# Trace-Driven Cache Attacks on AES (Short Paper)

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

[1] Oregon State University, School of EECS
Corvallis, OR 97331, USA
[2] Information Security Research Center, Istanbul Commerce University
Eminönü, Istanbul 34112, Turkey
aciicmez@eecs.oregonstate.edu, koc@cryptocode.net

**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 an efficient 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.

**Keywords:** Side-channel Analysis, cache attacks, trace-driven attacks, AES.

## 1 Introduction

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 a trace-driven cache based attack on AES in this paper. There are already two trace-driven attacks on AES in the literature [5,12]. However, our attack requires significantly less number of measurements (e.g. only 5 measurements in some cases) and is 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 a single attack on AES in this paper, 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.

P. Ning, S. Qing, and N. Li (Eds.): ICICS 2006, LNCS 4307, pp. 112–121, 2006.

## 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 [10,11]. D. Page described and simulated a theoretical cache attack on DES [16]. Actual cache based timing attacks were implemented by Tsunoo et al. [18,19]. The original attack on MISTY1 proposed in [19] has recently been improved in [20].

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 [4]. 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 [13] that this is only a fallacy. They described the circumstances in which the attack might work and also the limitations of the Bernstein attack. A realistic general remote cache attack was developed by Acıiçmez et al [3].

Osvik et al. described various local cache attack variants in [15]. 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 these attacks by taking the last AES round into consideration [14]. The same idea of exploiting collisions between two different processes was also used by Colin Percival in [17] to develop an 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 [18] and [19]. The attacks presented in [3,12,6] are also based on internal collisions.

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 [16]. Access-driven attacks are relatively new and first seen in [15]. 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 [12,5]. Bertoni et al. implemented a cache based power attack that exploits external collisions between different processes [5]. Their attack requires 256 power traces to reveal the secret AES key. Lauradoux's power attack exploits the internal collisions inside the cipher but only considers the first round AES accesses and can reduce the exhaustive search space of a 128-bit AES key to 80 bits.

We described much more efficient trace-driven attacks on AES in [2]. 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. Due to the space limitation, we only present our last round attack in this paper.

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 a miss.

The trace of an encryption can be captured by the use of power consumption measurements as done in [5,12]. 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 may increase depending on the amplitude of the noise. However, an analysis under the above assumption gives us a more clear understanding of the attack cost. Assumption of a noiseless environment also enables us to make more reliable comparison of different attacks.

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. [7,1]), these two phases may overlap and may be performed simultaneously.

We use two different metrics to evaluate the cost of our last round attack 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 attack. 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 a trace-driven attack on the most widely used implementation of AES, and estimate its cost. 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 [5,19,18,15,17]) 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 attack 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 [9] 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.

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. Four repetations of the same 8-bit long Sbox element are concatenated to each other to form the corresponding 32-bit long element of T4. There are 16 accesses to T4 in that round. The indices of these accesses are $S_w^{10}$, where $S_w^t$ is the byte $w$ of the

intermediate state value that becomes the input of round $t$ and $w \in \{0, .., 15\}$. 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:

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

where $RK_i^{10}$ is the $i^{th}$ byte of the last round key, $Sbox[I_i]$ is the S-box output for the input index $I_i$, and $I_i = S_w^{10}$ for known $w, i \in \{0, 1, ..., 15\}$.

$I_i$ is equal to $S_w^{10}$ for known values of $i$ and $w$, but the actual mapping between these variables is not relevant for our purposes. In this paper, we present our attack under the assumption that the AES memory accesses in the last round are issued by the processor in a given order, i.e., first T4[$I_0$], second T4[$I_1$], etc. However, the actual order is implementation specific and may differ from our assumption. Our attack can easily be adapted to any given order without any performance loss. We also need to mention that the S-box in AES implements a permutation, and therefore its inverse, i.e. $Sbox^{-1}$, exists.

The outcomes of the last round accesses to T4 leak 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 $I_0$ and $I_1$ 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 $I_i$ in terms of $RK_i^{10}$ and $c_i$:

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

If $I_0$ and $I_1$ 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 $RK_0^{10} \oplus RK_1^{10}$.

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, whenever an access retrieves an element, which lies in the same cache line of the previously accessed data, a cache hit occurs.

Let $\delta$ be the number of bytes in a cache line and assume that each element of the table is $k$ bytes long. Under this situation, there are $\delta/k$ elements in each line, which means any access to a specific element will map to the same line with $(\delta/k - 1)$ different other accesses. If two different accesses to the same array read

the same cache line, the most significant parts of their indices, i.e., all of the bits except the last $\ell = \log_2(\delta/k)$ bits, must be identical.

Therefore, we observe a cache hit in the second access to T4 whenever

$$\langle I_0 \rangle = \langle I_1 \rangle \ ,$$

and so

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

where $\langle A \rangle$ stands for the most significant part of A. However due to the non-linearity 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 can 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.

The value of $RK_2^{10}$ can also be determined by analyzing the first three accesses to T4 after the correct values of $RK_0^{10}$ and $RK_1^{10}$ are 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. 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.

We want to explain some details of our attack that are not mentioned above. We call all possible values that can be the correct value of a round key byte as the hypotheses of that particular round key byte or shortly round key byte hypotheses. Incorrect values are called wrong hypotheses. Initially all of the 256 possible values are considered as the round key byte hypotheses for a particular round key byte. During the course of the attack, we distinguish some of these values as wrong hypotheses; thus decrease the number of hypotheses and increase that of wrong hypotheses.

In our attack, we consider each access to T4 separately, starting from the second one. The first access is always a miss because of the cache cleaning and the assumptions explained above. We start a search on all possible hypotheses of ($RK_0$,$RK_1$) pair by assigning $RK_0 = x$, $RK_1 = y$ and checking whether ($x$,$y$) obeys the captured traces, $x, y \in \{0, ..., 255\}$. We then eliminate the wrong hypotheses those do not obey the traces. Then we extend our focus to the third access and perform a similar search on (($RK_0$,$RK_1$),$RK_2$). Again we eliminate the wrong hypotheses of $RK_2$ for each remaining ($RK_0$,$RK_1$) pairs and end up only with ($RK_0$,$RK_1$,$RK_2$) values those obey the traces. We continue this method by considering fourth access and so on. After we determine all of the round key hypotheses actually obeying the traces, we perform an exhaustive search on the final remaining set. The above method is the same as AC-3 algorithm, which is an optimal propogation algorithm for binary constraints just like our case.

## 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 attack presented above. 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 attack.

### 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 hypotheses that obeys a 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 for the next access 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 of the wrong key hypotheses 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 hypotheses that remain after eliminating the ones that do not cause a hit (miss, resp.).

Figure.1 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 hypotheses after $t$ measurements or the expected number of measurements to reduce the key space down to a specific number or in any such calculations.

| k | m=32 | | m=16 | |
|---|---|---|---|---|
| | $R^k_{expected}$ | $\#_k$ | $R^k_{expected}$ | $\#_k$ |
| 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 |

**Fig. 1.** The calculated values of $\#_k$ and $R_{expected}$ for different values of $m$

### 4.2 Trade-Off Between Online and Offline Cost

There is an obvious trade-off between online and offline cost of the attack. If an adversary can capture a higher number of traces, it becomes easier to find the key. Eliminating more wrong hypotheses in early steps reduces the cost of the later steps. The change in the offline cost of the attack with the number of captured traces can be seen in Figure.2.

As shown in the figure, 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. [8]). 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 model and empirical results and confirm the validness of our model 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. The purpose of our modifications was not to alter the execution 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 allows us to capture the traces and obtain the empirical results.

We generated one million randomly chosen cipherkeys and encrypted 100 random plaintext under each of these keys. In other words, we performed the

| m=16 | | m=32 | |
|---|---|---|---|
| Number of traces | Cost ≈ | Number of traces | Cost ≈ |
| 1 | $2^{117.68}$ | 1 | $2^{120.93}$ |
| 5 | $2^{74.51}$ | 5 | $2^{90.76}$ |
| 10 | $2^{35.12}$ | 10 | $2^{56.16}$ |
| 20 | $2^{24.22}$ | 20 | $2^{33.97}$ |
| 30 | $2^{21.36}$ | 30 | $2^{27.77}$ |
| 40 | $2^{20.08}$ | 40 | $2^{24.88}$ |
| 50 | $2^{19.46}$ | 50 | $2^{23.25}$ |
| 75 | $2^{19.13}$ | 75 | $2^{21.22}$ |
| 100 | $2^{19.12}$ | 100 | $2^{20.39}$ |

**Fig. 2.** The cost analysis results of the last round attack

attack steps with 100 random plaintext. After each encryption, we determined the ratio of the number of remaining wrong key hypotheses to the number of wrong key hypotheses 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 average difference between the empirical and calculated values of $R^k_{expected}$, i.e, the error rate, is less than 0.2%. The calculated $R^k_{expected}$ values are given in Subsection 4.1.

## 6 Conclusion

We have presented a trace-driven cache attack 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 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 reduce the strength of 128-bit AES cipher below the recommended minimum security level by capturing the traces of only 5 encryptions. 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 and Ç. K. Koç. Trace-Driven Cache Attacks on AES. Cryptology ePrint Archive, Report 2006/138, 2006.

3. O. Acıiçmez, W. Schindler, and Ç. K. Koç. Cache Based Remote Timing Attack on the AES, Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, to appear.
4. D. J. Bernstein. Cache-timing attacks on AES. April, 2005. Available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf
5. 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.
6. J. Bonneau and I. Mironov. Cache-Collision Timing Attacks against AES. Cryptographic Hardware and Embedded Systems - CHES 2006, to appear.
7. D. Brumley and D. Boneh. Remote Timing Attacks are Practical. Proceedings of the $12^{th}$ Usenix Security Symposium, pages 1-14, 2003.
8. Cryptographic Key Length Recommendation. Available at: http://www.keylength.com
9. J. Daemen and V. Rijmen. "The Design of Rijndael". Springer-Verlag, 2002.
10. 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.
11. 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.
12. 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.
13. 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.
14. M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. Selected Areas of Cryptography - SAC'06, to appear.
15. 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.
16. 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.
17. C. Percival. Cache missing for fun and profit. BSDCan 2005, Ottawa, 2005. Available at: http://www.daemonology.net/hyperthreading-considered-harmful/
18. 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.
19. Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. ISITA 2002, 2002.
20. 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.