Cryptanalysis of DES Implemented on Computers with Cache

Yukiyasu Tsunoo[1], Teruo Saito[2], Tomoyasu Suzaki[2], Maki Shigeri[2], and Hiroshi Miyauchi[1]

[1] NEC Corporation, Internet Systems Research Laboratories
4-1-1, Miyazaki, Miyamae-ku, Kawasaki, Kanagawa 216-8555, Japan
{tsunoo@bl, h-miyauchi@bc}.jp.nec.com
[2] NEC Software Hokuriku Ltd.
1, Anyoji, Tsurugi, Ishikawa 920-2141, Japan
{t-saito@qh, t-suzaki@pd, m-shigeri@pb}.jp.nec.com

Abstract. This paper presents the results of applying an attack against the Data Encryption Standard (DES) implemented in some applications, using side-channel information based on CPU delay as proposed in [11]. This cryptanalysis technique uses side-channel information on encryption processing to select and collect effective plaintexts for cryptanalysis, and infers the information on the expanded key from the collected plaintexts. On applying this attack, we found that the cipher can be broken with 2^{23} known plaintexts and 2^{24} calculations at a success rate $> 90\%$, using a personal computer with 600-MHz Pentium III.
We discuss the feasibility of cache attack on ciphers that need many S-box look-ups, through reviewing the results of our experimental attacks on the block ciphers excluding DES, such as AES.

Keywords: DES, AES, Camellia, cache, side-channel, timing attacks

1 Introduction

Recently, many proposals have been made for cryptanalysis techniques to measure physical information from a cryptographic device. These techniques are called "side-channel attacks." Typical examples are Differential Power Analysis [5], which measures the variation in power consumption caused by a cryptographic device, and Differential Fault Analysis [1], which causes some sorts of physically erroneous operation to occur in a cryptographic device and then measures resulting phenomena. Because techniques of this kind are mainly used for attacking cryptographic systems implemented on smart cards, anti-tampering measures e.g. adding noise to consumed power have been considered. "Timing attacks" [2][6] that measure the encryption time of a cryptographic application can also be treated as side-channel attacks. A countermeasure to attacks of this type is to eliminate branch processing in the implementing algorithm so that encryption times are equivalent.

Previously proposed timing attacks make use of the fact that conditional branches that occur during encryption processing cause variations in encryption

C.D. Walter et al. (Eds.): CHES 2003, LNCS 2779, pp. 62–76, 2003.

time. CPU cache misses, however, can also cause such variations. In this regard, most of the recent computers employ a "CPU cache", abbreviated simple to a "cache" from here on, between the CPU and main memory, since this type of hierarchical structure can speed program run-time on the average. If, however, the CPU accesses data that were not stored in the cache, i.e. if a cache miss occurs, a delay will be generated, as the target data must be loaded from main memory into the cache. The measurement of this delay may enable attackers to determine the occurrence and frequency of cache misses.

With the above in mind, we have focused our attention on data-access processing, i.e. the operations of the S-box commonly used by encryption algorithms, and have developed a new attack technique to infer the information on S-box input from the variations in encryption time for different plaintexts. This is classified as a side-channel attack on software-implemented ciphers, and it has already broken MISTY1 [11] successfully. It does not require specialized measuring equipment; the cipher can be broken in a relatively short time using a personal computer, if the encryption module of the cipher is available. Though Kelsey et al. described the feasibility of a cache-based attack on ciphers using a large S-box e.g. Blowfish [4], they did not refer a specific method. The first application of an attack using a cache is described in [11].

We made experimental attacks on some block ciphers including Data Encryption Standard (DES). This paper describes the cases we could break the cipher in spite of frequent S-box look-ups, or the resistance to the attack described in [11].

This paper is organized as follows. Section 2 describes the basics of the proposed attack. Section 3 then describes the method of applying this attack to DES and presents the results of our experiment. Section 4 shows the results of this attack on AES and Camellia. Lastly, section 5 concludes the paper.

2 The Basics of Attack

2.1 Cache Operation

A cache is a form of memory that allows faster reading and writing of data than those in a main memory. It is located between the CPU and main memory. When reading data from main memory, the CPU first checks the cache, and if the target data is present, it reads the data from the cache. Finding data in the cache in this way is called a "cache hit," while not finding data in the cache and reading it from main memory is called a "cache miss." In the latter case, the data read from main memory is also written to the cache [1], so that any subsequent reading of this data might speed up. In short, a delay in processing will occur even for the same instruction if target data does not exist in the cache, and this delay will appear as a variation in the program execution time.

[1] In reality, values near the referenced one will also be loaded into the cache.

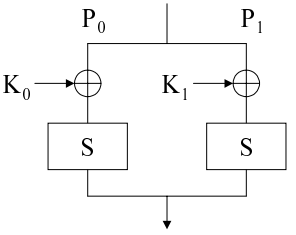


Fig. 1. Cipher With Two S-boxes

2.2 The Encryption Time and S-Box Operation

As described in Section 2.1, plaintext with the long encryption time should correspond to the frequency of cache miss. In the following, we examine the conditions for the generation of cache misses in the encryption process.

In the encryption processing, data access occurs when the S-box is referenced. What then are the conditions that would generate more cache misses when referencing the S-box? Consider that a cache miss occurs when first referencing the S-box and that the data in question is therefore loaded into the cache as described earlier. Now, when next referencing the S-box, if the S-box input value is the same as the already referenced value or its nearby one [2], data referencing can be done by accessing the cache; a cache miss does not occur. If, however, a value excluding already referenced ones and their nearby ones is referenced, the desired data will not be found in the cache and will have to be loaded from main memory; cache miss occurs. Accordingly, when making multiple S-box references during the encryption process, the number of cache misses increases proportionally with the number of different S-box input values.

Based on the above reasoning, the encryption time should be long if there are many different data referenced by the S-box during encryption. Thus, the measurement of the encryption time for a plaintext makes it possible to determine whether that plaintext is of the type that generates many cache misses in encryption (i.e., plaintext for which there are many different S-box input values).

2.3 Attack Model

The cipher with two S-boxes shown in Fig. 1 is used to explain the basics of the process of obtaining information on keys, which exploits side-channel information. The structure shown in the figure employs independent keys K_0 and K_1 in different S-boxes.

Referring to Fig. 1, we assume that the relationship between the input values of the two S-boxes under comparison is understood. The key differential value

[2] Values near the referenced value will be simultaneously loaded due to the characteristics of CPU.

$K_0 \oplus K_1$ (referred to below as "key difference") can therefore be inferred from the values of plaintext P_0 and P_1, using either of the following relations.

$$P_0 \oplus K_0 = P_1 \oplus K_1 \rightarrow P_0 \oplus P_1 = K_0 \oplus K_1 \quad (1)$$

$$P_0 \oplus K_0 \neq P_1 \oplus K_1 \rightarrow P_0 \oplus P_1 \neq K_0 \oplus K_1 \quad (2)$$

In other words, if the plaintexts for which S-box input values are frequently the same or frequently different are collected by measuring the encryption time, the information on the key differences can be obtained from those plaintexts. The attack comprised of 2 processes, the one for obtaining the key differences and the one for collecting cache timing data described in Section 3.2 is called a "cache attack." Obtaining key differences by a cache attack can reduce the key search space.

As the structure shown in Fig. 1 can be found in many block ciphers, it is thought that cache attacks can be widely applicable to ciphers of this type.

2.4 Non-elimination/Elimination Table Method

As described above, a correlation exists between the encryption time and the relationship between input values of separate S-boxes. We consider the following two methods of obtaining a key difference, based on such information.

The first method corresponds to the situation in which the input values of S-boxes under comparison are equivalent. In this case, Eq. (1) holds and the values for the key differences can be calculated from plaintext information. Implementing this method requires the collection of plaintexts resulting a short encryption time under the assumption that a plaintext having a small number of cache misses equals a plaintext having a short encryption time. It can therefore be guessed that most of the collected plaintexts result in equivalent input values between the S-boxes in question. Key differences can therefore be calculated for the collected plaintexts and the value counted most frequently can be regarded as the correct key difference. We call this method a "non-elimination table attack."

The second method corresponds to the situation in which the input values of S-boxes under comparison are different. In this case, Eq. (2) holds and values of improbable key differences can be excluded. Implementing this method requires the collection of plaintexts resulting a long encryption time under the assumption that a plaintext having a large number of cache misses equals a plaintext having a long encryption time. Thus, the most of the collected plaintexts are guessed to result in different input values between the S-boxes. Key differences for the collected plaintexts can therefore be calculated and the value that appears the least frequently is taken as the correct key difference. We call this method an "elimination table attack."

For DES, the number of S-box operations is 16, a rather small one, considering that each S-box has 64 entries. Therefore, it is predicted that many input values will be different between the S-boxes, making it easy to collect plaintexts. Thus, we applied an elimination table attack on DES.

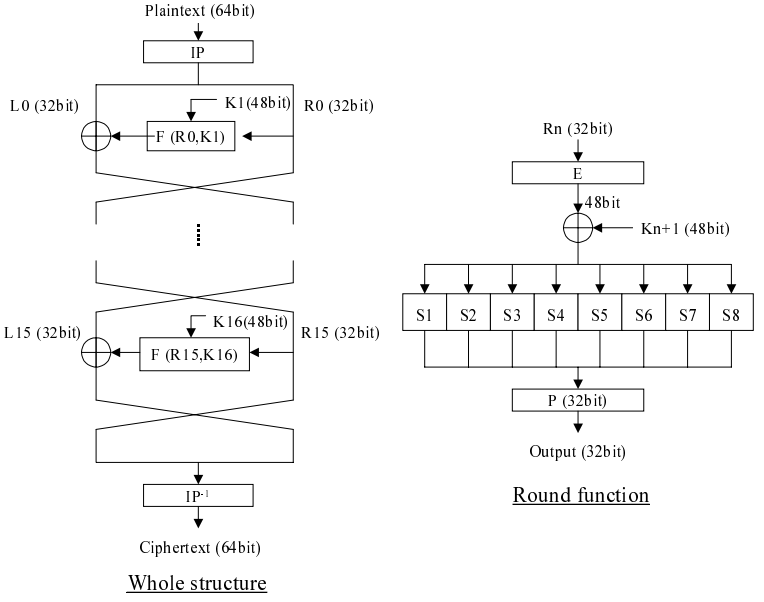


Fig. 2. Whole Structure and Round Function

3 Attack on DES

3.1 DES Structure

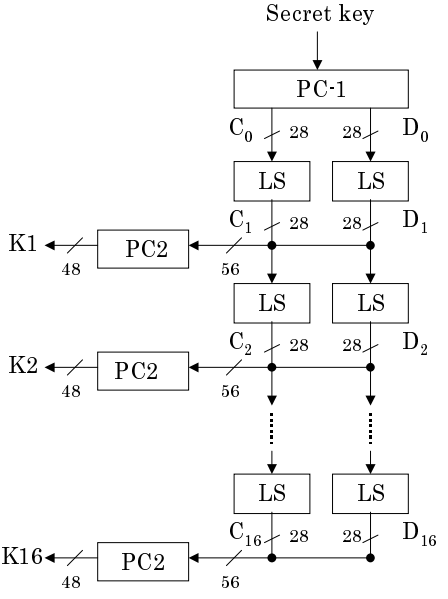
DES has a 16-round Feistel structure. Each round function features eight S-boxes each with a 6-bit input and a 4-bit output. An S-box operates 16 times, a small number compared to its $2^6 = 64$ entries. In the key scheduler, 48-bit of a 64-bit secret key is selected for each round, and its value is used as a expanded key for the corresponding round. Refer Fig. 2 and Fig. 3 for details.

As shown in Fig. 3, the total number of left cyclic shifts is set to 28 bits, which means that (C_0, D_0) and (C_{16}, D_{16}) have the same value. Thus, (C_1, D_1) used in the round 1 and (C_{16}, D_{16}) used in the round 16 are related by a 1-bit left cyclic shift. This relationship is used for the secret key recovery described in Section 3.2.

3.2 Attack Technique

This section describes the DES attack technique in detail. The steps making up this attack are divided into two main stages. Stage 1 is used to collect plaintexts for encryption, while Stage 2 is used to obtain key differences from the collected plaintexts. These stages are performed independently of each other.

The experiment described in this paper was done in the machine and compile environment summarized in Table 1.



Round number	Left shift number
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Fig. 3. Key Schedule

Collection of Plaintext. We first describe the method of collecting plaintext; Stage 1 of the attack. Here, plaintext having a long encryption time is needed to apply the elimination table attack described in Section 2.4. Our approach therefore is to encrypt a fixed amount of randomly generated plaintexts and to examine the resulting distribution of encryption time. The following method is used to measure the delay caused by cache misses as accurately as possible. The characteristics of the CPU (Pentium III) used in this experiment are also taken into account, and the DES source code that we use is the one described in [10]. In this source code, it is declared to assign 4 bytes to each entry of S-box, since S-box and bit permutation are computed simultaneously, for faster performance. The encryption time measurement method is as follows.

- Before beginning measurements, S-box data is deleted from the L1 data cache. In actuality, 16 kilobytes of random data are loaded into the 16-kilobyte data area of the L1 data cache to fill it.
- The *rdtsc* instruction, which loads the value of the processor's time stamp counter into a register, is used to measure encryption time ; the instruction is executed directly before and after encryption and the difference between the obtained values is used to compute the encryption time.

The above method enables to measure the encryption time for any plaintext and to collect plaintext/ciphertext pairs required for obtaining key differences.

Table 1. Experimental Environment

PC	NEC MateNX MA60J
CPU	Intel Pentium III(Katmai) 600MHz
L1 data cache	16-KB 4-Way Set Associative Cache 32-byte cache line
L2 cache (size / speed)	512KB / Half (300MHz)
Bus clock	100MHz
OS	Microsoft Windows2000 SP3
Compiler	Microsoft Visual C++ 6.0 SP5
Compile option	Maximize Speed (/O2)

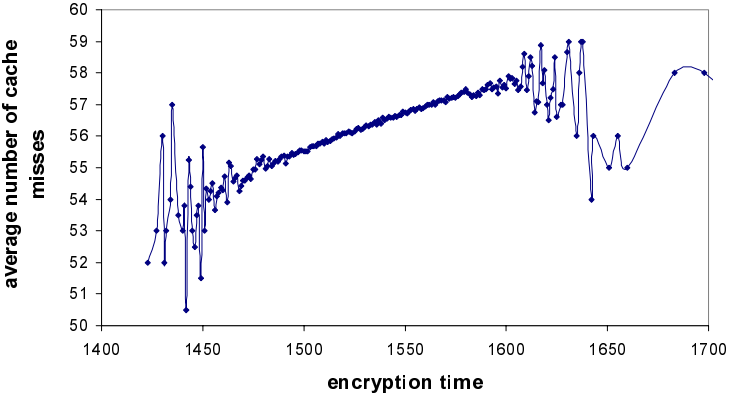


Fig. 4. Relationship Between Number of Cache Misses and Encryption Time

Relationship between Encryption Time and Cache Misses. We investigated whether the collected plaintexts actually operate as expected. Fig. 4 shows number of cache misses versus encryption time for the randomly generated plaintexts. We used a single arbitrary key for our experiment to measure the frequency of cache miss. Fig. 4 also shows that the number of cache misses increases as encryption time becomes long.

Fig. 5 shows the relationship between the number of plaintexts and the number of cache misses, for randomly generated plaintexts and plaintexts having a long encryption time. These results confirm that plaintexts having a long encryption time include significantly more plaintexts causing many cache misses than randomly generated plaintexts.

Making an Elimination Table Attack (Obtaining Key Differences). This part describes the method of obtaining key differences; Stage 2 of the attack. It is guessed that input values to the S-boxes of round 1 differ respectively from those to the corresponding S-boxes of round 16. Thus, Eq.(3) must hold.

$$K1 \oplus K16 \neq E(R0) \oplus E(R15) \tag{3}$$

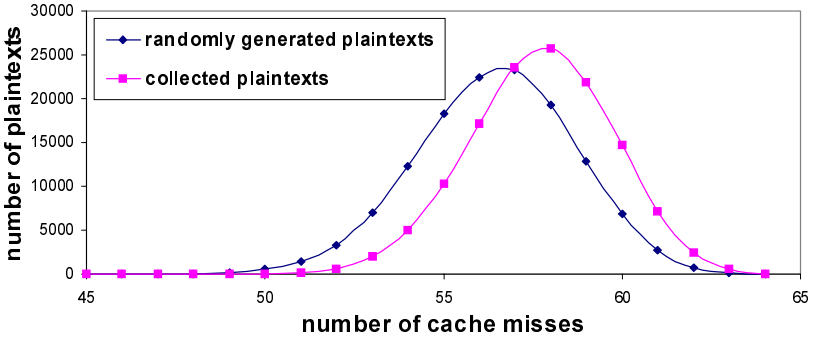


Fig. 5. Relationship Between Number of Plaintexts and Number of Cache Misses

Based on the concept presented in Section 2.4, the value appearing least frequently among those obtained by $E(R0) \oplus E(R15)$ is highly likely to be the correct key difference, when providing enough plaintexts collected in Stage 1. Thus, it can be determined that the value appearing least frequently as $E(R0) \oplus E(R15)$ is the correct key difference. This computation is performed for each pair of S-boxes S1 through S8 in rounds 1 and 16 to obtain eight key differences.

However, the 2 bits from the LSB; Least Significant Bit side of each key difference are indeterminate. This is because a cache miss does not occur if the input values of the 2 S-boxes under comparison differ to each other by the value within the range of the cache load size [3]. This means that the difference by the value less than the cache load size is ignored. Thus, the adjacent values of the value to be counted least frequently as a key difference are not counted, if the non-elimination table attack is applied. This is true to the adjacent values of the value to be counted most frequently, when elimination table attack is made. Thus, a key difference can be obtained, but the bits from the LSB side of it are still indeterminate; the 3 bits from that are be theoretically indeterminate. In our experiment, however, the 2 bits from the LSB side were found to be indeterminate because of absence of S-box addresses on a 32-byte boundary .

Considering above, we guess that the 4 bits from the MSB; Most Significant Bit side of each obtained key difference are correct, when recovering the secret key.

Recovering the Secret Key. The secret key is recovered from the 8 key differences obtained in Stage 2 in the following way.

Step 1. Prepare one plaintext/ciphertext pair by encrypting any plaintext with the actual secret key.

[3] The Pentium III Processor has a 32-byte cache load size i.e. 8 entries will be loaded simultaneously if it is declared to assign 4 bytes to each entry of S-box.

Table 2. Experimental Results

Number of Plaintext/Ciphertext Pairs used as 2^{n-m}	Number to be substituted for 2^{-m}	Probability of Success
2^{16}	2^{-6}	68.7%
	2^{-7}	74.7%
	2^{-8}	85.0%
2^{17}	2^{-6}	90.7%
	2^{-7}	92.3%
	2^{-8}	97.0%

Step 2. Determine 1 bit of the expanded key for round 16 by using the obtained key difference between round 1 and round 16 and then guessing any 1 bit of the expanded key for round 1. In this way, make a 32-bit (4 bit $\times$ 8 key differences) exhaustive search on the expanded key for round 1 with respect to the previously obtained key differences; this allows determining the expanded key for the corresponding round 16. 1 or more bits can be also determined by guessing 1 bit, based on the relationship between the two expanded keys for round 1 and round 16, which is described Section 3.1. Consequently, secret key is guessed by 24-bit exhaustive search on the expanded key for round 1. See the appendix for a detailed description on the secret-key recovery method.

Step 3. Encrypt the plaintext prepared in Step 1, using the secret key guessed in Step 2. If the resulting ciphertext agrees with the one obtained in Step 1, the secret key is correct. If they do not agree, return to Step 2. Note that if the secret key cannot be recovered by a 24-bit exhaustive search, the key differences guessed in Section 3.2 are mistaken.

3.3 Results of Experiment

Table 2 lists the results of DES elimination table attack described in Section 3.2. For the attack, we use 2^{n-m} out of 2^n randomly generated plaintexts which are collected in order of decreasing duration of encrypting. In reality, three numbers of 2^{-6}, 2^{-7} and 2^{-8} were taken as 2^{-m} to compare the probability of success of the attack, while two numbers of 2^{16} and 2^{17} were used as 2^{n-m}. The experiment was performed using 300 secret keys for two parameters; the number of plaintexts and the number to be substituted for 2^{-m}.

The results shown in Table 2 tell us that the secret key is recovered with a probability $> 90\%$, when collecting 2^{17} plaintext/ciphertext pairs and that setting a stricter condition for collecting plaintexts enables to collect the plaintext/ciphertext pairs having more cache misses.

3.4 Discussion

The above sections described a technique for breaking DES and the results of making attacks. Those results, however, are dependent on the experimental envi-

ronment specified in Table 1. Since a cache attack is a type of side-channel attack, there is a high possibility that the results will vary significantly according to the environment of computer. It is also thought that the result and its efficiency will vary according to source code. The following discusses these factors.

A cache attack infers the frequency of cache miss from side-channel information and uses it to obtain key differences. As a consequence, S-box size can have a great effect on the attack. In the DES source code used in our experiment, it is declared to assign 4 bytes to each entry of an S-box (referred to below as *int* type). For the S-box declared as *int* type, eight entries are loaded into the cache per 1 cache load. Thus, considering that an S-box of DES has 64 entries in all, all S-box data will be loaded into the cache if eight cache misses occur. In contrast, for an S-box, for which it is declared to assign 1 byte per entry (referred to below as *char* type), 32 entries are loaded into the cache per 1 cache load; this means that only two cache misses are needed to occur, to load all S-box data. It therefore seems impossible that the duration of encryption determines the frequency of cache miss and that useful plaintexts are selected and collected. For confirmation, we applied an experimental attack on DES with the source code described in [10], after changing only the S-box declaration type from *int* to *char*, to find that the attack failed entirely. However, when a 32-bit processor such as Pentium III is used, the *int* type data is processed faster than *char* type data. Thus, the data which can be declared as *char* type will often be declared as *int* type, when implementing ciphers. The kind of implementation for faster processing can lead to the vulnerability to cache attacks.

3.5 Attack on Triple-DES

In this section, we consider whether the above cache attack on DES can be made on Triple-DES. Triple-DES performs a DES process three times in the form of

- Encryption - Decryption - Encryption, or
- Encryption - Encryption - Encryption.

In addition, there are three ways of using keys, as follows.

(a) K1 - K2 - K3
(b) K1 - K2 - K1
(c) K1 - K1 - K1

Repeating DES three times in this manner makes greater resistance to cryptanalysis techniques like differential and linear cryptanalysis that employ the correlation of round functions. At the same time, secret key variations (a) and (b) feature a longer key length than DES, making it all the more difficult to perform an exhaustive key search.

In any of the above Triple-DES variations, an S-box operates 48 times; three times as many as DES. Still, if operation delay due to cache misses can be measured, it should be possible to make a cache attack against Triple-DES in the same way as DES.

Table 3. The type of S-box of cipher and the technique of applying the cache attack. S_{size} represents the number of S-box entries while S_{num} stands for the number of the times that S-box look-up is performed. S_{miss} represents the maximum possible number of the times that cache miss is caused by S-box look-up. *Technique* shows the combination of the type of the plaintexts used for cryptanalysis and the type of the technique of applying attack

	S_{size}	S_{num}	S_{miss}	*Technique*
DES	64	16	8	plaintexts with long encryption time and elimination table attack
MISTY1(S9)	512	48	64	
Camellia	256	36	32	
AES	256	160	8	plaintexts with long encryption time and non-elimination table attack

It is guessed that, similarly to DES, the key difference between round 1 and 48 can be determined. For cryptanalysis on an actual computer, we can expect 2 bits from LSB side of the key difference to be indeterminate and that the actually computed key difference consists of 4-bits$\times 8 = 32$ bits. Thus, for secret key variation (a) having a key length of 168 bits, the 32 bits of K3 can first be determined by guessing the 32 bits of K1. Then, if an exhaustive key search is performed on the remaining 104 bits$(= 168 - 64)$, it should be possible to break the cipher in 2^{136} calculations. This concept also holds for the other key variations, that is, it should be possible to break Triple-DES in a more efficient way than applying an exhaustive key search.

4 Other Ciphers

We made experimental cache attacks on AES and Camellia. Based on the results of the attacks, this section discusses the relationship between the number of the times that S-box look-up is performed and the cache attack.

4.1 Results of the Experiment

Table 3 shows the type of S-box and the technique of applying the cache attack for each cipher. Information on DES and MISTY1 is also given in the table for comparison. The following outlines the technique of applying cache attack on Camellia and AES.

Camellia. The source code is first modified by techniques for speeding-up the cipher which are recommended by the designer and described in the specification [3]. For each of the four S-boxes declared by the speeding-up techniques, the frequency of occurrence of cache miss is directly proportional to the encryption time, as is observed for DES. The usage of this property and 2^{18} plaintexts with long encryption time provides obtaining 168-bit key differences concerning with

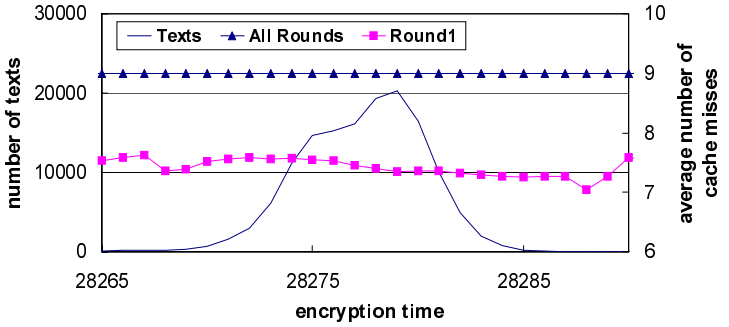


Fig. 6. Correlation Between the Encryption Time and the Average Number of Cache Misses (on AES). This graph also represents the encryption time distribution of plaintexts

a 256-bit equivalent key composed by the subkeys on round 1 through round 4, and the subkeys that are activated by the initial processing. Using the obtained key differences, approximately 2^{24} computations provides the recovery of the secret key.

AES. We employed the available source code in [7]. No correlation is found between the frequency of occurrence of cache miss and the encryption time. (See the plot labelled "All rounds" in Fig.6) However, studies on the 16 S-boxes used at the beginning of the algortithm have shown the correlation that lower frequency of cache misses implies longer encryption time. (See the plot labelled "Round 1" in Fig.6) This property provides 96-bit key differences through collecting 2^{18} plaintexts with long encryption time and regarding the value counted most often as a correct key difference. A 32-bit brute-force search using these key differences allows recovering the secret key.

4.2 Discussion

According to the paper [8] written by Ohkuma et al., the cache attack is theoretically feasible even if the number of the times that S-box look-up is performed is fairly large. The following equation represents the probability that the value of the frequency of cache miss is n, where N and M stand for the number of S-box input and the number of the times that S-box look-up is performed, respectively.

$$N^{-M} \binom{N}{n} \sum_{j=1}^{n} \binom{n}{j} (-1)^{n-j} j^M$$

This equation also indicates that it is theoretically feasible to break a cipher, if the cipher has the possibility that the value of the frequency of cache miss varies, depending on the collected plaintexts.

The accurate difference between the values of the frequency of cache miss is hard to obtain, when the attack is applied using a practical computer. When the cipher (e.g. AES) does not cause significant difference between the values of the frequency of cache miss, regardless of the plaintexts used for the attack, it is hard to perform the cryptanalysis using the values of the frequency of cache miss and the encryption time. However, we broke such kind of cipher, by utilizing the correlation between the encryption time and the probability that the values for some of S-box inputs are identical. When the ciphers cause fewer S-box look-ups and significant variations in the frequency of cache miss, like DES , we can expect that the frequency of cache miss and the encryption time correlate to each other. The correlation between the encryption time and the probability that the values for some of S-box inputs are identical, which is used for the cryptanalysis of AES, however, varies significantly, depending on the type of CPU and the method of implementing source code. For example, the durations of encrypting two sets of plaintexts with the same values of total frequency of cache miss on Intel Pentium III processor are sometimes different, depending on whether or not the cache misses occur continuously at the beginning of the encryption. In addition, which core is used for Intel Pentium III processor, Coppermine or Katmai decides which S-box to use for cryptanalysis and which attack to apply, non-elimination table attack or elimination table attack. In this case, the cipher can be broken, if we take possession of the source cord of the target cipher in advance and find the values of S-box inputs whose probability of being identical correlates to the encryption time.

5 Conclusion

We have shown that the Data Encryption Standard (DES) can be broken with 2^{23} known plaintexts and 2^{24} calculations at a success rate $> 90\%$, using a personal computer with 600-MHz Pentium III. We have also shown that a cache attack can be made against a cipher using S-boxes of different input/output widths or S-boxes of several types. Furthermore, in applying this cache attack to Triple-DES, it was found that there is a high possibility of it being broken more efficiently than an exhaustive key search.

This paper reports applying cache attack using a personal computer. In 2002, cache based cryptanalysis [9] was proposed where cache hits and/or cache misses are observed by the use of electric power or magnetic force. Since the next generation of 32-bit smartcards will use cache memories, the combination of the cache attack we proposed and Power Analysis attacks could probably be a more effective cryptanalysis technique.

We also consider countermeasures against cache attacks; a cache attack infers the number of times of occurred cache misses by observing the encryption time. Thus, if a total-data load is executed before processing, differences between the frequencies of cache misses will not be observed, making it impossible to determine the relationships between sets of S-boxes. If it is possible to clear a cache

during the encryption, generating noise that has no relation with encryption at random time intervals is an effective countermeasure against cache attacks.

The cache attacks are newer technique in comparison with the timing attacks on RSA. The encryption efficiencies can be enhanced by the studies to be conducted in future.

Acknowledgement. The authors would like to thank Hiroyasu Kubo, Takeshi Kawabata, Etsuko Tsujihara and Yuya Yoshioka for their useful comments and suggestions. We are also grateful to the anonymous reviewers of CHES 2003 for their helpful comments.

References

1. E. Biham, A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," CRYPTO'97, LNCS1294, pp. 513–525, 1997.
2. J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestre, J.J. Quisquater, J.L. Willems, "A Practical Implementation of the Timing Attack", UCL Report, 1998, CG1998-1, available at http://www.dice.ucl.ac.be/crypto/techreports.html
3. Information-Technology Promotion Agency, Japan and Telecomminications AdvancementOrganization of Japan, "CRYPTREC Report 2001," 2002.
4. J. Kelsey, B. Schneier, D. Wagner, C. Hall, "Side Channel Cryptanalysis of Product Ciphers," Journal of Computer Security, vol.8, pp. 141–158, 2000.
5. P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis," CRYPTO'99, LNCS1666, pp. 388–397, Springer-Verlag, 1999.
6. F. Koeune, J.J. Quisquater, "A Timing Attack against Rijndael," UCL Report, CG1999-1, 1999, available at http://www.dice.ucl.ac.be/crypto/techreports.html
7. National Institute of Standards and Technology, "ANSI C Reference Code V2.0 (October 24, 2000)," available at http://csrc.nist.gov/CryptoToolkit/aes/rijndael/
8. K. Ohkuma, S.Kawamura, H.Shimizu, H.Muratani, "Key Inference in a Side-Channel Attack Based on Cache Miss," The 2003 Symposium on Cyptography and Information Security, 2003.(In Japanese)
9. 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 available at http://www.cs.bris.ac.uk/
10. B. Schneier, "APPLIED CRYPTOGRAPHY," John Wiley & Sons, Inc. , 1996.
11. Y. Tsunoo, E. Tsujihara, K. Minematsu, H. Miyauchi, "Cryptanalysis of Block Ciphers Implemented on Computers with Cache," ISITA 2002, 2002.
12. "Data Encryption Standard (DES)," Federal Information Processing Standards Publication 46-3, 1999, available at http://csrc.nist.gov/publications/fips/

Appendix: Secret-Key Recovering Method

The following describes the process for recovering a secret key. As is described in the body of this paper, the following precondition must be satisfied to recover a secret key.

– The 4 bits from the MSB side of each key difference between S1 through S8 S-boxes of round 1 and round 16 can be obtained.

In the following, nth bit from the MSB side of the variable X is defined $X[n]$. We take the expanded key K1 of round 1 as an example:

$$K1 = K1[1] \| K1[2] \| \cdots \| K1[48]$$

Next, based on key-schedule structure, the relationship between computed key differences and secret-key information C and D can be represented in the following way.

$$\begin{aligned} K1 \oplus K16 &= PC2(C_1) \oplus PC2(C_{16}) \\ &= PC2(C_1) \oplus PC2(LS(C_1)) \end{aligned} \tag{4}$$

Using Eq. (4), C_1 and D_1 information can be computed in a step-by-step manner. The following 16 equations are Eq. (4)s expressed on a bit basis.

$$\begin{aligned} K1[7] \oplus K16[7] &= C_1[3] \oplus C_1[4] \\ K1[16] \oplus K16[16] &= C_1[4] \oplus C_1[5] \\ K1[10] \oplus K16[10] &= C_1[6] \oplus C_1[7] \\ K1[20] \oplus K16[20] &= C_1[7] \oplus C_1[8] \\ K1[3] \oplus K16[3] &= C_1[11] \oplus C_1[12] \\ K1[15] \oplus K16[15] &= C_1[12] \oplus C_1[13] \\ K1[1] \oplus K16[1] &= C_1[14] \oplus C_1[15] \\ K1[9] \oplus K16[9] &= C_1[15] \oplus C_1[16] \\ K1[19] \oplus K16[19] &= C_1[16] \oplus C_1[17] \\ K1[2] \oplus K16[2] &= C_1[17] \oplus C_1[18] \\ K1[14] \oplus K16[14] &= C_1[19] \oplus C_1[20] \\ K1[22] \oplus K16[22] &= C_1[20] \oplus C_1[21] \\ K1[13] \oplus K16[13] &= C_1[23] \oplus C_1[24] \\ K1[4] \oplus K16[4] &= C_1[24] \oplus C_1[25] \\ K1[21] \oplus K16[21] &= C_1[27] \oplus C_1[28] \\ K1[8] \oplus K16[8] &= C_1[28] \oplus C_1[1] \end{aligned}$$

Using the 16 equations above to guess 7 bits of $C_1[3]$, $C_1[6]$, $C_1[11]$, $C_1[14]$, $C_1[19]$, $C_1[23]$, and $C_1[27]$ allows obtaining 16 bit of $C_1[4]$, $C_1[5]$, $C_1[7]$, $C_1[8]$, $C_1[12]$, $C_1[13]$, $C_1[15]$, $C_1[16]$, $C_1[17]$, $C_1[18]$, $C_1[20]$, $C_1[21]$, $C_1[24]$, $C_1[25]$, $C_1[28]$, and $C_1[1]$. 28 bits, i.e. all bits of C_1 are obtained by guessing 7 bits in the way described above and then guessing the remaining 5 bits of C_1; $C_1[2]$, $C_1[9]$, $C_1[10]$, $C_1[22]$, $C_1[26]$.

D_1 is treated similarly. 12-bit exhaustive search on $D_1[2]$, $D_1[5]$, $D_1[6]$, $D_1[7]$, $D_1[8]$, $D_1[11]$, $D_1[16]$ $D_1[20]$ $D_1[21]$, $D_1[26]$, $D_1[27]$, and $D_1[28]$ allows determining the 28-bit of D1 .

Overall, the above uniquely recovers 56 bits of the secret key by guessing 24 bits.