Trace Driven Cache Attack 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 a trace-driven cache attack on a widely used implementation of the AES cryptosystem. We also evaluate the cost of the proposed attack in detail under the assumption of a noiseless environment. We develop an accurate mathematical model that we use in the cost analysis of our attack. 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.

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 [10, 8, 11, 3, 1]. In this paper, we focus on a type of side-channel cryptanalysis that takes advantage of the information, which 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 Kelsey et al. in [9, 10]. D. Page described and simulated a theoretical cache attack on DES [14]. Actual cache based timing attacks were implemented by Tsunoo et al. [15, 16]. The main focus of those papers was on DES[6]. Tsunoo et al. claimed that the search space of the AES key could be narrowed to 32 bits using cache based timing attacks; however they did not give the details of such an attack.

[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`

Bernstein showed the vulnerability of AES by performing a cache based timing attack on OpenSSL's AES implementation [4]. Efficient cache based timing attacks on AES were presented by Osvik et al. in [13]. They described and simulated several different methods to perform local cache attacks. They made use of a local array and exploited the collisions between the table lookups and the access operations to this array.

Theoretical cache attacks were first described by Page in [14]. Page characterized two types of cache attacks, namely trace-driven and time-driven. In trace-driven attacks, 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 the cache hits and misses. Therefore, the adversary has the ability to observe if the second 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. Adversary is assumed to be able to observe the aggregate profile, i.e., total numbers of cache hits and misses or 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. Time-driven attacks are based on statistical inferences, and therefore require much higher number of samples than trace-driven attacks.

In this paper, we will focus only on trace driven attacks. The trace of an encryption can be captured by measuring power consumption during the encryption as shown in [5]. We will not give details of how to capture these traces, but assume that an adversary is able to do it. We will analyze a trace-driven attack on AES under the assumption that the adversary can capture the traces of AES encryption during the first two rounds. 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 and 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, these two phases may overlap and may be performed simultaneously.

We will present and analyze a trace driven attack on AES below. We will use two different metrics to evaluate the cost of our attack. The first metric is ıthe 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 ıthe 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 attack. 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 will see in this paper, there is a trade-off between the costs of these two phases.

2 The Basics of a Trace Driven 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.[3] 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.

Cryptosystems have data-dependent memory access patterns. Cache architectures leak information about the cache hit/miss statistics of ciphers through side channels, e.g., execution time and power consumption. Therefore, it is possible to exploit cache behavior of a cipher to obtain information about its memory access patterns, i.e. indices of S-box and table lookups.

Cache attacks rely on the cache hits and misses that occur during the encryption or decryption process of the cryptosystem. Even if the same

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

instructions are executed for any particular (plaintext, cipherkey) pair, 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.

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. In this paper, 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 or M represents a cache hit or miss respectively.

The trace of an encryption can be captured by the use of power consumption measurements. Bertoni et al. performed a cache based power attack on AES[5]. This attack is not indeed a trace driven cache attack. Instead of capturing the traces, Bertoni et al. determined the particular cache lines, content of which were overwritten by AES table lookups. Furthermore, they could determine exactly which table access overwrites which particular cache line.

3 Trace Driven Cache Attack on the AES

In this paper, we present a trace driven attack on a widely used implementation of AES [2], and calculate its cost. We assume that the cache does not contain any AES data prior to each encryption, because the captured traces will not be accurate otherwise. Therefore, the adversary needs to clean the cache (e.g., by loading some garbage data) before the encryption process starts.

The implementation we will analyze is described in [7] 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. Before the first round, there is a round key addition, which adds the cipherkey to the state that has the actual plaintext. In other words, the input to the first round is the bitwise addition of the plaintext and the cipherkey. We want to mention that only 128-bit block and key sizes will be considered. Although the idea we will present below can be extended to cases of larger block and key sizes, we will omit these cases due to space limitations.

Our attack consists of two parts. In the first part, we analyze the outcomes of the accesses of the first round. The second part takes also the second round accesses into consideration. We will describe both parts in the following subsections.

3.1 First Round Attack

The AES implementation described in [7] uses 5 tables: T0, T1, T2, T3, and T4; each has 256 many 32-bit values. In each round, except the last one, it makes 4 references to each of the first 4 tables. 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 gives information about K_0 and K_4. For example, if the second access results a cache hit, we can directly conclude that the indices $P_0 \oplus K_0$ and $P_4 \oplus K_4$ are equal in case of a cache line contains a single element of T0. 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 $K_0 \oplus K_4$.

However, in a real environment, 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 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.

We can also find the other key byte differences $\epsilon_{i,j} = \langle K_i \oplus K_j \rangle$, 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 $\langle K_i \oplus K_{4*j+i} \rangle$, where i,j $\in$ {0,1,2,3}, and it is enough to find the entire 128-bit key by searching only 32 bits in case ℓ is zero. Recall 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. ℓ can be as low as zero bits, in which case the search space becomes only 32 bits. For $\ell = 4$ the search space is reduced by 48 bits yielding an 80 bit problem.

We want to emphasize that a length of 32 bits is a theoretical lower bound for the exhaustive search space. The realistic space length depends on the actual cache architecture, i.e., ℓ value, and is either 68, or 80 for 128-bit keys on most widely used processors.

In our attack, we consider each access separately, starting from the second one. The first access is always a miss because of the cache cleaning as explained above. Outcome of the second access to , e.g., T0 allows us to eliminate the wrong key byte differences for $\langle K_0 \oplus K_4 \rangle$. After we find the correct values for $\langle K_i \oplus K_{4+i} \rangle$, where $i \in \{0,1,2,3\}$, we extend our attack considering the third accesses, 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. 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.

3.2 Second Round Attack

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 sample attack that uses only accesses to T0, i.e., the first 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 T0 in the second round is:

$$2 \bullet s(\Delta_0) \oplus 3 \bullet s(\Delta_5) \oplus s(\Delta_{10}) \oplus s(\Delta_{15}) \oplus s(K_{13}) \oplus K_0 \oplus 01 \ .$$

Here s(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.

Therefore, only using the first 5 accesses to T0, i.e., up to fourth step of the attack, and searching through 5 key bytes, K_0, K_5, K_{10}, K_{15}, and K_{13}, we can recover the values of these bytes and the most significant parts of all of the other key bytes. This guessing problem has a key space of $2^{40-\ell}$, at least one being the correct value. If there is an x value such that $\langle s(x) \rangle = \langle s(K_{13}) \rangle$, $\langle x \rangle = \langle K_{13} \rangle$, and $x \neq K_{13}$, then the key space can only be reduced to the number of different such x values plus one. However, we will ignore this detail since the maximum number of different x values is small. Moreover, when we extend the attack considering further access indices, we can eliminate all of these x values.

Notice that we can already recover $\langle K_5 \oplus K_{13} \rangle$ by applying the first round attack on T1. The indices of the first accesses to each of the lookup tables are functions of different key bytes in the second round and these functions covers each 16 key bytes. Hence, we can recover the entire key using only the outcomes of the first 5 accesses to each of the four tables.

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 this attack. The reason is the number of key bytes that determines the indices of the later round's accesses. Starting from the third round, the indices become functions of the entire key, making an exhaustive search as efficient as our attack.

The second access to T0 in the second round is:

$$2 \bullet s(\Delta_4) \oplus 3 \bullet s(\Delta_9) \oplus s(\Delta_{14}) \oplus s(\Delta_3) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus 01 \ .$$

For a particular combination K_0, K_5, K_{10}, K_{15}, and K_{13} values, we can search the lower bits of K_3, K_4, K_9, K_{14} in order to reveal four more key bytes in the fifth step of the attack. The size of search space in the fifth step is $2^{(4*\ell)}$. Applying all seven steps of this attack to accesses to a single table is sufficient to recover the entire key. However, we can reduce the necessary number of traces if we consider each of the four tables and merging the final results that come from each single table. The following table shows the search space size in each step for each table. Recall that we assume to know the results of the first round attack.

Step	T0 and T2	T1 and T3
1-3	$2^{(8-\ell)}$	
4	$2^{(32+\ell)}$	2^{32}
5	$2^{(4*\ell)}$	$2^{(4*\ell)}$
6	$2^{(3*\ell)}$	$2^{(4*\ell)}$
7	$2^{(4*\ell)}$	$2^{(4*\ell)}$

4 The Cost of the Attack

In this section we will calculate the number of traces we need to capture to recover the secret key or reduce the exhaustive search space as much as possible. In other words, we will determine the cost of the attack presented above.

In the following subsections, we will present the cost results of both parts of the attack. The online cost, i.e. average number of necessary traces, of the first round attack was determined empirically. However, we did not determine that of the second round via the same technique. The main reason is the large size of the search space. Therefore, we will develop an accurate model of the attack and calculate the cost based on it. We verified the accuracy of our model experimentally. We performed our experiments under the setup explained in the next section.

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 in a single cache line together is defined as a block of this table. The two most common values of m are 16 and 32 today. We will evaluate the cost of the attack for these two values of m.

4.1 First Round Attack

The following table shows the average number of traces, denoted as E, that are needed to be captured in order to reveal the key byte differences as explained in the previous section. The bottleneck of the first round attack is the first step. If the adversary collects enough number of traces to successfully apply the first step, this number will most likely be sufficient to obtain $\langle K_i \oplus K_{4*j+i} \rangle$, for $i,j \in \{0,1,2,3\}$.

In the following subsections, we will assume that 31 and 15 traces are sufficient to find all of these key byte differences for $m = 32$ and $m = 16$, respectively.

	E	
Step	$m = 32$	$m = 16$
1	31.0	15.0
2	20.1	10.0
3	15.2	7.9

4.2 Second Round Attack

In this subsection, we will develop an estimation model to determine the cost of the second round attack. We want to mention that this model, as well as the previous results, is experimentally verified. The difference between the calculated and empirical values of the second round attack is less than 0.4% in average. We present the calculated values, which are the core of the model, in a table at the end of this subsection.

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 is $(\frac{m-1}{m})^k$. The expected number of blocks that are not loaded becomes $m * (\frac{m-1}{m})^k$. Therefore,

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

If the adversary captures the outcome of the first k accesses ($5 \leq k \leq 8$) to a lookup table during a single encryption, she can eliminate $(1 - R^k_{expected})$

fraction of the wrong key hypothesis in the $(k-1)^{th}$ step of the attack, where

$$R^k_{expected} = \frac{\#_{(k-1)}}{m} * \frac{\#_{(k-1)}}{m} + (1 - \frac{\#_{(k-1)}}{m}) * (1 - \frac{\#_{(k-1)}}{m}) \ , 5 \leq k \leq 8 \ .$$

Notice that $R^k_{expected}$ is not 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 table presents the values of $R^k_{expected}$ and $\#_k$ for different values of k and m. We want to mention again that these values are experimentally verified. 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. For example, we can calculate the expected number of remaining wrong key hypothesis in fourth step of the attack on T0 after $t = 20$ measurements for $m = 16$ as $(0.648487)^{20} * (2^{36} - 1) \approx 2^{23.5}$. Assume that we want to reduce the wrong hypothesis space to $x = 0.1$ many wrong hypothesis in the fifth step of the attack on T0 for given K_0, K_5, K_{10}, K_{15}, and K_{13} values when $m = 32$. The expected number of measurements for this is

$$E = \text{Expected number of measurements} = log_{(R_{expected})}(\frac{x}{2^{(4*\ell)} - 1})$$

$$= log_{(R_{expected})}(x) - log_{(R_{expected})}(2^{12} - 1) = 36.84 \ ,$$

where $R_{expected} = 0.749522$ for $k = 6$ and $\ell = 3$.

Step	k	m=32		m=16	
		$R_{expected}$	$\#_k$	$R_{expected}$	$\#_k$
4	5	0.789923	3.816376	0.648487	3.640381
5	6	0.749522	4.697114	0.600528	4.412857
6	7	0.713273	5.550329	0.564035	5.137053
7	8	0.680868	6.376881	0.537265	5.815988

4.3 Online versus Offline Cost of the Attack

The attack on a single table gives us a subset of the entire key space, which we call initial subset, and the correct key value always belongs to this subset. The entire attack consists of single attacks on each of the four lookup tables. After collecting these 4 initial subsets and finding the intersection of them, the final subset becomes the result of the entire attack. We give the expected size of these subsets for different number of captured traces in the following tables.

Approximate Size of the Initial Subsets					
Number of traces	m=16		Number of traces	m=32	
	T0 and T2	T1 and T3		T0 and T2	T1 and T3
15	$2^{35.2}$	$2^{34.1}$	31	$2^{25.1}$	$2^{22.3}$
16	$2^{32.9}$	$2^{31.3}$	32	$2^{24.6}$	$2^{21.7}$
17	$2^{30.7}$	$2^{28.7}$	33	$2^{24.2}$	$2^{21.3}$
18	$2^{28.8}$	$2^{26.3}$	34	$2^{23.7}$	$2^{20.8}$
19	$2^{27.1}$	$2^{24.2}$	35	$2^{23.3}$	$2^{20.4}$

Heuristic approaches show that 15 and 31 traces are sufficient to reveal the entire key for $m = 16$ and $m = 32$, respectively. The number of remaining wrong key hypothesis becomes less than 10^{-60} for each case. However, the offline cost of the attack, i.e., the number of operations in the analysis phase, is reduced when we increase the number of traces. Eliminating more wrong hypothesis in early steps reduces the cost of the later steps. The change of the offline cost with the number of traces can be seen in the following table.

m=16		m=32	
Number of traces	Cost ≈	Number of traces	Cost ≈
15	$2^{50.1}$	31	$2^{39.3}$
16	$2^{48.4}$	32	$2^{39.0}$
17	$2^{46.8}$	33	$2^{38.6}$
18	$2^{45.4}$	34	$2^{38.3}$
19	$2^{44.1}$	35	$2^{38.0}$
20	$2^{43.0}$	36	$2^{37.8}$
22	$2^{41.1}$	38	$2^{37.3}$
24	$2^{39.7}$	40	$2^{37.0}$
26	$2^{38.6}$	42	$2^{36.7}$
28	$2^{37.9}$	44	$2^{36.5}$
30	$2^{37.5}$	46	$2^{36.4}$
$\geq$35	$< 2^{37.1}$	$\geq$50	$< 2^{36.2}$

5 Experimental Details

We performed experiments to test the validity of the values we 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 [5]. 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[12], 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 allowed us to capture the traces and obtain the empirical results presented below.

Due to the lack of space, we cannot present the complete experimental results in this paper. Therefore, we restrict ourselves only to indicate the difference between the calculated and empirical values of the second round attack, which is less than 0.4% in average. We believe this shows enough accuracy to validate our model.

The online cost of the first round attack was determined empirically by applying the attack on one million randomly chosen cipherkeys. For each of these keys, we performed the attack by encrypting random plaintext and eliminating wrong key hypothesis by capturing the traces of the encryption. We determined the number of traces/plaintext that were processed to eliminate all of the wrong key hypothesis for each of these random keys. The results presented in Subsection 4.1 are the averages of these numbers.

We did not use the same technique for the second round attack due to the large size of the search space. But, we developed an accurate model of the attack and calculated the cost based on it. We verified the accuracy of our model experimentally. Again, we generated one million randomly chosen cipherkeys and encrypted 20 many random plaintext under each of these keys. In other words, we performed the second round steps with 20 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 $R_{expected}$. We calculated the average of these measured values. Our results show very close correlation between the measured and calculated values. The calculated $R_{expected}$ values are given in Subsection 4.2. In our experiments, we performed all second round guessing problems with only 2^4 different key hypotheses for each key byte, one of them being the correct value. Our intention was to validate the general principle but to save many operations.

6 Conclusion

We have presented a trace driven cache attack on a widely used implementation of AES cryptosystem. We have also developed a mathematical model, accuracy of which is experimentally verified, to evaluate the cost of the proposed attack. 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 break the cipher using around

2^{50} operations by capturing only 15 encryption traces. 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] Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001, available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[3] 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.

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

[5] G. Bertoni, L. Breveglieri, M. Monchiero, G. Palermo, and V. Zaccaria. AES Power Attack Based on Induced Cache Miss and Countermeasure. International Conference on Information Technology: Coding and Computing - ITCC05, IEEE Computer Society, 2005.

[6] R. H. Brown, M. L. Good, and A. Prabhakar. Data Encryption Standard (DES) (FIPS 46-2). Federal Information Processing Standards Publication (FIPS), Dec 1993. http://www.itl.nist.gov/fipspubs/fip46-2.html (initial version from Jan 15, 1977).

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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

[14] 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.

[15] 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.

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