Analysis of Countermeasures Against Access Driven Cache Attacks on AES

Johannes Blömer and Volker Krummel*

Faculty of Computer Science, Electrical Engineering and Mathematics
University of Paderborn, Germany
{bloemer,krummel}@uni-paderborn.de

Abstract. Cache based attacks (CBA) exploit the different access times of main memory and cache memory to determine information about internal states of cryptographic algorithms. CBAs turn out to be very powerful attacks even in practice. In this paper we present a general and strong model to analyze the security against CBAs. We introduce the notions of *information leakage* and *resistance* to analyze the security of several implementations of AES. Furthermore, we analyze how to use random permutations to protect against CBAs. By providing a successful attack on an AES implementation protected by random permutations we show that random permutations used in a straightforward manner are not enough to protect against CBAs. Hence, to improve upon the security provided by random permutations, we describe the property a permutation must have in order to prevent the leakage of some key bits through CBAs.

Keywords: cache attacks, AES, threat model, countermeasures, random permutations.

1 Introduction

Modern computers use a hierarchical organization of different types of memories among them fast but small cache memory and slow but large main memory. In 2002 Page [14] presented a theoretical attack on DES that exploited timing information to deduce information about cache accesses, which in turn reveal information about secret keys being used. In the sequel we call attacks that exploit information about the cache behavior *cache based attacks* or CBAs. In particular, it turned out that large tables such as sboxes render an encryption algorithm susceptible to CBAs. Tsunoo et al. [17] published a practical CBA against DES. Further publications of Page [15], Percival [16], Bernstein [3], Osvik et al. [13] and Brickell et al. [7] disclosed the full power of CBAs. See [4,10,12,1,2] for further improvements of CBAs. In particular, the fast AES implementation [8] is susceptible to CBAs. Note that the fast implementation is used in virtually all crypto libraries. It is susceptible to CBAs since it depends heavily on the usage of 5 large sboxes $\mathbf{T}_0, \ldots, \mathbf{T}_4$ each of the size of 1024 bytes.

* This work was partially supported by grants from Intel Corporation, Portland.

C. Adams, A. Miri, and M. Wiener (Eds.): SAC 2007, LNCS 4876, pp. 96–109, 2007.

In this paper we present a strong model for CBAs. Within this model we propose and analyze countermeasures that although they are quite general we describe in detail only for AES. As was pointed out by Bernstein in [3], the threat model that is often implicitly used for CBAs may not be strong enough. In particular, often it is assumed that the adversary $\mathcal{A}$ only can extract information from the cache before and after the encryption. This assumption is wrong from the theoretical point of view due to the process switching of the operating system. Moreover, it also has been practically disproved in [11]. Hence, several of the countermeasures proposed in the literature so far may not be effective. In this paper, we take into account powerful adversaries that are able to obtain cache information even during the encryption. Within this model we show that using random permutations to mitigate the leakage of information as proposed in [7] is not an effective countermeasure for CBAs against AES. On one hand, we present a CBA that shows that random permutations do not increase the complexity of CBAs as much as one might expect. On the other hand, the same attack shows that a random permutation does not prevent the leakage of the complete secret key. We also consider a modified countermeasure based on so called *distinguished permutations* that hedge a certain number of bits of the last round key in AES. By this we mean, that using our countermeasure a CBA on the last round of AES, say, will only reveal about half the bits of the last round key. As one can see, this is the least amount of leaking information that can be provably protected by permutations. To determine the remaining bits, an attacker has to combine the CBA with another attack, for example a CBA on the next to last round. We give a mathematical precise description and analysis of the property of permutations that we need for our countermeasure. This analysis also sheds some light on the difference between the CBA by Osvik et al. on the first two rounds of AES [13] and the attack of Brickell et al. on the tenth round of AES [7]. We give a more detailed comparison of these two attacks in [6]. Finally, we analyze the security of several implementations of AES against CBAs. One of these implementations is provably secure within our model and can also be used to protect the applications of permutations that are realized as table lookups. How to apply permutations securely has not been considered before.

The paper is organized as follows. In Section 2 we introduce our threat model. After that we introduce our main security measures, information leakage and resistance in Section 3. We use our security measures to analyze the security of several different implementations of AES in Section 4. In Section 5, first we consider random permutations as a countermeasure and describe a CBA on this countermeasure. Then, we present and discuss an improved countermeasure using so called distinguished permutations.

2 Threat Model

We consider computers with a single processor, fast but small cache memory and large but slow main memory. Every time a process wants to read a word from the main memory a portion of data in the size of a cache line is transferred to the

cache. An AES encryption or decryption process is running on that computer that takes as input a plaintext (or ciphertext) and computes the corresponding AES ciphertext (or plaintext) with a fixed secret key k. To define our threat model we make the following assumptions about an adversary $\mathcal{A}$.

1. $\mathcal{A}$ knows all technical details about the underlying cryptographic algorithm and its implementation, i.e., the position of sbox tables in memory.
2. $\mathcal{A}$ can feed the AES process with chosen plaintexts (or ciphertexts) and gets the corresponding ciphertexts (or plaintexts).
3. $\mathcal{A}$ can determine the indices of the cache lines that were accessed during the encryption (decryption). To do so $\mathcal{A}$ could use a similar method as described in [9]. In the sequel, we call the set of indices of accessed cache lines *cache information*. The plaintext or ciphertext together with the cache information is called a *measurement*.
4. $\mathcal{A}$ can restrict the cache information to certain rounds of the encryption. As mentioned in [3] that this assumption might be realistic and the authors of [7] practically proved its correctness.
5. $\mathcal{A}$ cannot distinguish between the elements of a single cache line.

A more detailed description of our threat model, i.e., further explanations and justifications of our assumptions, can be found in [6].

3 Information Leakage and Resistance

The threat model described above is stronger than the threat models published so far. The adversary is more powerful because $\mathcal{A}$ can restrict the cache information to a smaller interval of encryption operations. This reduces the number of accessed cache lines per measurement and increases the efficiency of CBAs. The main questions when analyzing the security against CBAs are information leakage and complexity of a CBA. After giving a formal definition of information leakage we introduce the notion of the so called *resistance* of an implementation as a measure that allows to estimate the complexity of a CBA.

Information leakage. The most important aspect of an implementation regarding the security against access driven CBAs is to determine the maximal amount of information that leaks via access driven CBAs. As we will see, the amount of leaking information about the secret key varies depending on the details of the CBA and the implementation of the cryptographic algorithm. We make the following definition:

Definition 1 (information leakage). *We consider an adversary who can mount a CBA using an arbitrary number of measurements as described in Assumption 3. Let $\widehat{\mathcal{K}}_i$ be the set of remaining key candidates for a key byte k_i^{10} at the end of the attack on AES. Then the leaking information is $8 - \log_2\left(|\widehat{\mathcal{K}}_i|\right)$ bits.*

The amount of leaking information allows to estimate the uncertainty of an attacker about the secret key that remains after an access driven CBA. To quantify

the maximal amount of information $\mathcal{A}$ can obtain about the secret key by access driven CBAs, we define $|CL|$ to be the size of a cache line in bits, $|S|$ the number of entries of the sbox and s the size of a single sbox element in bits. Hence, the number of elements that fits into a cache line is $\frac{|CL|}{s}$ and the cache information of a single measurement leaks at most $\log_2(|S|) - \log_2\left(\frac{|CL|}{s}\right) = \log_2\left(\frac{|S|}{|CL|} \cdot s\right)$ bits. Depending on the exact nature of an attack, the sets of measurements let the attacker reduce the number of remaining key candidates after the attack. The information leakage varies between 0 and 8 bits of information per byte. For example, the attack on the first round of [13] mounted on the fast implementation can determine at most 4 bits of every key byte regardless of the number of measurements. In contrast, the attack of [7] based on the last round allows an adversary to determine all key bits. In Section 4 we present an implementation that does not leak any information in our model.

Complexity of a CBA. The information leakage as defined above measures the maximal amount of information a CBA can provide using an arbitrary number of measurements. Determining the expected number of measurements an attacker needs to obtain the complete leaking information depends on the details of the implementation and on details of the CBA. For simplification we introduce the notion of so called resistance. It is a general measure to estimate the complexity of CBAs on different implementations.

Definition 2 (resistance). *The resistance of an implementation is the expected number E_r of key candidates that are proven to be wrong during a single measurement that is based on r rounds of the encryption.*

The larger E_r the more susceptible is the implementation to access driven CBAs. In particular, if an implementation does not leak any information, then an adversary cannot rule out key candidates and hence the resistance is 0. To compute E_r we always assume that all sbox lookups are independently and uniformly distributed. This assumption is justified because an attacker $\mathcal{A}$ usually does not have any information about the distribution of the sbox lookups. Hence, the best he can do in an attack is to choose the parts of the plaintexts/ciphertexts that are not relevant for the attack uniformly at random.

Let m be the number of cache lines needed to store the complete sbox. Each cache line can store v elements of an sbox. Furthermore, let w be the number of sbox lookups per round and let r be the number of rounds the attack focuses on. In an access driven CBA a key candidate is proven to be incorrect if it causes an access of a cache line that was not accessed during a measurement. Assuming that all sbox lookups are uniformly distributed the expected number of key candidates that can be sorted out after a single measurement is

$$E_r := \left(\frac{m-1}{m}\right)^{r \cdot w} \cdot m \cdot v \tag{1}$$

However, the maximal amount of information an arbitrary number of measurements can reveal is limited by the information leakage. Further measurements will

not reveal additional information. We verified by experiments that the number of measurements needed to achieve the full information leakage only depends on E_r.

In the sequel, we focus on methods to counteract CBAs. In general, there are two approaches to counteract such a side channel. The first approach is to use some kind of randomization to ensure that the leaking information does not reveal information about the secret key. Using randomization is a general strategy that protects against several kinds of side channel attacks, see for example [5]. In Section 5 we analyze a more efficient method based on random permutations. Before that, we consider the second approach, that is methods to reduce the bandwith of the side channel. We present several implementations of AES and examine their information leakage and their resistance.

4 Countermeasure 1: Modify Implementation

As Bernstein pointed out in [3] to thwart CBAs it is not sufficient to load all sbox entries into the cache before accessing the sbox in order to compute an intermediate result because $\mathcal{A}$ can get cache information at all times. Hence, loading the complete sbox into the cache does not suffice to hide all cache information. Therefore, he advises to avoid the usage of table lookups in cryptographic algorithms. Computing the AES SubBytes operation according to its definition $f : \{0,1\}^8 \to \{0,1\}^8$, $x \mapsto a \cdot \mathrm{INV}(x) \oplus b$ would virtually cause no cache accesses and hence seems to be secure against CBAs. However, implementing SubBytes like this would result in a very inefficient implementation on a PC. To achieve a high level of efficiency people prefer to use precomputed tables. In the sequel, we analyze the security of some well known and some novel variations of implementations of AES. First, we explain the different implementations of AES. See [8] for a detailed description of AES. After that we examine the information leakage and the resistance as defined in (1) against CBAs:

the standard implementation as described in Section 3 of [8].

the fast implementation as described in Section 4.2 of [8].

fastV1 is based on the fast implementation. The only difference is that the sbox $\mathbf{T}_4$ of round 10 is replaced by the standard sbox as proposed in [7].

fastV2 is also based on the fast implementation but uses only sbox $\mathbf{T}_0$. The description of the fast implementation of AES shows that the ith entry of the sboxes $\mathbf{T}_1, \ldots, \mathbf{T}_3$ is equal to the ith entry of the sbox $\mathbf{T}_0$ cyclically shifted by $1, 2$ and 3 bytes to the right respectively (see [8]). Hence, we propose to use only sbox $\mathbf{T}_0$ in the encryption and shift the result as needed to compute the correct AES encryption. E.g., to compute the sbox lookup $\mathbf{T}_1[i]$ using the sbox $\mathbf{T}_0$ we simply cyclically shift the value $\mathbf{T}_0[i]$ by 1 byte to the right.

small-n: A simple but effective countermeasure to counteract CBAs is to split the sbox $\mathbf{S}$ into n smaller sboxes $\mathbf{S}_0, \ldots, \mathbf{S}_{n-1}$ such that every small sbox $\mathbf{S}_i$ fits completely into a single cache line. An application $\mathbf{S}_i[x]$ of sbox $\mathbf{S}_i$ yields d_i bits of the desired result $\mathbf{S}[x]$. Hence, the correct result can be calculated by computing all bits separately and shift them into the correct position. We construct the small sboxes $\mathbf{S}_i$ for $0 \le i \le n-1$ as follows:

$$\mathbf{S}_i : \{0,1\}^8 \to \{0,1\}^{d_i},\ x \mapsto \lfloor \mathbf{S}[x] \rfloor_{(\sum_{j=0}^{i-1} d_j, (\sum_{j=0}^{i} d_j)-1)}$$

where $\lfloor y \rfloor_{(b,e)}$ are the bits $y_b \dots y_e$ of the binary representation of $y = (y_0, \dots, y_7)$. Instead of applying the sbox $\mathbf{S}$ to x directly each $\mathbf{S}_i$ is applied. The result is computed as $\mathbf{S}[x] = \sum_{i=0}^{n-1} \mathbf{S}_i[x] \cdot 2^{\sum_{j=0}^{i-1} d_j}$. In the sequel, we assume that the size of the sbox is a multiple of the size of a cache line and that all d_j are equal. Depending on the number n of required sboxes we call this implementation small-n. E.g., let $|CL| = 512$ and for $0 \le i \le 3$ let each $\mathbf{S}_i$ store the bits $\langle \mathbf{S}[x] \rangle_{2i,2i+1}$. The result $\mathbf{S}[x]$ is then computed as $\mathbf{S}[x] = \mathbf{S}_0[x] \oplus \mathbf{S}_1[x] \cdot 4 \oplus \mathbf{S}_2[x] \cdot 16 \oplus \mathbf{S}_3[x] \cdot 64$. We call this implementation *small*-4. Obviously, the performance depends on the number of involved sboxes and shifts to move bits into the right position. To estimate the efficiency we used the small-n variants in the last round of the fast implementation. Due to the inefficient bit manipulations on 32 bit processors our ad hoc implementation of using small-4 only in the last round shows that the penalty is about 60%. We expect that a more sophisticated implementation reduces this penalty significantly. Table 1 in the appendix shows a summary of timing measurements of the implementations described above. The measurements were done on a Pentium M (1400MHz) running linux kernel 2.6.18, gcc 4.1.1.

Next, we consider CBAs based on different sboxes and examine the information leakage and the resistance of each of the implementations described above. The standard implementation uses only a single sbox. Hence, a CBA as described above is based on that sbox. We verified by experiments that measurements taken over ≤ 3 rounds of the standard implementation leak all key bits. Experiments with a larger number of rounds are too complex due to the rapidly decreasing resistance E_r. We assume that even more rounds will leak all key bits. The resistance for all numbers of rounds is listed in column 1 of Table 2 in the appendix.

The second implementation is the fast implementation. The CBA on the first round of [13] on one of the sboxes $\mathbf{T}_0, \dots, \mathbf{T}_3$ shows that in this case the fast implementation will reveal half of the key bits, even with an arbitrary number of measurements. The resistance of the fast implementation against such an attack is shown in column 2 of Table 2. The CBA on the last round of [7] based on the sbox $\mathbf{T}_4$ shows that in this case the fast implementation leaks all key bits. Since this sbox is only used in the last round the resistance as shown in column 3 of Table 2 does not change for a different number of rounds.

The implementation called fastV1 also leaks all key bits. The resistance against CBAs based on sboxes $\mathbf{T}_0, \dots, \mathbf{T}_3$ remains the same as listed in column 2 of Table 2. The resistance against CBAs based on the standard sbox is shown in column 4 of Table 2. It remains constant over the number of rounds because the standard sbox is only used in the last round.

Like the fast implementation, the variation called fastV2 also leaks all key bits. It uses only the large sbox $\mathbf{T}_0$ in every round. The resistance for all possible numbers of rounds is listed in column 5 of Table 2.

Last, we consider the variants small-2, small-4 and small-8 that use smaller sboxes than the standard sboxes. Computing $\mathbf{S}[x]$ using variant small-4 or

small-8 leaks 0 bits of information having cache lines of size 512 bits because of two reasons:

1. Every $\mathbf{S}_i$ fits completely into a single cache line.
2. For every x each $\mathbf{S}_i$ is used exactly once to compute $\mathbf{S}[x]$.

Hence, the cache information remains constant for all inputs. The only assumption that is involved is that $\mathcal{A}$ cannot distinguish between the accesses on different elements within the same cache line (Assumption 5). We expect that the variant small-2 leaks all key bits in our setting. As we have shown above, the variants small-4 and small-8 leak no key bit and hence have resistance 0 (see column 7 and 8 of Table 2). The resistance of small-2 is listed in column 6 of Table 2.

Comparison of implementations. As Table 2 shows, the standard implementation provides rather good resistance against CBAs but only has low efficiency. The fast implementation provides the lowest resistance against CBAs but is very efficient. Its variants fastV1 and fastV2 are almost as efficient on 32 bit platforms but provide better resistance against CBAs. The variants using small sboxes provide the best resistance. Especially small-4 and small-8 prevent the leakage of information. For high security applications we propose to use one of the variants using small sboxes and adapt the number of sboxes to the actual size of cache lines of the system.

5 Countermeasure 2: Random Permutation

Another class of countermeasure that was already proposed but not analyzed in [7] is to use secret random permutations to randomize the accesses to the sbox. In this section we present a CBA against an implementation of AES secured by a random permutation that needs roughly 2300 measurements to reveal the complete key. This shows that the increase of the complexity of CBAs induced by random permutations is not as high as one would expect. In particular, the uncertainty of the permutation is not a good measure to estimate the gain of security. A random permutation has uncertainty of $\log_2(256!) \approx 1684$ bits and the uncertainty of the induced partition on the cache lines is $\log_2(256!/(16!)^{16}) \approx 976$ bits.

On the other hand, we present a subset of permutations, so called distinguished permutations, that reduce the information leakage from 8 bits to 4 bits per key byte. Hence, the remaining bits must be determined by an additional attack thereby increasing the complexity. In our standard scenario this is the best one can achieve.

We focus only on the protection of the last round of AES and we assume that the output x of the 9th round is randomized using some secret random permutation π. To be more precise, each byte x_i of the state $x = x_0, \ldots, x_{15}$ is substituted by $\pi(x_i)$. To execute the last round of AES a modified sbox $\mathbf{T}'_4$ that depends on π fulfilling $\mathbf{T}'_4[\pi(x_i)] = \mathbf{T}_4[x_i]$ is applied to every byte x_i. This

ensures that the resulting ciphertext $c = c_0, \ldots, c_{15}$ is correct. We denote the ℓ-th cache line used for the table lookups for $\mathbf{T}'_4$ by $CL_\ell, \ell = 0, \ldots, 15$. Hence, CL_ℓ contains the values $\{\mathbf{S}[\pi^{-1}(x)] | x = 16\ell, \ldots, 16\ell + 15\}$. Using a permutation π, information leaking through accessed cache lines does not depend directly on x_i but only on the permuted value $\pi(x_i)$. Since π is unknown to $\mathcal{A}$ the application of π prevents him to deduce information about the secret key $k^{10} = k_0^{10}, \ldots, k_{15}^{10}$ directly. However, in the sequel we will show how to bypass random permutations by using CBAs.

5.1 An Access Driven CBA on a Permuted Sbox

We assume that we have a fast implementation of AES that is protected by a random permutation π as described above. We also assume that the adversary $\mathcal{A}$ has access to the AES decryption algorithm. This assumption can be avoided. However, the exposition becomes easier if we allow $\mathcal{A}$ access to the decryption. We show how $\mathcal{A}$ can compute the bytes $k_0^{10}, \ldots, k_{15}^{10}$ of the last round key. Let $\widehat{k}_0$ denote a candidate for byte k_0^{10} of the last round key. In a first step for each possible value $\widehat{k}_0$ the adversary $\mathcal{A}$ determines the assignment $P_{\widehat{k}_0}$ of bytes to cache lines induced by π under the assumption that $\widehat{k}_0 = k_0^{10}$. To be more precise $\mathcal{A}$ computes a function

$$P_{\widehat{k}_0} : \{0,1\}^8 \to \{0, \ldots, 15\}$$

such that if $\widehat{k}_0$ is correct then for all x:

$$\pi(x) \in \{16P_{\widehat{k}_0}(x), \ldots, 16P_{\widehat{k}_0}(x) + 15\}.$$

I.e., if $\widehat{k}_0$ is correct then $P_{\widehat{k}_0}$ is the correct partition of values $\pi(x)$ into cache lines. Let us fix some x and a candidate $\widehat{k}_0$ for k_0^{10}. We set $c_0 = \mathbf{S}[x] \oplus \widehat{k}_0$ and $\widehat{M}_0 = \{0, \ldots, 15\}$. The adversary repeats the following steps for $j = 1, 2, \ldots,$ until $\widehat{M}_0$ contains a single element.

1. $\mathcal{A}$ chooses a ciphertext c^j, whose first byte is c_0, while the remaining bytes of c^j are chosen independently and uniformly at random.
2. Using his access to the decryption algorithm, $\mathcal{A}$ computes the plaintext p^j corresponding to the c^j.
3. By encrypting p^j, the adversary $\mathcal{A}$ determines the set D_0^j of indices of cache lines accessed for the table lookups for T'_4 during the encryption of p^j.
4. $\mathcal{A}$ sets $\widehat{M}_0 := \widehat{M}_0 \cap D_0^i$.

If $\widehat{M}_0 = \{y\}$, then $\mathcal{A}$ sets $P_{\widehat{k}_0}(x) = y$. Repeating this process for all x yields the function $P_{\widehat{k}_0}$ which has the desired property.

Under the assumption that the guess $\widehat{k}_0$ was correct, the function $P_{\widehat{k}_0}$ is the correct partition of values $\pi(x)$ into cache lines. Moreover, it is not difficult to see that the information provided by $P_{\widehat{k}_0}$ enables the adversary to mount an

attack similar to the CBA on the last round of [7]. This attack can be used to determine for each possible $\widehat{k}_0$ a set of vectors $\widehat{k}_1, \ldots, \widehat{k}_{15}$ of hypotheses for the other key bytes. For the time being, we assume that π has the property that for each $\widehat{k}_0$ there remains only a single vector of hypotheses for the other key bytes. In general, a random permutation has this property (for a mathematical precise definition and analysis of that property see Section 5.2). Hence, based on this property in the end there are only 256 AES keys left and a simple brute force attack reveals the correct one.

Cost Analysis. Experiments show that in the first step of the attack $\mathcal{A}$ needs on average 9 measurements consisting of a pair (p^i, c^i) and the corresponding cache information D_0^i such that the intersection $\widehat{M}_0 := \bigcap D_0^i$ contains only a single element $y = P_{\widehat{k}_0}(x)$. We need to determine the mapping $P_{\widehat{k}_0}(x)$ for every key candidate $\widehat{k}_0$ and every argument $x \in \{0,1\}^8$. Hence, a straightforward implementation of the attack needs roughly $256 \cdot 256 \cdot 9$ measurements to determine the function $P_{\widehat{k}_0}(x)$ for all arguments $x \in \{0,1\}^8$ and all key candidates $\widehat{k}_0 \in \{0,1\}^8$. However, one can reuse measurements for different key candidates $\widehat{k}_0, \widehat{k}_0'$ to reduce the number of measurements to roughly $256 \cdot 9 = 2304$. To determine the vector of hypothesis based on the candidate $\widehat{k}_0$ we can reuse the measurements obtained by determining the function $P_{\widehat{k}_0}$. Hence, the expected number of measurements of this attack is 2304.

5.2 Separability and Distinguished Permutations

From a security point of view, it is desirable to reduce the information leakage. E.g., a CBA alone should reveal as little information as possible, in particular it should not reveal the complete key. Then the adversary is forced to either mount a refined and more complex CBA based on other intermediate results or combine the CBA with some other method to determine the key bytes uniquely. In this case, the situation is similar to the attack of [13], where a CBA on the first round only reveals 4 bits of each key byte. Hence Osvik et al. combine CBAs on the first and second round of AES.

First, we present the property a permutation applied to the result of the 9-th round should have such that $\mathcal{A}$ cannot determine the key bytes uniquely using only a CBA on the last round. We denote the ℓth cache line by CL_ℓ and the elements of CL_ℓ by $a_0^{(\ell)}, \ldots, a_{15}^{(\ell)}$. Hence, the underlying permutation used to define this cache line is given by

$$\pi^{-1}(16\ell + j) = \mathbf{S}^{-1}[a_j^{(\ell)}]. \tag{2}$$

We say that a key candidate $\widehat{k}_0$ is *separable* from the first key byte k_0 of the last round if there exists a measurement that proves $\widehat{k}_0$ to be wrong. Conversely, a key candidate $\widehat{k}_0$ is *inseparable* from the key k_0 if there does not exist a measurement that proves $\widehat{k}_0$ to be wrong. More precisely, writing $\widehat{k}_0 = k_0 \oplus \delta$ the bytes $\widehat{k}_0$ and k_0 are inseparable if and only if

$$\forall \ell \in \{0, \ldots, 15\} \forall a \in CL_\ell : a \oplus \delta \in CL_\ell. \tag{3}$$

Notice that this property only depends on the difference δ and not on the value of k_0. Since there are 16 elements of the sbox in every cache line property (3) can only be satisfied by at most 16 differences. It turns out that for $|\Delta| = 16$ the set

$$\Delta := \{\delta \mid \text{for all } k_0 \in \{0,1\}^8 \text{ the bytes } k_0 \text{ and } k_0 \oplus \delta \text{ are inseparable}\}$$

forms a 4 dimensional subspace of $\mathbb{F}_{2^8}$ viewed as a 8 dimensional vector space over $\mathbb{F}_2$. It is obvious that the neutral element 0 is an element of Δ and that every $\delta \in \Delta$ is its own inverse. It remains to show that Δ is closed with respect to addition. Consider $\delta, \delta' \in \Delta$ and an arbitrary $a \in CL_\ell$. Then $a' = a \oplus \delta \in CL_\ell$ implies that $a' \oplus \delta' = a \oplus \delta \oplus \delta' \in CL_\ell$ because of (3) and $\delta \oplus \delta' \in \Delta$ holds.

Hence, any partition that has the maximal number of inseparable key candidates must generate a subspace of dimension 4. Using this observation we describe how to efficiently construct permutations such that the set Δ of inseparable differences has size 16. In the sequel, we will call any such permutation a *distinguished permutation*. Next, we describe how to construct the subspace.

Construction of the subspace. We first construct a set Δ of 16 differences that is closed with respect to addition over $\mathbb{F}_{256}$. We can do this in the following way

1. set $\Delta := \{\delta_0 := 0\}$, choose δ_1 uniformly at random from the set $\{1, \ldots, 255\}$, set $\Delta := \Delta \cup \{\delta_1\}$
2. choose δ_2 uniformly at random from $\{1, \ldots, 255\} \setminus \Delta$, set $\Delta := \Delta \cup \{\delta_2, \delta_3 := \delta_1 \oplus \delta_2\}$
3. choose δ_4 uniformly at random from $\{1, \ldots, 255\} \setminus \Delta$, set $\Delta := \Delta \cup \{\delta_4, \delta_5 := \delta_4 \oplus \delta_1, \delta_6 := \delta_4 \oplus \delta_2, \delta_7 := \delta_4 \oplus \delta_3\}$
4. choose δ_8 uniformly at random from $\{1, \ldots, 255\} \setminus \Delta$, set $\Delta := \Delta \cup \{\delta_8, \delta_9 := \delta_8 \oplus \delta_1, \delta_{10} := \delta_8 \oplus \delta_2, \delta_{11} := \delta_8 \oplus \delta_3, \delta_{12} := \delta_8 \oplus \delta_4, \delta_{13} := \delta_8 \oplus \delta_5, \delta_{14} := \delta_8 \oplus \delta_6, \delta_{15} := \delta_8 \oplus \delta_7\}$

This construction ensures that Δ is closed with respect to addition and hence Δ forms a subspace as desired.

Construction of the permutation. Now we can compute the function P that maps $\mathbf{S}[x] \in \mathbb{F}_2^8$ to a cache line. We use the fact that 16 proper translations of a 4 dimensional subspace form a partition of a 8 dimensional vector space $\mathbb{F}_2^8$. A basis $\{b_0, \ldots b_3\}$ of the subspace Δ can be expanded by 4 vectors $b_4, \ldots b_7$ to a basis of $\mathbb{F}_2^8$. The 16 translations of Δ generated by linear combinations of $b_4, \ldots, b_7$ form the quotient space $\mathbb{F}_2^8/\Delta$ that is a partition of $\mathbb{F}_2^8$. To construct the function P we do the following:

1. for every cache line CL_ℓ do
2. choose $a^{(\ell)}$ uniformly at random from $\mathbb{F}_{256}/\{a^{(j)} \oplus \delta \mid j < \ell, \delta \in \Delta\}$
3. fill CL_ℓ with the values of the set $\{a^{(\ell)} \oplus \delta \mid \delta \in \Delta\}$

Using (2) this partition into cache lines defines the corresponding permutation.

Analysis of the countermeasure. The security using a distinguished permutation as defined above rests on two facts.

1. Using a distinguished permutation where the set Δ of inseparable differences has size 16, a CBA on the last round of AES will reveal only four bits of each key byte k_i^{10}. Overall 64 of the 128 bits of the last round key remain unknown. Therefore, the adversary has to combine his CBA on the last round with some other method to determine the remaining 64 unknown bits. For example, he could try a modified CBA on the 9-th round exploiting his partial knowledge of the last round key. Or he could use a brute force search to determine the last round key completely.
2. There are several distinguished permutations and each of these permutations leads to 16! different functions mapping elements to 16 lines. If we choose randomly one of these functions, before an adversary can mount a CBA on the last round of [7], he first has to use some method like the one described in Section 5.1 to determine the function P that is actually used.

We stress that we consider the first fact to be the more important security feature. We saw already in Section 5.1 that determining a random permutation used for mapping elements to cache lines is not as secure as one might expect. Since we are using permutations of a special form the attack described in Section 5.1 can be improved somewhat. In the remainder of this section we briefly describe this improvement. To do so, first we have to determine the number of subspaces leading to distinguished permutations. As before view $\mathbb{F}_2^n := \{0,1\}^n$ as an n-dimensional $\mathbb{F}_2$ vector space. For $0 \leq k \leq n$ we define $D_{n,k}$ to be the number of k-dimensional subspaces of $\mathbb{F}_2^n$. To determine $D_{n,k}$ for V an arbitrary m-dimensional subspace of $\mathbb{F}_2^n$ we define

$$N_{m,k} := |\{(v_1, \ldots, v_k) | v_i \in V, v_1, \ldots v_k \text{ are linearly independent}\}|.$$

The number $N_{m,k}$ is independent of the particular m-dimensional subspace V, it only depends on the two parameters m and k. Then $D_{n,k} = \frac{N_{n,k}}{N_{k,k}}$. Next we observe that $N_{m,k} = \prod_{j=0}^{k-1}(2^m - 2^j) = 2^{k(k-1)/2} \prod_{j=0}^{k-1}(2^{m-j} - 1)$. Hence, we obtain that

$$D_{n,k} = \frac{\prod_{j=0}^{k-1}(2^{n-j} - 1)}{\prod_{j=0}^{k-1}(2^{k-j} - 1)}.$$

In our special case we have $n = 8$ and $k = 4$ and hence the number of 4 dimensional subspaces is $D_{8,4} = \frac{255 \cdot 127 \cdot 63 \cdot 31}{15 \cdot 7 \cdot 3 \cdot 1} = 200787$.

As mentioned above, each subspace leads to 16! different distinguished permutations. Hence, overall we have $200787 \cdot 16! \approx 2^{60}$ distinguished permutations. On the other hand, because of the special structure of our permutations, to determine the function P by CBAs can be done more efficiently than determining an arbitrary function mapping elements to cache lines (see Section 5.1). In particular, $\mathcal{A}$ only needs to observe about 7 accesses of a single but arbitrary cache line. With high probability this will be enough to determine a basis of the subspace being used. In addition, $\mathcal{A}$ needs at least one access for every other cache

line in order to determine the function P. The corresponding probability experiment follows the multinomial distribution. We did not calculate the expected number of tries exactly. Experiments show that if we can determine the accessed cache line exactly, on average 62 measurements suffice to compute the function P exactly. However, a single measurement only yields a set of accessed cache lines. But arguments similar to the ones used for the first part of the attack in Section 5.1 show that we need on average 9 measurements to uniquely determine an accessed cache line. Therefore, on average we need $62 \cdot 9 = 558$ experiments to determine the function P.

Hence, compared to the results of Section 5.1 we have reduced the number of measurements used to determine the function P by a factor of 3. However, we want to stress again, that the main security enhancement of using distinguished permutations instead of arbitrary permutations is the fact, that distinguished permutations have a lower information leakage. To improve the security, one can choose larger key sizes such as 192 bits or 256 bits. Since distinguished permutations protect half of the key bits, the remaining uncertainty about the secret key after CBAs can be provably increased from 64 bits to 96 bits or 128 bits, respectively. In the full version of the paper [6] we describe an efficient and secure realization of random and distinguished permutations using small sboxes as described in Section 4.

Separability and random permutations. In our CBA on an implementation protected by a random permutation (Section 5.1) we assumed that fixing a candidate $\widehat{k}_0$ determines the candidates for all other key bytes. With sufficiently many measurements for a fixed $\widehat{k}_0$ we can determine the function $P_{\widehat{k}_0}$ as defined in Section 5.1. Furthermore, we saw that the separability of candidates $\widehat{k}, \widehat{k}'$ depends only on their difference $\delta = \widehat{k} \oplus \widehat{k}'$. Hence, to be able to rule out all but one candidate $\widehat{k}_i$ at position i for a fixed $\widehat{k}_0$ the permutation π must have the following property:

$$\forall \delta \neq 0 \exists j \in \{0, \ldots, 15\} \exists a \in CL_j : a \oplus \delta \notin CL_j.$$

There are less than 2^{844} of the $256! \approx 2^{1684}$ permutations that do not have this property. Hence, a random permutation satisfies this condition with probability $1 - \frac{2^{844}}{2^{1684}}$.

6 Summary of Countermeasures and Open Problems

In this paper we presented and analyzed the security of several different implementations of AES. Moreover, we analyzed countermeasures based on permutations: random permutations and distinguished permutations. We give a short overview over the advantages and disadvantages of selected countermeasures:

countermeasure	# measurements	information in bits /security	efficiency
small-4	∞	0 / high	slow
random permutation	2300	128 / low	fast
distinguished permutations	560	64 / medium	fast

The second column shows the expected number of measurements an attacker has to perform in order to get the amount of information shown in the third column.

References

1. Aciiçmez, O., Koç, Ç.K.: Trace-driven cache attacks on AES (short paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 112–121. Springer, Heidelberg (2006)
2. Aciiçmez, O., Schindler, W., Koç, Ç.K.: Cache based remote timing attack on the AES. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 271–286. Springer, Heidelberg (2006)
3. Bernstein, D.J.: Cache-timing attacks on AES (2005), `http://cr.yp.to/papers.html`, Document ID: cd9faae9bd5308c440df50fc26a517b4
4. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: AES power attack based on induced cache miss and countermeasure. In: ITCC (1), pp. 586–591. IEEE Computer Society, Los Alamitos (2005)
5. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004)
6. Blömer, J., Krummel, V.: Analysis of countermeasures against access driven cache attacks on AES (full version). Cryptology ePrint Archive, Report 2007/282 (2007)
7. Brickell, E., Graunke, G., Neve, M., Seifert, J.-P.: Software mitigations to hedge AES against cache-based software side channel vulnerabilities. Cryptology ePrint Archive, Report 2006/052 (2006), `http://eprint.iacr.org/`
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002)
9. Hu, W.-M.: Lattice scheduling and covert channels. In: IEEE Symposium on Security and Privacy, pp. 52–61. IEEE Computer Society Press, Los Alamitos (1992)
10. Lauradoux, C.: Collision attacks on processors with cache and countermeasures. In: Wolf, C., Lucks, S., Yau, P.-W. (eds.) WEWoRC. LNI, vol. 74, pp. 76–85 (2005)
11. Neve, M., Seifert, J.-P.: Advances on access-driven cache attacks on AES. In: Proceedings of Selected Areas in Cryptography 2006 (2006)
12. Neve, M., Seifert, J.-P., Wang, Z.: A refined look at Bernstein's AES side-channel analysis. In: Lin, F.-C., Lee, D.-T., Lin, B.-S., Shieh, S., Jajodia, S. (eds.) ASIACCS, p. 369. ACM, New York (2006)
13. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
14. Page, D.: Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Report 2002/169 (2002), `http://eprint.iacr.org/`
15. Page, D.: Partitioned cache architecture as a side-channel defence mechanism. Cryptology ePrint Archive, Report 2005/280 (2005), `http://eprint.iacr.org/`
16. Percival, C.: Cache missing for fun and profit. In: BSDCan 2005 (2005)
17. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of DES implemented on computers with cache. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 62–76. Springer, Heidelberg (2003)

A Appendix

Table 1. Timings for different implementations of AES

# sboxes	fast	standard	fastV1	fastV2	small-2	small-4	small-8
time factor	1	~ 3	~ 1	~ 1	1.32	1.6	1.95

Table 2. The resistance E_r of AES implementations as defined in (1)

	1	2	3	4	5	6	7	8
	standard	fast	fast T_4	fastV1	fastV2	small-2	small-4	small-8
	$\mathbf{S}$	$\mathbf{T}_0, \ldots, \mathbf{T}_3$	$\mathbf{T}_4$	$\mathbf{S}$	$\mathbf{T}_0$	$\mathbf{S}_0, \mathbf{S}_1$	$\mathbf{S}_0, \ldots, \mathbf{S}_3$	$\mathbf{S}_0, \ldots, \mathbf{S}_7$
E_1	2.57	198.0	91.2	2.57	91.2	$3.91 \cdot 10^{-3}$	0	0
E_2	$2.57 \cdot 10^{-2}$	153.0	91.2	2.57	32.5	$5.96 \cdot 10^{-8}$	0	0
E_3	$2.58 \cdot 10^{-4}$	118.0	91.2	2.57	11.6	$9.09 \cdot 10^{-13}$	0	0
E_4	$2.58 \cdot 10^{-6}$	91.2	91.2	2.57	4.12	$1.39 \cdot 10^{-17}$	0	0
E_5	$2.59 \cdot 10^{-8}$	70.4	91.2	2.57	1.47	$2.12 \cdot 10^{-22}$	0	0
E_6	$2.59 \cdot 10^{-10}$	54.4	91.2	2.57	$5.22 \cdot 10^{-1}$	$3.23 \cdot 10^{-27}$	0	0
E_7	$2.60 \cdot 10^{-12}$	42.0	91.2	2.57	$1.86 \cdot 10^{-1}$	$4.93 \cdot 10^{-32}$	0	0
E_8	$2.61 \cdot 10^{-14}$	32.5	91.2	2.57	$6.62 \cdot 10^{-2}$	$7.52 \cdot 10^{-37}$	0	0
E_9	$2.61 \cdot 10^{-16}$	25.1	91.2	2.57	$2.36 \cdot 10^{-2}$	$1.15 \cdot 10^{-41}$	0	0
E_{10}	$2.62 \cdot 10^{-18}$	25.1	91.2	2.57	$8.39 \cdot 10^{-3}$	$1.75 \cdot 10^{-46}$	0	0