Trace-Driven Cache Attacks on AES

Onur Acıiçmez[1] and Çetin Kaya Koç[2]

Abstract

Cache based side-channel attacks have recently been attracted significant attention due to the new developments in the field. In this paper, we present efficient trace-driven cache attacks on a widely used implementation of the AES cryptosystem. We also evaluate the cost of the proposed attacks in detail under the assumption of a noiseless environment. We develop an accurate mathematical model that we use in the cost analysis of our attacks. We use two different metrics, specifically, the expected number of necessary traces and the cost of the analysis phase, for the cost evaluation purposes. Each of these metrics represents the cost of a different phase of the attack.

Keywords

Side-channel cryptanalysis, cache attacks, trace-driven attacks, AES

1 Introduction

Implementations of cryptosystems may leak information through side channels due to the properties and 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 cryptographic algorithm. Power, electromagnetic, and timing attacks are well-known side-channel attacks studied in the past [14, 12, 15, 4]. In this paper, we focus on a type of side-channel cryptanalysis that takes advantage of the information that leaks through the cache architecture of a CPU.

There are various cache based side-channel attacks in the literature, which are discussed in detail in the next section. Trace-driven attacks are one of the three types of cache based attacks that had been distinguished so far. We present different trace-driven cache based attacks on AES in this

[1]Oregon State University, School of EECS, Corvallis, OR 97331, USA, aciicmez@eecs.oregonstate.edu

[2]Information Security Research Center, Istanbul Commerce University, Eminönü, Istanbul 34112, TURKEY koc@cryptocode.net

paper. There are already two trace-driven attacks on AES in the literature [7, 16]. However, our attacks require significantly less number of measurements (e.g. only 5 measurements in some cases) and are much more efficient than the previous attacks. We show that trace-driven attacks have indeed much more power than what was stated in the previous studies.

Furthermore, we present a robust computational model for trace-driven attacks that allows one to evaluate the cost of such attacks on a given implementation and platform. Although, we only apply our model to the attacks on AES, it can also be used for other symmetric ciphers like DES. The main contribution of our model to the field is that it can be used to quantitatively analyze the cost of trace-driven attacks on different implementations of a cipher. Therefore, we can analyze the effectiveness of various mitigations that can be used against such attacks. Thus, a designer can use our model to determine which mitigations she needs to implement against trace-driven attacks to achieve a predetermined security level.

2 Background and Previous Work

The feasibility of the cache based side-channel attacks, abbreviated to "cache attacks" from here on, was first mentioned by Kocher and then Kelsey et al. in [13, 14]. D. Page described and simulated a theoretical cache attack on DES [23]. Actual cache based timing attacks were implemented by Tsunoo et al. [27, 28]. The original attack on MISTY1 proposed in [28] has recently been improved in [29].

The topic of cache based side-channel attacks has been very popular since early 2005. Although, cache side-channel threat had been known for a couple of years, the first efficient and realistic attacks were not developed until 2005. Bernstein showed the vulnerability of AES software implementations on various platforms [5]. There was a common belief that Bernstein's attack is a realistic remote attack and it can recover an entire AES key. However, Neve et al. showed in [17] that this is only a fallacy. They described the circumstances in which the attack might work and also the limitations of the Bernstein attack. The details of this analysis can also be found in [19].

Simultaneously, but independently of Bernstein's efforts, a research team that consists of Acıiçmez, Schindler, and Koç developed a realistic remote attack on the AES. Although there is not any publicly available report of their work, they presented the basics of the attack in several occasions [2].

Osvik et al. described various local cache attack variants first in [21] in 2005, then they presented their results at CT-RSA in early 2006 [22]. They made use of a local array and exploited the collisions between the table lookups and the access operations to this array. Neve et al. improved the attacks in [22] by taking the last AES round into consideration [18]. The same idea of exploiting collisions between two different processes was also

used by Colin Percival in [26]. He made use of simultaneous-multithreading feature of the modern processors and developed a cache attack on RSA.

Similar to external collisions between different processes, the internal collisions inside a cipher can also be taken advantage of. Internal cache collisions were first used in [27] and [28]. The remote attack of Acıiçmez et al. and Lauradoux's attack are also based on internal collisions [2, 16]. A recent manuscript that summarizes cache collision attacks on AES will be presented at CHES'06 [6]. Several hardware and software based countermeasures were proposed to prevent cache attacks in [24, 25, 21, 5, 8].

There are three different types of cache attacks, namely time-driven, trace-driven, and access-driven. Time-driven and trace-driven attacks were first described by Page in [23]. Access-driven attacks are relatively new and first seen in [21, 22]. The difference between these attack types are the capabilities of the adversary.

The adversary is assumed to be able to capture the profile of the cache activity during an encryption in trace-driven attacks. This profile includes the outcomes of every memory access the cipher issues in terms of cache hits and misses. Therefore, the adversary has the ability to observe if a particular 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 because they do not rely on the ability of capturing the outcomes of individual memory accesses. Adversary is assumed to be able to observe the aggregate profile, i.e., total numbers of cache hits and misses or at least a value that can be used to approximate these numbers. For example, the total execution time of the cipher can be measured and used to make inferences about the number of cache misses in a time-driven cache attack.

In access-driven 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 wrong key assumptions that would cause an access to unaccessed parts of the tables can be eliminated.

2.1 Overview of Trace-Driven Cache Attacks

Trace-driven attacks on AES were first presented in [16] and [7]. We describe much more efficient attacks in this paper. Our two-round attack is a known-plaintext attack and exploits the collisions among the first two rounds of AES. A more efficient version, which we call the last round attack, considers last round accesses and is a known-ciphertext attack.

In trace-driven cache attacks, the adversary obtains the traces of cache hits and misses for a sample of encryptions and recovers the secret key of a cryptosystem using this data. We define a trace as a sequence of cache hits

and misses. For example,

$$MHHM, HMHM, MMHM, HHMH, MMMM, HHHH$$

are examples of a trace of length 4. Here H and M represents a cache hit and miss respectively. The first one in the first example is a miss, second one is a hit, and so on. If an adversary captures such traces, she can determine whether a particular access during an encryption is a hit or miss.

The trace of an encryption can be captured by the use of power consumption measurements as done in [7]. It is still a question if these traces can be obtained more easily using more sophisticated methods like exploiting processor specific features, e.g., by making use of performance counters. In this paper, we do not get into the details of how to capture cache traces.

We analyze trace-driven attacks on AES under the assumption that the adversary can capture the traces of AES encryption. This assumption corresponds to clean measurements in a noiseless environment. In reality, an adversary may have noise in the measurements in some circumstances, in which case the cost of the attack increases depending on the amplitude of the noise. However, an analysis under the above assumption results a more clear understanding of the attack cost. Assumption of a noiseless environment also enables us to make more reliable comparison of different attacks. Comparison of different attacks on a specific platform is not very reliable, because the cost of any side-channel attack depends on the noise in the measurements and the amplitude of the noise significantly differs between different platforms.

In a side-channel attack, there are essentially two different phases:

- Online Phase: consists of the collection of side-channel information of the target cipher. This phase is also known as the sampling phase of the attack. The adversary encrypts or decrypts different input values and measures the side-channel information, e.g., power consumption or execution time of the device.

- Offline Phase: is also known as the analysis phase. In this phase, the adversary processes the data collected in the online phase and makes predictions and verifications regarding the secret value of the cipher.

An adversary usually performs the former phase completely before the latter one. However, in some cases, especially in adaptive chosen-text attacks (e.g. [9, 1]), these two phases may overlap and may be performed simultaneously.

We use two different metrics to evaluate the cost of our attacks presented in this paper. The first metric is *the expected number of traces* that we need to capture to narrow the search space of the AES key down to a certain degree. The second metric is *the average number of operations* we need to perform to analyze the captured traces and eliminate the wrong key

assumptions. These metrics basically represent the cost of the online and offline phases of our attacks. In other words, the first metric gives the cost of the online phase and the second gives that of the offline phase. As the reader can clearly see in this paper, there is a trade-off between the costs of these two phases.

3 Trace-Driven Cache Attacks on the AES

In this paper, we present trace-driven attacks on the most widely used implementation of AES [3], and estimate their costs. We assume that the cache does not contain any AES data prior to each encryption, because the captured traces cannot be accurate otherwise. Therefore, the adversary is assumed to clean the cache (e.g., by loading some garbage data as done in [28, 27, 22, 26]) before the encryption process starts.

Another assumption we make is that the data in AES lookup tables cannot be evicted from the cache during the encryption once they are loaded into the cache. This assumption means that each lookup table can only be stored in a different non-overlapping location of the cache and there is no context-switch during an encryption or any other process that runs simultaneously with the cipher and evicts the AES data. These assumptions hold if the cache is large enough, which is the case for most of the current processors. An adversary can also discard a trace if a context-switch occurs during the measurement.

We also assume that each measurement is composed of the cache trace of a single message block encryption. In this paper, we only consider AES with 128-bit key and block sizes. Our attacks can easily be adapted to longer key and block sizes; however we omit these cases for the sake of simplicity.

The implementation we analyze is described in [11] and it is suitable for 32-bit architectures. It employs 4 different lookup tables in the first 9 rounds and a different one in the last round. In this implementation, all of the component functions, except AddRoundKey, are combined into four different tables and the rounds turn to be composed of table lookups and bitwise exclusive-or operations as shown in Figure 1.

In each round, except the last one, it makes 4 references to each of the first 4 tables. The S-box lookups in the final round are implemented as table lookups to another 1KB-large table , called T4, with 256 many 32-bit elements. There are 16 accesses to T4 in that round. The indices of these accesses are S_i^{10}, where S_i^t is the byte i of intermediate state value that becomes the input of round t and $i \in \{0, .., 15\}$. Let C be the ciphertext, i.e. the output of the last round, and represented as an array of 16 bytes, $C = (c_0, c_1, ..., c_{15})$. Individual bytes of C are computed as:

$$c_i = Sbox[S_w^{10}] \oplus RK_i^{10} \ ,$$

All rounds except the last:

$$(S_0^{(r+1)},S_1^{(r+1)},S_2^{(r+1)},S_3^{(r+1)}) = \text{T0}[S_0^r] \oplus \text{T1}[S_5^r] \oplus \text{T2}[S_{10}^r] \oplus \text{T3}[S_{15}^r] \oplus (RK_0^r,RK_1^r, RK_2^r, RK_3^r)$$

$$(S_4^{(r+1)},S_5^{(r+1)},S_6^{(r+1)},S_7^{(r+1)}) = \text{T0}[S_4^r] \oplus \text{T1}[S_9^r] \oplus \text{T2}[S_{14}^r] \oplus \text{T3}[S_3^r] \oplus (RK_4^r,RK_5^r, RK_6^r, RK_7^r)$$

$$(S_8^{(r+1)},S_9^{(r+1)},S_{10}^{(r+1)},S_{11}^{(r+1)}) = \text{T0}[S_8^r] \oplus \text{T1}[S_{13}^r] \oplus \text{T2}[S_2^r] \oplus \text{T3}[S_7^r] \oplus (RK_8^r,RK_9^r, RK_{10}^r,RK_{11}^r)$$

$$(S_{12}^{(r+1)},S_{13}^{(r+1)},S_{14}^{(r+1)},S_{15}^{(r+1)}) = \text{T0}[S_{12}^r] \oplus \text{T1}[S_1^r] \oplus \text{T2}[S_6^r] \oplus \text{T3}[S_{11}^r] \oplus (RK_{12}^r,RK_{13}^r,RK_{14}^r,RK_{15}^r)$$

Last round:

$$(c_0, c_1, c_2, c_3) = \text{Sbox}[S_0^{10}] \oplus \text{Sbox}[S_5^{10}] \oplus \text{Sbox}[S_{10}^{10}] \oplus \text{Sbox}[S_{15}^{10}] \oplus (RK_0^{10},RK_1^{10},RK_2^{10},RK_3^{10})$$

$$(c_4, c_5, c_6, c_7) = \text{Sbox}[S_4^{10}] \oplus \text{Sbox}[S_9^{10}] \oplus \text{Sbox}[S_{14}^{10}] \oplus \text{Sbox}[S_3^{10}] \oplus (RK_4^{10},RK_5^{10},RK_6^{10},RK_7^{10})$$

$$(c_8, c_9, c_{10}, c_{11}) = \text{Sbox}[S_8^{10}] \oplus \text{Sbox}[S_{13}^{10}] \oplus \text{Sbox}[S_2^{10}] \oplus \text{Sbox}[S_7^{10}] \oplus (RK_8^{10},RK_9^{10},RK_{10}^{10},RK_{11}^{10})$$

$$(c_{12},c_{13}, c_{14}, c_{15}) = \text{Sbox}[S_{12}^{10}] \oplus \text{Sbox}[S_1^{10}] \oplus \text{Sbox}[S_6^{10}] \oplus \text{Sbox}[S_{11}^{10}] \oplus (RK_{12}^{10},RK_{13}^{10},RK_{14}^{10},RK_{15}^{10})$$

Figure 1: The round computations in the AES

where RK_i^{10} is the i^{th} byte of the last round key and $Sbox[S_w^{10}]$ is the S-box output for the input S_w^{10} for a known $w \in \{0, 1, ..., 15\}$. The S-box in AES implements a permutation, and therefore its inverse, i.e. $Sbox^{-1}$, exists.

In this paper, we present our attacks under the assumption that the AES memory accesses are issued by the processor in the certain order given in Figure 1, i.e., first T0$[S_0^r]$, second T1$[S_5^r]$, etc. However, the actual order is implementation specific and may differ from our assumption. Our attacks can easily be adapted to any given order without any performance loss.

3.1 Overview of an Ideal Two-Round Attack

The access indices in the first round are in the form $P_i \oplus K_i$, where P_i and K_i are the i^{th} bytes of the plaintext and the cipherkey respectively and $i \in \{0, 1, ..., 15\}$. The indices of the first 4 references to the first table, T0, are:

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

The outcome of the second access to T0, i.e. the one with the index $P_4 \oplus K_4$, gives information about K_0 and K_4. For example, if the second access results a cache hit, we can directly conclude that the index $P_4 \oplus K_4$ has to be equal to the index of the first access, i.e., $P_0 \oplus K_0$. If it is a cache miss, then the inequality of these values becomes true. We can use this fact to find the

correct key byte difference $K_0 \oplus K_4$.

$$P_0 \oplus K_0 = P_4 \oplus K_4 \quad => \quad K_0 \oplus K_4 = P_0 \oplus P_4$$

$$P_0 \oplus K_0 \neq P_4 \oplus K_4 \quad => \quad K_0 \oplus K_4 \neq P_0 \oplus P_4$$

In other words, if we capture a cache trace during the first round of AES and the second access to T0 results in a cache hit, then we can directly conclude that $K_0 \oplus K_4 = P_0 \oplus P_4$. Recall that the plaintext is assumed to be known to an attacker and the cache is clean prior to the first table lookup so that the first access to a table always results in a cache miss.

On the other hand, if we see a miss, then $K_0 \oplus K_4$ cannot be equal to $P_0 \oplus P_4$ and we can eliminate this wrong value. If we collect a sample of traces, we can find the correct value of $K_0 \oplus K_4$ by either eliminating all possible wrong values or directly finding the correct value when we realize a cache hit in the second access in any of the sampled traces.

We can also find the other key byte differences $K_i \oplus K_j$, where $i,j \in \{0,4,8,12\}$, using the same idea. We can further reduce the search space by considering the accesses to other three tables. In general, we can obtain $K_i \oplus K_{4*j+i}$, where $i,j \in \{0,1,2,3\}$, and it is enough to find the entire 128-bit key by searching only 32 bits.

A final search space of 32 bits is only a theoretical lower bound in the first round attack due to the complications explained in Subsection 3.3. We also have to consider second round accesses to really reduce the search space to 32 bits. The first round attack only reveals some of the bits of $K_i \oplus K_j$. However, when we examine the collisions between the first and second round accesses in the same way, i.e., in a "two-round attack", we can reveal the entire AES key.

3.2 Overview of an Ideal Last Round Attack

Another way to find the cipherkey is to exploit the collisions between the last round accesses. The outcomes of the last round accesses to T4 leaks information about the values of the last round key bytes, i.e., RK_i^{10} where $i \in \{0,..,15\}$.

For example, if the second access to T4 results in a cache hit, we can conclude that the indices S_0^{10} and S_1^{10} are equal. If it is a cache miss, then the inequality of these values becomes true. We can use this fact to find the correct round key bytes RK_0^{10} and RK_1^{10} as the following.

We can write the value of S_w^{10} in terms of RK_i^{10} and c_i:

$$S_w^{10} = Sbox^{-1}[c_i \oplus RK_i^{10}] \ ,$$

If S_0^{10} and S_5^{10} are equal, so are $Sbox^{-1}[c_0 \oplus RK_0^{10}]$ and $Sbox^{-1}[c_1 \oplus RK_1^{10}]$, which also mandates the equality of $c_0 \oplus RK_0^{10}$ and $c_1 \oplus RK_1^{10}$. This equality

can also be written as

$$c_0 \oplus RK_0^{10} = c_1 \oplus RK_1^{10} \Rightarrow c_0 \oplus c_1 = RK_0^{10} \oplus RK_1^{10}$$

Since the value of C is known to the attacker, $RK_0^{10} \oplus RK_1^{10}$ can directly be computed from the values of c_0 and c_1 if the second access to T4 results in a cache hit. In case of a cache miss, we can replace the = sign in the above equations with $\neq$ and we can use the inequalities to eliminate the values that cannot be the correct value of $RK_0^{10} \oplus RK_1^{10}$.

The value of RK_2^{10} relative to RK_0^{10} can also be determined by analyzing the first three accesses to T4 after the correct value of $RK_0^{10} \oplus RK_1^{10}$ is found. Similarly, if we extend our focus to the first four accesses, we can find RK_3^{10}. Then we can find RK_4^{10} and so on.

In general, we can find all of the round key byte differences $RK_i^{10} \oplus RK_j^{10}$, where $i, j \in \{0, 1, ..., 15\}$. The value of any single byte RK_i^{10} can be searched exhaustively to determine the entire round key. After revealing the entire round key, it becomes trivial to compute the actual secret key, because the key expansion of the AES cipher is a reversible function.

3.3 Complications in Reality and Actual Attack Scenarios

In a real environment, even if the index of the second access to a certain lookup table is different than the index of the first access, a cache hit may still occur. Any cache miss results in the transfer of an entire cache line, not only one element, from the main memory. Therefore, if the former access retrieves an element, which lies in the same cache line of the previously accessed data, a cache hit will occur.

Let δ 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 δ/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. Using this fact, we can find the difference of the most significant part of the key bytes using the equation:

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

where $\langle A \rangle$ stands for the most significant part of A.

Therefore, we can only reveal $\langle K_i \oplus K_{4*j+i} \rangle$, where i,$j \in \{0,1,2,3\}$, using the collisions in the first round. Notice that $(8 - \ell)$ is the size of the most significant part of a table entry in terms of the number of bits, where $\ell = log_2(\delta/k)$. First round attack allows us to reduce the search space by $12 * (8 - \ell)$ bits. In theory ℓ can be as low as zero bits, in which case the search space becomes only 32 bits. The most common values of δ are 32 and 64

in widely used processors. For $\delta = 64$ the search space is reduced by 48 bits yielding an 80 bit final search space. This is the reason why we need to consider the second round indices along with the first round to achieve full key disclosure.

This complication does affect the last round attack too. We observe a cache hit in the second access to T4 whenever

$$\langle S_0^{10} \rangle = \langle S_5^{10} \rangle \ ,$$

and so

$$\langle Sbox^{-1}[c_0 \oplus RK_0^{10}] \rangle = \langle Sbox^{-1}[c_1 \oplus RK_1^{10}] \rangle \ .$$

However due to the nonlinearity of the AES S-box, only the correct RK_0^{10} and RK_1^{10} values obey the above equation for every ciphertext sample. Therefore, we need to find the correct RK_0^{10} and RK_1^{10} values instead of their difference. This increases the search space of this initial guessing problem from 8 bits to 16 bits. However, once we find these round key bytes, we only need to search through 8 bits to find each of the remaining round key bytes.

3.4 Further Details of Our Attacks

In this subsection we explain some details of our attacks that are not mentioned above. To be more precise, we explain the overall attack strategy and how to exploit second round accesses.

We call all possible values that can be the correct value of a key byte (round key byte, respectively) as the hypothesis of that particular key byte (round key byte, resp.) or shortly key byte hypothesis (round key byte hypothesis, resp.). Incorrect values are called wrong hypothesis. Initially all of the 256 values, i.e. from 0x00 to 0xff, are considered as the key byte hypothesis for a particular key byte. During the course of the attack, we distinguish some of these values as wrong key byte hypothesis; thus decrease the number of hypothesis and increase that of wrong hypothesis.

In our attacks, we consider each access to a lookup table separately, starting from the second one. The first access is always a miss because of the cache cleaning and the assumptions explained above. We want to use the last round attack as an example to explain the overall attack strategy.

Outcome of the second access to T4 allows us to eliminate the wrong key hypothesis for RK_0 and RK_1. After we find the correct values for these bytes, we extend our attack considering the third access to find RK_2, then fourth access to find RK_3 and so on. Therefore, there are different steps in the attack and each further step considers one more access than the number of accesses considered in the previous step. Each step has a different set of wrong key hypothesis. It decreases the overall attack cost if we eliminate as many wrong key hypothesis in a step as possible before proceeding with the next attack step.

For example, the first step of the last round attack examines the outcomes of the first two accesses to T4 in each captured trace in the sample and eliminates all of the possible RK_0 and RK_1 values that are determined to be wrong. The second step considers the third access to T4 and the remaining hypothesis of RK_0 and RK_1 and eliminates all of the (RK_0, RK_1, RK_2) triples that cannot generate the captured traces. The attack continues with the later steps and only those key hypothesis that can generate the captured traces remain at the end. If we can capture a large enough sample then we end up with only the correct key. If we have less number of traces, then more than one hypothesis remain at the end of the attack and we need to have an exhaustive search on this reduced key set.

Eliminating as many wrong key hypothesis as possible in earlier steps reduces the cost of the later ones and therefore the total cost of this attack. We eliminate all of the key hypothesis that do not obey the captured trace in each step. In this sense, our decision strategy is optimal, because it eliminates maximum possible number of hypothesis.

The two round attack is slightly different than this scheme. There are four different lookup tables used in the first two rounds of AES. Therefore a single step of the two-round attack considers four more accesses than the previous step, i.e., the next unexamined access to each of the four tables. For example, the first step considers the first 8 accesses in the first round. These 8 accesses consist of two accesses to each of the four tables. The next step considers the first 12 accesses and so on.

We also want to give more details of the two-round attack, especially the second round attack, in this subsection. Using the guesses from the first round, a similar guessing procedure can be derived in the second round in order to obtain further key bits. We describe a possible attack that uses only accesses to T1, i.e., the second table. Recall that AES implementation we work on uses 5 different tables with 256 entries in each.

Let Δ_i represent $P_i \oplus K_i$. The index of the first access to T1 in the second round is:

$$Sbox(\Delta_4) \oplus 2 \bullet Sbox(\Delta_9) \oplus 3 \bullet Sbox(\Delta_{14}) \oplus Sbox(\Delta_3) \oplus Sbox(K_{14}) \oplus K_1 \oplus K_5 \ .$$

Here $Sbox(x)$ stands for the result of AES S-box lookup with the input value x and $\bullet$ is the finite field multiplication used in AES.

Using only the first 5 accesses to T1, i.e., up to fourth step of the two-round attack, and searching through K_3, K_4, K_9, and K_{14}, we can recover these four bytes. This guessing problem has a key space of 2^{32}. Notice that we can already recover $\langle K_1 \oplus K_5 \rangle$ in the first round attack.

The indices of the first accesses to each of the lookup tables in the second round are functions of different key bytes and these functions span each of the 16 key bytes. Hence, we can recover the entire key by analyzing only the outcomes of the first 5 accesses to each of the four tables, i.e., a total of 20 accesses.

Although knowing only the outcomes of the first 5 accesses is sufficient to recover the key, extending the attack by taking advantage of further accesses reduces the number of required traces. We want to mention that only the accesses of the first two rounds can be used in such a known-plaintext attack. The reason is the full avalanche effect. Starting from the third round, the indices become functions of the entire key, making an exhaustive search as efficient as our attack.

4 Analysis of the Attacks

In this section we estimate the number of traces need to be capture to recover the secret key. In other words, we determine the cost of the attacks presented above.

In the following subsections, we first present a computational model that allows us to determine the cost of trace-driven attacks and then we use this model to perform the cost analysis of the proposed attacks. The accuracy of our model has been verified experimentally.

4.1 Our Model

Let m be $2^{(8-\ell)}$, i.e. the number of blocks in a table. A block of elements of a lookup table that are stored together in a single cache line is defined as a block of this table. The cost of a trace-driven attack is a function of m. The two most common values of m are 16 and 32 today and thus we evaluate the cost of the attacks for these two values of m.

In order to calculate the expected number of traces, first we need to find an equation that gives us the expected number of table blocks that are loaded into the cache after the first k accesses. We denote this expected number as $\#_k$.

The probability of being a single table block not loaded into the cache after k accesses to this table is $(\frac{m-1}{m})^k$. The expected number of blocks that are not loaded becomes $m * (\frac{m-1}{m})^k$. Therefore,

$$\#_k = m - m * (\frac{m-1}{m})^k \ .$$

Let $R^k_{expected}$ be the expected fraction of the wrong key hypothesis that obeys the captured trace in k^{th} step of the attack. In other words, a wrong key hypothesis that generated the same trace with the correct key in the first k accesses of an encryption has a chance of generating the captured outcome in the next step with a probability of $R^k_{expected}$. Therefore, if the adversary captures the outcomes of the first $(k+1)$ accesses $(1 \leq k \leq 15)$ to T4 during a single encryption, she can eliminate $(1 - R^k_{expected})$ fraction

k	m=32		m=16	
	$R_{expected}$	$\#_k$	$R_{expected}$	$\#_k$
1	0.939453	1.000000	0.882813	1.000000
2	0.884523	1.968750	0.787140	1.937500
3	0.834806	2.907227	0.709919	2.816406
4	0.789923	3.816376	0.648487	3.640381
5	0.749522	4.697114	0.600528	4.412857
6	0.713273	5.550329	0.564035	5.137053
7	0.680868	6.376881	0.537265	5.815988
8	0.652021	7.177604	0.518709	6.452488
9	0.626464	7.953304	0.507063	7.049208
10	0.603946	8.704763	0.501197	7.608632
11	0.584236	9.432739	0.500138	8.133093
12	0.567116	10.137966	0.503050	8.624775
13	0.552384	10.821155	0.509209	9.085726
14	0.539850	11.482994	0.517999	9.517868
15	0.529340	12.124150	0.528890	9.923002

Figure 2: The calculated values of $\#_k$ and $R_{expected}$ for different values of m.

of the wrong key hypothesis in the k^{th} step of the attack, where

$$R^k_{expected} = \frac{\#_k}{m} * \frac{\#_k}{m} + (1 - \frac{\#_k}{m}) * (1 - \frac{\#_k}{m}) \ , 1 \leq k \leq 15 \ .$$

Notice that $R^k_{expected}$ is not the k^{th} power of a constant $R_{expected}$ here, but it is defined as a variable that is specified by the parameter k. The left (right) side of the above summation is the product of the probability of a cache hit (miss, resp.) and the expected ratio of the wrong hypothesis that remains after eliminating the ones that does not cause a hit (miss, resp.).

The following figure shows the approximations of $R^k_{expected}$ and $\#_k$ for different values of k and m. We want to mention again that these values are experimentally verified. The differences between the calculated and empirical values of $R^k_{expected}$ are less than 0.2% in average. We can use these values to find the expected number of remaining wrong key hypothesis after t measurements or the expected number of measurements to reduce the key space down to a specific number or in any such calculations.

4.2 Trade-off Between Online and Offline Cost

There is an obvious trade-off between online and offline cost of the attacks. If an adversary can capture a higher number of traces, it becomes easier to find the key. Eliminating more wrong hypothesis in early steps reduces the

m=16		m=32	
Number of traces	Cost ≈	Number of traces	Cost ≈
15	$2^{47.47}$	30	$2^{36.12}$
20	$2^{38.88}$	35	$2^{34.20}$
25	$2^{34.45}$	40	$2^{33.01}$
30	$2^{32.54}$	45	$2^{32.38}$
35	$2^{32.07}$	50	$2^{32.13}$
≥40	$< 2^{32.01}$	≥60	$< 2^{32.01}$

Figure 3: The cost analysis results of the two-round attack.

m=16		m=32	
Number of traces	Cost ≈	Number of traces	Cost ≈
1	$2^{117.74}$	1	$2^{120.97}$
5	$2^{74.93}$	5	$2^{90.39}$
10	$2^{39.57}$	10	$2^{59.32}$
20	$2^{26.17}$	20	$2^{36.21}$
30	$2^{22.68}$	30	$2^{29.25}$
40	$2^{20.70}$	40	$2^{25.92}$
50	$2^{19.02}$	50	$2^{23.95}$
75	$2^{16.45}$	75	$2^{21.13}$
≥100	$< 2^{16.10}$	≥200	$< 2^{16.10}$

Figure 4: The cost analysis results of the last round attack.

cost of the later steps. The change in the offline cost of the attacks with the number of captured traces can be seen in the following figures.

As shown in Figure 4, the last round attack requires only 5 measurements to reduce the computational effort of breaking the entire 128-bit key below the recommended minimum security levels (c.f. [10]). NSA and NIST recommends a minimum key length of 80 bits for symmetric ciphers so that the computational effort of an exhaustive search should not be lower than 2^{80}.

5 Experimental Details

We performed experiments to test the validity of the values we have presented above. The results show a very close correlation between our models and empirical results that confirms the validness of the models and calculations.

Bertoni et al. showed that the cache traces could be captured by measuring power consumption [7]. In our experimental setup, we did not measure the power consumption, instead we assumed the correctness of their argument.

We simply modified the AES source code of OpenSSL[20], which is arguably the most widely used open source cryptographic library. The purpose of our modifications was not to alter the execution flow of the cipher, but to store the values of the access indices. These index values were then used to generate the cache traces. This process allowes us to capture the traces and obtain the empirical results. The average difference between the empirical and calculated values of $R^k_{expected}$, i.e, the error rate, is less than 0.2%. We believe this shows enough accuracy to validate our model.

We generated one million randomly chosen cipherkeys and encrypted 100 random plaintext under each of these keys. In other words, we performed the last round attack steps with 100 random plaintext. After each encryption, we determined the ratio of the number of remaining wrong key hypothesis to the number of wrong key hypothesis that were present before the encryption. We call this ratio the reduction ratio, which is denoted as $R^k_{expected}$. We calculated the average of these measured values. Our results show very close correlation between the measured and calculated values. The calculated $R^k_{expected}$ values are given in Subsection 4.1.

6 Conclusion

We have presented trace-driven cache attacks on the most widely used software implementation of AES cryptosystem. We have also developed a mathematical model, accuracy of which is experimentally verified, to evaluate the cost of the proposed attacks. We have analyzed the cost using two different metrics, each of which represents the cost of a different phase of the attack.

Our analysis shows that such trace-driven attacks are very efficient and require very low number of encryptions to reveal the secret key of the cipher. To be more specific, an adversary can reduce the strength of 128-bit AES cipher below the recommended minimum security level by capturing the traces of only 5 encrpytions. Having more traces reduces the total cost of the attack significantly. Our results also show this trade-off between the online and offline cost of the attack in detail.

References

[1] O. Acıiçmez, W. Schindler, and Ç. K. Koç. Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations. Proceedings of the 12^{th} ACM Conference on Computer and Communications Security, pages 139-146, Alexandria, Virginia, November 7-11, 2005.

[2] O. Acıiçmez. "Remote Timing Attacks". Presentation given at Intel Corporation, Oregon, USA in December 2005. Available at: http://web.engr.oregonstate.edu/ aciicmez/osutass/

[3] Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[4] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). Cryptographic Hardware and Embedded Systems - CHES 2002, B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, pages 29-45, Springer-Verlag, LNCS Nr. 2523, 2003.

[5] D. J. Bernstein. Cache-timing attacks on AES. April, 2005. Available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[6] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks against AES. Cryptographic Hardware and Embedded Systems - CHES 2006, to appear.

[7] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. International Symposium on Information Technology: Coding and Computing - ITCC 2005, Volume 1, 4-6 April 2005, Las Vegas, Nevada, USA.

[8] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. Cryptology ePrint Archive, Report 2006/052, February 2006. Available at: http://eprint.iacr.org/2006/052

[9] D. Brumley and D. Boneh. Remote Timing Attacks are Practical. Proceedings of the 12^{th} Usenix Security Symposium, pages 1-14, 2003.

[10] Cryptographic Key Length Recommendation. Available at: http://www.keylength.com

[11] J. Daemen and V. Rijmen. "The Design of Rijndael". Springer-Verlag, 2002.

[12] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. Cryptographic Hardware and Embedded Systems - CHES 2001, Ç. K. Koç, D. Naccache, and C. Paar, editors, pages 251-261, Springer-Verlag, LNCS Nr. 2162, 2001.

[13] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, vol.8, pages 141-158, 2000.

[14] P. C. Kocher. Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems. Advances in Cryptology - CRYPTO '96, N. Koblitz, editors, pages 104-113, Springer-Verlag, LNCS Nr. 1109, 1996.

[15] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. Advances in Cryptology - CRYPTO '99, M. Wiener, editors, pages 388-397, Springer-Verlag, LNCS Nr. 1666, 1999.

[16] C. Lauradoux. Collision attacks on processors with cache and countermeasures. Western European Workshop on Research in Cryptology - WEWoRC 2005, C. Wolf, S. Lucks, and P.-W. Yau, editors, pages 76-85, 2005.

[17] M. Neve, J.-P. Seifert, and Z. Wang. A refined look at Bernstein's AES side-channel analysis. Proceedings of ACM Symposium on Information, Computer and Communications Security - ASIACCS'06, Taipei, Taiwan, March 21-24, 2006.

[18] M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. Selected Areas of Cryptography - SAC'06, to appear.

[19] M. Neve. Cache Vulnerabilities and SPAM analysis. PhD thesis, July 2006.

[20] OpenSSL Project. Openssl. http://www.openssl.org

[21] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: the Case of AES. August, 2005. Available at: http://eprint.iacr.org/2005/271

[22] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, D. Pointcheval, editor, pages 1-20, Springer-Verlag, LNCS Nr. 3860, 2006.

[23] 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.

[24] D. Page. Defending Against Cache Based Side-Channel Attacks. Technical Report. Department of Computer Science, University of Bristol, 2003.

[25] D. Page. Partitioned Cache Architecture as a Side Channel Defence Mechanism. Cryptography ePrint Archive, Report 2005/280, August 2005. Available at: http://eprint.iacr.org/2005/280

[26] C. Percival. Cache missing for fun and profit. BSDCan 2005, Ottawa, 2005. Available at: http://www.daemonology.net/hyperthreading-considered-harmful/

[27] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. Cryptographic Hardware and Embedded Systems - CHES 2003, C. D. Walter, Ç. K. Koç, and C. Paar, editors, pages 62-76, Springer-Verlag, LNCS Nr. 2779, 2003.

[28] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. ISITA 2002, 2002.

[29] Y. Tsunoo, E. Tsujihara, M. Shigeri, H. Kubo, and K. Minematsu. Improving cache attacks by considering cipher structure. International Journal of Information Security, February 2006.