS'06, 369-369, ACM Press, 2006.
19. M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. SAC'06, E. Biham, A. Youssef, editors, to appear.
20. D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. CT-RSA'06, D. Pointcheval, editor, 1-20, Springer, LNCS 3860, Berlin 2006.
21. 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.
22. D. Page. Defending Against Cache Based Side-Channel Attacks. Technical Report. Department of Computer Science, University of Bristol, 2003.

23. D. Page. Partitioned Cache Architecture as a Side Channel Defence Mechanism. Cryptography ePrint Archive, Report 2005/280, 2005. Available at: http://eprint.iacr.org/2005/280
24. C. Percival. Cache missing for fun and profit. BSDCan'05, Ottawa, 2005. Available at: http://www.daemonology.net/hyperthreading-considered-harmful/
25. W. Schindler: On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods, PKC '05, S. Vaudenay, editor, 85–103, Springer, LNCS 3386, Berlin 2005.
26. Y. Tsunoo, E. Tsujihara, K. Minematsu, H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. ISITA'02, 803-806, IEEE Information Theory Society, 2002.
27. Y. Tsunoo, T.Saito, T. Suzaki, M. Shigeri, H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. CHES'03, C. D. Walter, Ç. K. Koç, and C. Paar, editors, 62-76, Springer, LNCS 2779, Berlin 2003.
28. Y. Tsunoo, E. Tsujihara, M. Shigeri, H. Kubo, K. Minematsu. Improving cache attacks by considering cipher structure. International Journal of Information Security, February 2006.


## Appendix: Scaling the Sample Size $N$

In order to save measurements we performed our practical experiments to the basic second round attack from Subsect. 3.3 with a reduced key space. Clearly, to maintain the success probability for the full subkey space the sample size $N$ must be increased to $N'$ since the adversary has to distinguish between more admissible alternatives. In this section we estimate the ratio $r := N'/N$.

We interpret the measured average execution times for the particular subkey candidates as realizations of normally (Gaussian) distributed random variables, denoted by $Y$ (related to the correct subkey) and $X_1, \ldots, X_{m-1}$ (related to the wrong subkey candidates) for the reduced subkey space, resp. $X_1, \ldots, X_{m'-1}$ when all possible subkeys are admissible. We may assume $Y \sim N(\mu_A, \sigma_A^2)$ while $X_j \sim N(\mu_B, \sigma_B^2)$ for $j \leq m-1$, resp. for $j \leq m'-1$, with unknown expectations $\mu_A$ and $\mu_B$ and variances $\sigma_A^2$ and $\sigma_B^2$. Clearly, $\mu_A \neq \mu_B$ since our attack exploits differences of the average execution times. Since it only exploits the relation between two table lookups $\sigma_A^2 \approx \sigma_B^2$ seems to be reasonable, the variances clearly depending on $N$. W.l.o.g. we may assume $\mu_A > \mu_B$. We point out that $E(X_1 + ... + X_{m-1} + Y)/m \approx \mu_B$ unless $m$ is very small.

$$\begin{aligned} &\mathrm{P}rob(\text{correct guess}) \approx \mathrm{P}rob(|Y - \mu_B| > \max\{|X_1 - \mu_B|, \ldots, |X_{m-1} - \mu_B|\}) \\ &= \mathrm{P}rob(\min\{X_1, ...., X_{m-1}\} > \mu_B - (Y - \mu_B), \max\{X_1, ..., X_{m-1}\} < Y) \\ &\approx \mathrm{P}rob(\max\{X_1, ..., X_{m-1}\} < Y)^2 \end{aligned} \tag{8}$$

Unless $m$ is very small the $\approx$ sign should essentially be "=". If the random variables $Y, X_1, \ldots, X_{m-1}$ were independent we had

$$\mathrm{P}rob(\max\{X_1, ..., X_{m-1}\} \leq t) = \prod_{j=1}^{m-1} \mathrm{P}rob(X_j \leq t) = \tag{9}$$
$$= \Phi((t - \mu_B)/\sigma_B)^{m-1}$$

where $\Phi$ denotes the cumulative distribution function of the standard normal distribution. From (9) one immediately deduces

$$\text{P}rob(\max\{X_1,...,X_{m-1}\} < Y) \approx \int_{-\infty}^{\infty} \Phi((z-\mu_B)/\sigma_B)^{m-1} f_A(z)dz \qquad (10)$$

where $Y$ has density $f_A$. In the context of Subsect. 3.3 the random variables $Y, X_1, \ldots, X_{m-1}$ are yet dependent. However, for different subkey candidates $k_i$ and $k_j$ the size of the intersection of the respective subsets is small compared to the size of these subsets themselves. Hence we may hope that the influence of the correlation between $X_i$ and $X_j$ is negligible. Under this asumption (10) provides a concrete formula for the probability for a true guess. However, this formula cannot be evaluated in practice since $\mu_A, \mu_B$ and $\sigma_A^2 \approx \sigma_B^2$ are unknown. Instead, we prove useful Lemma.

**Lemma 1.** *(i) Let $f$ denote a probability density, while $0 \le g, h \le 1$ are integrable functions and $M_c := \{y : g(y) \le c\}$. Assume further that $h \ge g$ on $R \setminus M_c$. Then*

$$\int h(z)f(z)dz \ge 1 - \frac{\epsilon}{1-c} \quad \textit{if} \quad \int g(z)f(z)dz = 1-\epsilon \qquad (11)$$

*(ii) Let $s, u, b > 1$. Then there exists a unique $y_0 > 0$ with $\Phi(y_0 s)^{ub} = \Phi(y_0)^b$. In particular, $\Phi(ys)^{ub} > \Phi(y)^b$ iff $y > y_0$.*

Proof. Assertion (i) follows immediately from

$$(1-c)\int_{M_c} f(z)dz \le \int_{M_c} (1-g(z))f(z)dz \le \int (1-g(z))f(z)dz = \epsilon$$

and hence

$$\begin{aligned}\int h(z)f(z)\,dz &\ge 0 + \int_{\mathrm{R}\setminus M_c} g(z)f(z)\,dz = (1-\epsilon) - \int_{M_c} g(z)f(z) \\ &\ge (1-\epsilon) - c\int_{M_c} f(z)\,dz \ge 1-\epsilon-\frac{c\epsilon}{1-c} = 1-\frac{\epsilon}{1-c}.\end{aligned}$$

Since $\Phi(ys)^{ub}/\Phi(y)^b = (\Phi(ys)^u/\Phi(y))^b$ we may assume $b = 1$ in the remainder w.l.o.g. Clearly, $\Phi(ys)^u < \Phi(y)$ for $y < 0$. Hence we concentrate to the case $y \ge 0$. In particular, $\log(1-x) = -x + O(x^2)$ implies

$$\begin{aligned}\psi(y) &:= \log\left(\Phi(ys)^u/\Phi(y)\right) = u\log(\Phi(ys)) - \log(\Phi(y)) \\ &= u\log(1-(1-\Phi(ys))) - \log(1-(1-\Phi(y))) \\ &= -u\,(1-\Phi(ys)) + (1-\Phi(y)) + O\left((1-\Phi(y))^2\right) \\ &\ge \frac{1}{\sqrt{2\pi}}\left(\frac{1}{y}-\frac{1}{y^3}\right)e^{-y^2/2} - \frac{1}{\sqrt{2\pi}}\frac{u}{ys}\left(e^{-y^2/2}\right)^{s^2} + O\left(\left(\frac{1}{y}e^{-y^2/2}\right)^2\right) \\ &> 0 \text{ for sufficiently large } y, \qquad \text{and } \lim_{y\to\infty}\psi(y) = 0. \end{aligned} \qquad (12)$$

We note that the last assertion follows immediately from the definition of $\psi$ while the '$\ge$' sign is a consequence from a well-known inequality of the tail of

$1 - \Phi$ (see, e.g., [12], Chap. VII, 175 (1.8)). Since $\psi$ is continuous and $\psi(0) = \log(0.5^{u-1}) < 0$ there exists a minimal $y_0 > 0$ with $\psi(y_0) = 0$. For any $y_1 \in \{y \geq 0 \mid \psi'(y) = 0\}$ the second derivative simplifies to $\psi''(y_1) = t(y_1)\Phi'(y_1)/\Phi(y_1)$ with $t(x) := (1 - s^2)x + (1 - 1/u)\Phi'(x)/\Phi(x)$. (Note that $\Phi''(ys) = -ys\Phi'(ys)$ and $\Phi''(y) = -y\Phi'(y)$.) Assume that $\psi(y_{00}) = 0$ for any $y_{00} > y_0$. As $\psi(0) < 0$ and $\psi(y_0) = \psi(y_{00}) = 0$ the function $\psi$ attains a local maximum in some $y_m \in [0, y_{00})$. Since $t\colon [0, \infty) \to \mathrm{R}$ is strictly monotonously decreasing $\psi$ cannot attain a local minimum in $(y_m, \infty)$ (with $\psi(\cdot) \leq 0 = \psi(y_{00})$) which contradicts (12). This proves the uniqueness of $y_0$ and completes the proof of (ii).

Our goal is to apply Lemma 1 to the right-hand side of (10). We set $u := (m' - 1)/(m - 1)$, $b := 1$ and $s := \sqrt{r}$ with $r := N'/N$. Further, $f(z) := f_A(z)$, $g(z) := (\Phi((z - \mu_B)/\sigma_B))^{m-1}$ and $h(z) := (\Phi(\sqrt{r}(z - \mu_B)/\sigma_B))^{u(m-1)}$. By (ii) we have $c = \Phi((z_0 - \mu_B)/\sigma_B)^{m-1}$ and $M_c = (\infty, z_0]$ with $g(z_0) = h(z_0)$. Lemma 1 and (8) imply

$$\left[\mathrm{P}rob(\text{correct guess for } (m, N)) = (1 - \epsilon)^2\right] \Rightarrow \tag{13}$$
$$\left[\mathrm{P}rob(\text{correct guess for } (m', N' = rN)) \geq \left(1 - \frac{\epsilon}{1 - c}\right)^2\right]$$

providing a lower probability bound for a correct guess in the full key space attack. Note that $\mu_A, \mu_B, \sigma_A^2 \approx \sigma_B^2, N, r$ determine $\epsilon, c$ and $z_0$ which are yet unknown in real attacks since $\mu_A$ and $\mu_B$ are unknown. Example 1 gives an idea of the magnitude of $r$.

*Example 1.* Let $m = 2^{12}$, $m' = 2^{32}$, and $y_0 := (z_0 - \mu_B)/\sigma_B = \Phi^{-1}(c^{1/(m-1)})$. If $c = 0.5$ (resp., if $c = 100/101$) the number $r = N'/N = 3.09$ (resp., $r = 3.85$) gives $\Phi(y_0\sqrt{r})^{u(m-1)} = \Phi(y_0)^{m-1} = 0.5$ (resp., $= 100/101$).