Cache Timing Attacks on Camellia Block Cipher*

ZHAO Xin-jie, WANG Tao, ZHENG Yuan-yuan

(Department of Computer Engineering, Ordnance Engineering College, Shijiazhuang 050003, China)

zhaoxinjieem@163.com

Abstract: Camellia, as the final winner of 128-bit block cipher in NESSIE, is the most secure block cipher of the world. In 2003, Tsunoo proposed a Cache Attack using a timing of CPU cache, successfully recovered Camellia-128 key within 2^{28} plaintexts and 35 minutes. In 2004, IKEDA YOSHITAKA made some further improvements on Tsunoo's attacks, recovered Camellia-128 key within $2^{21.4}$ plaintexts and 22 minutes. All of their attacks are belonged to timing driven Cache attacks, our research shows that, due to its frequent S-box lookup operations, Camellia is also quite vulnerable to access driven Cache timing attacks, and it is much more effective than timing driven Cache attacks. Firstly, we provide a general analysis model for symmetric ciphers using S-box based on access driven Cache timing attacks, point out that the F function of the Camellia can leak information about the result of encryption key XORed with expand-key, and the left circular rotating operation of the key schedule in Camellia has serious designing problem. Next, we present several attacks on Camellia-128/192/256 with and without FL/FL^{-1}. Experiment results demonstrate: 500 random plaintexts are enough to recover full Camellia-128 key; 900 random plaintexts are enough to recover full Camellia-192/256 key; also, our attacks can be expanded to known ciphertext conditions by attacking the Camellia decryption procedure; besides, our attacks are quite easy to be expanded to remote scenarios, 3000 random plaintexts are enough to recover full encryption key of Camellia-128/192/256 in both local and campus networks. Finally, we discuss the reason why Camellia is weak in this type of attack, and provide some advices to cipher designers for hardening ciphers against cache timing attacks.

Key words: Camellia-128/192/256; block cipher; access driven; Cache timing attack; side channel attack; remote attack; F function; S-box lookup index; left circular rotating operation; key schedule; known ciphertext

1 Introduction

1.1 Related Works

Recently, with the introduction of side channel attacks, ciphers were facing serious threats. Traditionally speaking, the security of cipher depends on the mathematical function $E_K[P]\rightarrow C$, the adversary tries to crack K from (P,C) by linear or differential methods. With the development of cipher designing, both the key length and algorithm complexity has been greatly improved, so it's very difficult to predict K through mathematically analysis. However, recent research demonstrates that: during the cipher execution procedure, it may leaks some other information, such as execution timing, power consumption, electromagnetic dissipation, faults etc, which is what we called side channel information. So, in reality, a cipher is a function $E_K[P]\rightarrow(C, L)$. The additional side channel information L has close relations with encrypt/decrypt key K. Through certain methods, combing the P or C, the adversary can derive K efficiently. The attacks presented in this paper use timing data.

Cache Timing Attacks are new classes of side channel attacks. The feasibility of the Cache attacks was first mentioned by Kocher[1] and then Kelsey et al.[2] in 1996. They assumed that the adversary can use timing measurements to learn something about the cache accesses of a legitimate party, which turns out to be the case in some practical applications. Page[3] described and simulated a theoretical cache attack on DES in 2002. And subsequently, Tsunoo et al. [4][5] proposed a Cache Attack using a timing of CPU cache, successfully devised timing-based attack on DES and Camellia. In 2004, IKEDA YOSHITAKA et al.[6] made some further improvement on Tsunoo et al.'s Camellia attacks. In 2005, Bernstein[7] and Osvik, Shamir, and Tromer[8][9] showed in independent work that the Advanced Encryption Standard (AES) is particularly vulnerable to this type of side-channel attack, generating a lot of attention for the field. Subsequent work dealt with verifying the findings[10][11][12][13][14], improving the attack[15][16][17][18][19], and devising and analyzing countermeasures[20][21][22].

The cryptanalytic attention was mainly focused on DES and AES, and the countermeasures mainly target the implementation of cryptographic designs. None of the mentioned papers above except Tsunoo[5] and IKEDA YOSHITAKA[6] analyzed the Camellia-128 with Cache timing attacks. In Tsunoo's Camellia attack[5], they picked up 2^{18} faster plaintexts from 2^{28} random plaintexts and using

* Supported by the National Natural Science Foundation of China (Grant No. 60772082) and the Natural Science Foundation of Hebei Province. China (Grant No. 08M010). ZHAO Xin-jie, male, born in 1986, Ph.D.candidate, his main research interests include: Cache based side-channel attack analysis on Block cipher, formal methods analysis; WANG Tao, male, born in 1964, Ph.D., professor, Ph.D. supervisor, his main research interests include: Network security analysis, side-channel attack analysis; ZHENG Yuan-yuan, female, born in 1981, Ph.D.candidate, her main research interests include: Public key encryption security analysis, timing attack analysis.

them with 2^{24} brute force search for the sub-keys, finally found the secret key; In IKEDA YOSHITAKA's Camellia-128 attack [6] , they made some further improvement on Tsunoo et al.'s Camellia-128 attack, through finding constraints of S-box inputs, recovered Camellia key with $2^{21.4}$ plaintexts and 22 minutes CPU time under PentiumIII 550MHz. All of the Camellia attacks were measuring the whole encryption times and exploiting the effects of collisions between the various memory lookups invoked internally by the cipher, thus their attacks were all belonged to timing driven Cache attacks, and millions of samples were needed to predict the correct key. In the remote network environment, even the traffic jitter is more than the encryption time, so their attacks were quite impossible to be implemented into real remote scenarios. In this paper, we analyzed access driven Cache timing attacks on Camellia-128/192/256, a spy process was used to gather the accessed Cache set during Camellia encryption or decryption, and combined certain analysis methods to predict the encryption key. Our research shows that no matter known plaintext attack, known ciphertext attack, local and remote attack, it's possible to recover full encryption key of Camellia-128/192/256 within less than 2^{12} samples and 1 second analysis, even Camellia in OPENSSL-1.0.0-beta3 with FL/FL^{-1} function, which is the latest version published by July 2009.

1.2 Our Contributions

The contributions of our work can be summarized as follows:

1) Provide a general analysis model for symmetric ciphers based on Cache timing side channel attack.

This model can be applied to analyze any symmetric cipher using S-boxes, such as AES, SMS4, Camellia, ARIA, HC-128, HC-256 etc, it's quite effective, also it can be used to evaluate and test the security of ciphers against Cache timing attacks.

2) Point out that Camellia F function might leak information about encryption key.

During the analysis of Camellia F function, we find out: the Camellia F function of encryption procedure can leak secret information about $K_i \oplus KE_j$, K_i denotes one byte of encryption key K, KE_j denotes one byte of the expand-key, and Camellia F function in key schedule can leak secret information about K_i or KE_j, which means that sometimes we may derive partial bytes of K directly.

3) The <<< n left circular rotating operation of the key schedule in Camellia has serious designing problem.

Once the adversary find a XOR collision about two variables generated by the left circular rotating of the same secret variable, it's very easy to recover the correct secret variable. Combing the cipher algorithm design, the adversary can recover encryption key quite efficiently. Our experiments demonstrate that just analyzing one <<< 15 operation can narrow down the key searching space from2^{128} to 2. And the key schedule of Camellia-192/256 has serious designing problem, the 5th input of the F function $K_A \oplus K_R$ can be recovered due to the left circular rotating of K_A and K_R by the same 15 bits during the generation of the 3rd to 6th round sub-key, and the output of the 6th F function K_B can be recovered, which enable us to further recover full Camellia-192/256 key directly.

4) Propose and realize several Cache timing attacks on Camellia-128/192/256.

After analyzing on the F function and <<< operation of Camellia implementation, we have successfully designed and implemented several Cache timing attacks on Camellia-128/192/256. Experiment results demonstrate that even Camellia with FL/FL^{-1} is vulnerable to Cache timing attacks, all of the attacks can be realized within 2^{12} samples and 1 second analysis; also, our attacks can be expanded to known ciphertext conditions by attacking the Camellia decryption procedure. Moreover, our techniques can be easily adapted into remote scenarios, 3000 random plaintexts are enough to recover full encryption key for Camellia-128/192/256 in both local and campus networks. Table 1 demonstrates the improvements of the attacks in this paper over several previous attacks.

Table 1. Overview of attacks against Camellia

Cipher	Attack	Type of Attack	Rounds	FL/FL^{-1}	Data	Time	Key Recovery
Camellia-128	[23]	Square attack	6	x	$2^{11.7}$	2^{112}	128-bit
Camellia-128	[24]	Truncated differential	8	x	$2^{83.6}$	$2^{55.6}$	128-bit
Camellia-128	[25]	Impossible differential	7	x	---	---	128-bit
Camellia-128	[33]	Impossible differential	11	x	2^{120}	$2^{83.4}$	128-bit
Camellia-128	[5]	Cache Timing attack	6	x	2^{28}	35minutes	128-bit
Camellia-128	[6]	Cache Timing attack	6	x	$2^{21.4}$	22minutes	128-bit
Camellia-128	Section 4	Cache Timing attack	4	x / √	$2^{8.97}$	1s	128-bit
Camellia-128	Section 6	Known ciphertext Cache Timing attack	4	x / √	$2^{8.97}$	1s	128-bit
Camellia-128	Section 7	Remote Cache Timing attack	4	x / √	$2^{11.55}$	1s	128-bit
Camellia-192/256	[33]	Boomerang attack	9	√	2^{124}	2^{170}	192/256-bit
Camellia-192/256	[27]	Collision attack	9	x	2^{13}	$2^{175.6}$	192/256-bit
Camellia-192/256	[28]	Square attack	10	x	---	2^{186}	192/256-bit
Camellia-192/256	[29]	Impossible differential	12	x	2^{120}	2^{181}	192/256-bit
Camellia-192/256	[33]	Impossible differential	12	x	2^{119}	$2^{147.3}$	192/256-bit
Camellia-192/256	Section 5	Cache Timing attack	6	x / √	$2^{9.81}$	1s	192/256-bit
Camellia-192/256	Section 6	Known ciphertext Cache Timing attack	6	x / √	$2^{9.81}$	1s	192/256-bit
Camellia-192/256	Section 7	Remote Cache Timing attack	6	x / √	$2^{11.55}$	1s	192/256-bit
Camellia-256	[30]	Square attack	9	√	2^{60}	2^{202}	256-bit
Camellia-256	[31]	Integral cryptanalysis	9	√	$2^{60.5}$	$2^{202.5}$	256-bit
Camellia-256	[26]	Rectangle attack	10	√	2^{127}	2^{241}	256-bit
Camellia-256	[27]	Collision attack	10	x	2^{14}	$2^{239.9}$	256-bit
Camellia-256	[26]	Differential	11	x	2^{104}	2^{232}	256-bit

Cipher	Attack	Type of Attack	Rounds	FL/FL^{-1}	Data	Time	Key Recovery
Camellia-256	[32]	High-order differential	11	x	2^{21}	2^{255}	256-bit
Camellia-256	[32]	High-order differential	11	√	2^{93}	2^{256}	256-bit
Camellia-256	[34]	Square attack	11	x	---	2^{250}	256-bit
Camellia-256	[26]	Linear cryptanalysis	12	x	2^{119}	2^{247}	256-bit
Camellia-256	[33]	Impossible differential	13	x	2^{120}	$2^{211.7}$	256-bit

5) Discuss why and how these attacks work on Camellia, and provide some possible countermeasures.

The responsibility of this attack should mainly lies in Camellia algorithm itself, both the F function and key schedule design of Camellia have serious weaknesses, which can be used to analyze the key efficiently. So, in order to prevent such kind of attack, we discuss how cipher designers can make such attacks more difficult in Section 8.

1.3 Organization

This paper is organized as follows: Section 2 briefly describes preliminaries of Cache timing attack on Camellia. Section 3 presents the basic attack model and how it can be used into Camellia F function analysis. Section 4 presents attacks on Camellia-128, and Section 5 displays attacks on Camellia-192/256. Section 6 displays several known ciphertext attacks on Camellia, and Section 7 displays remote timing attacks on Camellia. Section 8 discusses on the reason why Camellia is vulnerable to this type of attack and provides several advices to the cipher designer. Section 9 is the conclusion.

2 Preliminaries

2.1 Notations

S_1, S_2, S_3, S_4: denote SBOX1_1110, SBOX2_0222, SBOX3_3033, SBOX4_4404 S-box separately in the code of Camellia integrated in OPENSSL-1.0.0-beta3.

X_L: the left-half data of X

X_R: the right-half data of X

$\oplus$: bitwise exclusive-OR operation

||: concatenation of two operations

<<< n: rotations to the left by n bits

K: Camilla encryption key

KE: Camellia expand key

δ: the element count of a Cache block

s^r_i: denotes the *i-st* 32-bit input of the *r-st* F function call

2.2 Cache Timing Attack

Cache timing attacks exploit that loading data into a CPU register is faster when done from Cache than from RAM. By measuring Cache timings, the adversary can obtain or deduce certain information about the inner state of cipher. In the following, we point out why and how Cache becomes a convert channel for side channel attacks.

Cache workings:

Modern processors use one or more levels of set-associate memory Cache to solve the bottleneck between CPU and bus bandwidth. The cache is divided into S Cache sets, each contains W Cache lines, each line contains δ Cache elements (B bytes), so the overall Cache size is $S*W*B$ bytes.

The mapping of memory addresses into the Cache is limited as follows:

Feature 1: When CPU reads a word A from the main memory, it first sends the memory address of A to the Cache and main memory, after that, Cache control logic unit judges whether A is in the Cache right now, if it is, a "Cache hit" occurs; if not, a "Cache miss" occurs, not only A but the memory block A belonging to (B bytes) is copied into one of the Cache lines.

Feature 2: Each memory block can be cached only a specific Cache set, specifically, the memory block started at address a can be cached only in Cache set $[a/B]$ mod/S.

From Feature 1 we know that the same instruction to access the memory is affected by the historic state of whether the target data is in the Cache, if not , a delay by "Cache miss" appears, which might be represented by longer execute clock cycles or more power consumptions of the program. As to the clock cycle variation of typical processors, a "Cache hit" approximately needs 3 cycles, while a "Cache miss" might spend 12-100 cycles. So Cache hit and missing feature provides the timing leakage source for timing attacks.

According to Feature 2, when different processes access self private data, as these data can be mapped into the same Cache set, thus share the Cache space together, so the spy process can monitor other process's Cache access patterns by detecting self data access timing and power consumption pattern. So Cache resources sharing mechanism provides the convert channel for timing attacks.

From the analysis above, the adversary can obtain a profile of Cache blocks that have been used by the cipher process. In local attack, this profile has very little noise from other system processes, and in remote attacks, this profile has more noises from the receiving and sending data packets from the network. By repeating the experiment a number of times or increase the sample size, a good approximation of the real Cache access profile can be obtained.

Note that instead of learning the content of the Cache block, the adversary learns something about the addresses of the Cache set accessed by the cipher. In symmetric ciphers using S-box, this address can be transferred into the indices of the S-box entries used for encryption, which in return can be used for an attack.

Attack Classifications and Practicality:

According to the attacked unit of cipher implementation, Cache timing attack can be classified into data Cache, instruction Cache two types. Modern symmetric ciphers use many S-boxes to access data Cache, and become the main targets of data Cache timing attacks; meanwhile, public-key cryptosystems use many switch or jump cases to access instruction Cache, and thus become the main targets for I-Cache timing attacks.

According to the different timing information gathered from Cache, the attack can be classified into timing driven, access driven, trace driven three types.

As we all know, the attack can be composed of measuring and analyzing two phases. Timing driven Attack measuring phase is quite simple, it just measures the whole encrypt/decrypt times, combing certain statistics methods to analyze the key, due to the big timing noises by other processes, it needs millions of samples and complex methods to predict the correct key, meanwhile, in the remote network environment, even the traffic jitter is more than the encrypt/decrypt time, so it's quite impossible to implement it into real remote scenarios.

In access driven timing attack, the adversary use a spy process to get the Cache access profile about which Cache set has been accessed or un-accessed during one round or one time encryption, it's much more efficient than timing driven attack. As the measuring spy process is disposed in the target cryptographic server, so the measuring accuracy and analysis efficiency is quite high, it has good applicability in both local and remote scenarios.

Trace-driven analysis is of high efficiency, but needs to get the specific Cache hit/miss information during each S-box lookup to access the Cache, generally speaking, it is usually realized on electromagnetic leakage and power analysis, also needs to contact with the encryption equipment physically. So, the applicability of this attack in both local and remote environment is not quite practical.

Attack Responsibility:

Cache hit and miss timing variations provides the source for timing attacks; Cache space sharing and OS scheduling mechanism becomes the convert channel for spy process to measure cipher's Cache access profile, which can be transferred into indices for S-box lookup entries; modern ciphers use many S-boxes access Cache to improve the efficiency, the S-box lookup indices has close relationship with the key. So Cache, OS, cipher algorithm all should share the responsibility for Cache timing attacks. Generally speaking, defending against Cache timing attack should consider from three aspects above, but due to the countermeasure cost and cipher applicability, the cipher designer should burden more responsibilities. The reason is that algorithms are designed only once, but implemented many times on many platforms. Thus, if we find out the convert channel in the algorithms, fix it at the balance of timing and security, thus side channel attacks can be avoided into the design phase, implementation become easier, which seems to be more preferable. Section 7 analyzes the convert Cache timing leakage channel and provides several advices for the cipher designers.

Due to the applicability of access driven Cache timing attacks, we choose block cipher Camellia as the attack target, so it's also belonged to data Cache attack.

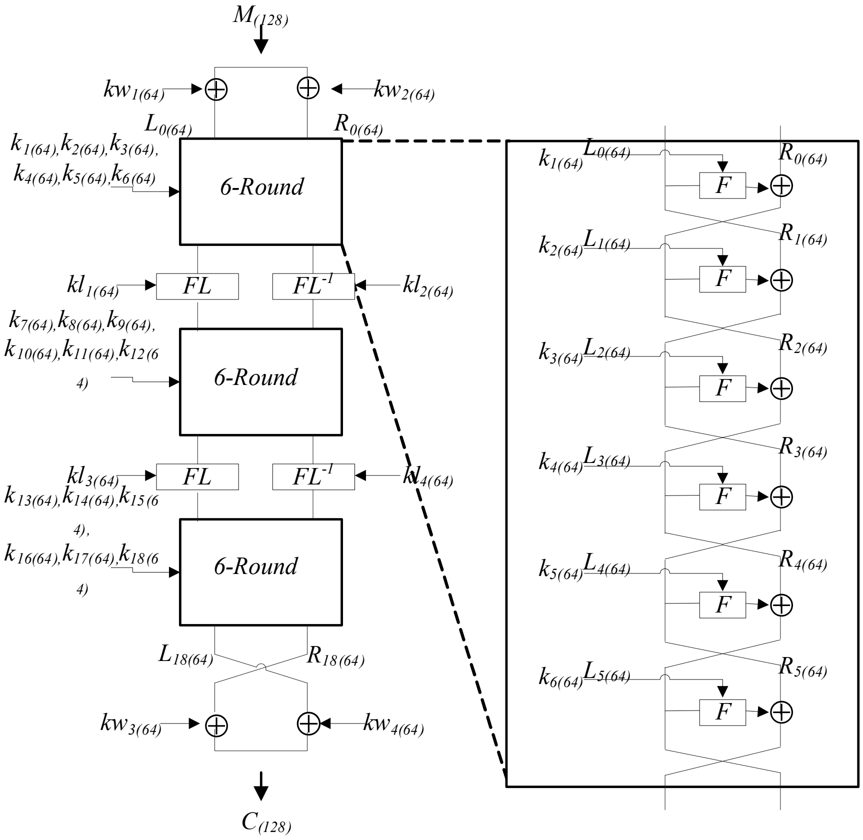
2.3 Description of the Camellia

Camellia is a 128-bit block cipher jointly developed by NTT and Mitsubishi Electric Corporation in 2000[35]. It was chosen as a recommended algorithm by the NESSIE (New European Schemes for Signatures, Integrity and Encryption) project in 2003 [36] and was certified as the IETF (Internet Engineering Task Force) standard cipher for XML security URIs, SSL/TLS cipher suites and IPsec in 2005[37][38][39]. In March 2009, Camellia was integrated into the OPENSSL-1.0.0-beta1[40], which is the most widespread cryptographic library of the world. A full description of the Camellia cipher is provided in [35][36], but below is a brief description of the cipher's properties that are utilized in this study.

Encryption Procedure:

Camellia is an iterated cipher. Camellia takes a 128-bit plaintext P as input, and has a total of N rounds, where N is 18 for Camellia-128, and 24 for Camellia-192/256. Camellia-128(192/256) requires 22(29)-rounds of data processing composed of three main parts: an 18(24)-round Feistel structure, two (three) FL function and FL^{-1} function rounds inserted every 6 rounds, and two input/output whitenings. Fig.1 shows the entire encryption process using 128-bit key. In the first and last round, the 128-bit data block is XORed with 128-bit round keys. Before the data block is fed to the Feistel network, it is separated into two 64-bit data blocks. The left half goes into the F function together with the 64-bit round key and the output of the F function is XORed with the right half block. At the end of each round, the right and left half block will be exchanged. In the F function, the input 64-bit data is first XORed with the 64-bit round key and then grouped into eight 8-bit data blocks. All of them are separately input to eight S-boxes.

In Camellia, four types of S-boxes are applied and each one consists of a multiplicative inversion and affine transformations. A linear 64-bit permutation follows the nonlinear substitution of S-boxes. The *FL* and FL^{-1} functions inserted every 6 rounds are used to provide non-regularity between the rounds so that the security of the cipher is increased and these two functions are similarly constructed by logical operations including AND, OR, XOR, and rotations.



$M_{(128)}$
$kw_{1(64)}$
$kw_{2(64)}$
$L_{0(64)}$
$R_{0(64)}$
$k_{1(64)},k_{2(64)},k_{3(64)},$
$k_{4(64)},k_{5(64)},k_{6(64)}$
6-Round
$kl_{1(64)}$
FL
FL^{-1}
$kl_{2(64)}$
$k_{7(64)},k_{8(64)},k_{9(64)},$
$k_{10(64)},k_{11(64)},k_{12(64)}$
6-Round
$kl_{3(64)}$
FL
FL^{-1}
$kl_{4(64)}$
$k_{13(64)},k_{14(64)},k_{15(64)},$
$k_{16(64)},k_{17(64)},k_{18(64)}$
6-Round
$L_{18(64)}$
$R_{18(64)}$
$kw_{3(64)}$
$kw_{4(64)}$
$C_{(128)}$
$k_{1(64)}$ $L_{0(64)}$ $R_{0(64)}$
F
$k_{2(64)}$ $L_{1(64)}$ $R_{1(64)}$
F
$k_{3(64)}$ $L_{2(64)}$ $R_{2(64)}$
F
$k_{4(64)}$ $L_{3(64)}$ $R_{3(64)}$
F
$k_{5(64)}$ $L_{4(64)}$ $R_{4(64)}$
F
$k_{6(64)}$ $L_{5(64)}$ $R_{5(64)}$
F

Fig.1 Encryption process of Camellia

Key Schedule:

Fig.2 shows the key schedule of Camellia. Two 128-bit variables K_L and K_R are defined as follows. For 128-bit keys, the 128-bit key *K* is used as K_L and K_R is 0. For 192-bit keys, the left 128-bit of the key *K* is used as K_L, and concatenation of the right 64-bit of *K* and the complement of the right 64-bit of *K* is used as K_R. For 256-bit keys, the left 128-bit of the key *K* is used as K_L and the right 128-bit of *K* is used as K_R. Two 128-bit variables K_A and K_B are generated from K_L and K_R as shown in Fig 2. Note that K_B is used only if the length of the secret key is 192 or 256 bits. The 64-bit constants $\sum_i$ (i = 1, 2, …, 6) are used as ”keys” in the Feistel network. They are defined as continuous values from the second hexadecimal place to the seventeenth hexadecimal place of the hexadecimal representation of the square root of the *i*-th prime. The 64-bit sub-keys k_{wt}, k_u, and k_{lv} are generated from K_L, K_R, K_A, and K_B. The sub-keys are generated by rotating K_L, K_R, K_A, and K_B and taking the left or right-half of them. Details are shown in Table 2 and Table 3.

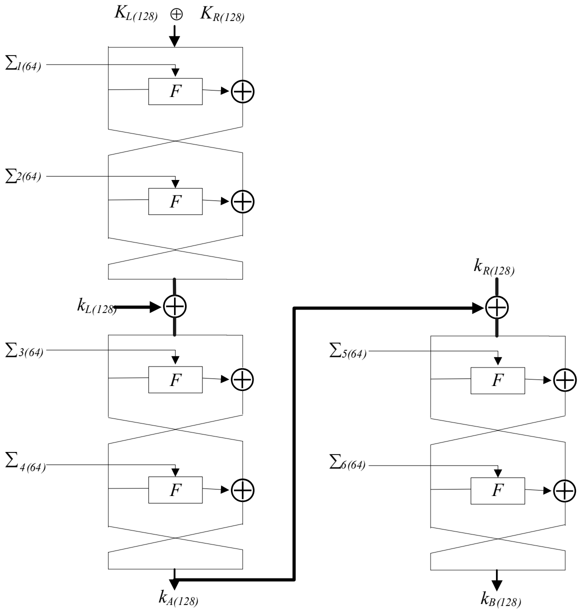


Fig.2 key schedule of Camellia

Table 2. Sub-keys for 128-bit keys

NO	Operation	subkey	value	NO	Operation	subkey	value
1	Prewhitening	$kw_1(KE_0, KE_1)$	$(K_L<<<0)_L$	14	F (Round10)	$k_{10}(KE_{26}, KE_{27})$	$(K_L<<<60)_R$
2	Prewhitening	$kw_2(KE_2, KE_3)$	$(K_L<<<0)_R$	15	F (Round11)	$k_{11}(KE_{28}, KE_{29})$	$(K_A<<<60)_L$
3	F (Round1)	$k_1(KE_4, KE_5)$	$(K_A<<<0)_L$	16	F (Round12)	$k_{12}(KE_{30}, KE_{31})$	$(K_A<<<60)_R$
4	F (Round2)	$k_2(KE_6, KE_7)$	$(K_A<<<0)_R$	17	FL	$k_{13}(KE_{32}, KE_{33})$	$(K_L<<<77)_L$
5	F (Round3)	$k_3(KE_8, KE_9)$	$(K_L<<<15)_L$	18	FL^{-1}	$k_{14}(KE_{34}, KE_{35})$	$(K_L<<<77)_R$
6	F (Round4)	$k_4(KE_{10}, KE_{11})$	$(K_L<<<15)_R$	19	F (Round13)	$k_{13}(KE_{36}, KE_{37})$	$(K_L<<<94)_L$
7	F (Round5)	$k_5(KE_{12}, KE_{13})$	$(K_A<<<15)_L$	20	F (Round14)	$k_{14}(KE_{38}, KE_{39})$	$(K_L<<<94)_R$
8	F (Round6)	$k_6(KE_{14}, KE_{15})$	$(K_A<<<15)_R$	21	F (Round15)	$k_{15}(KE_{40}, KE_{41})$	$(K_A<<<94)_L$
9	FL	$kl_1(KE_{16}, KE_{17})$	$(K_A<<<30)_L$	22	F (Round16)	$k_{16}(KE_{42}, KE_{43})$	$(K_A<<<94)_R$
10	FL^{-1}	$kl_2(KE_{18}, KE_{19})$	$(K_A<<<30)_R$	23	F (Round17)	$k_{17}(KE_{44}, KE_{45})$	$(K_L<<<111)_L$
11	F (Round7)	$k_7(KE_{20}, KE_{21})$	$(K_L<<<45)_L$	24	F (Round18)	$k_{18}(KE_{46}, KE_{47})$	$(K_L<<<111)_R$
12	F (Round8)	$k_8(KE_{22}, KE_{23})$	$(K_L<<<45)_R$	25	Postwhitening	$kw_3(KE_{48}, KE_{49})$	$(K_A<<<111)_L$
13	F (Round9)	$k_9(KE_{24}, KE_{25})$	$(K_A<<<45)_L$	26	Postwhitening	$kw_4(KE_{50}, KE_{51})$	$(K_A<<<111)_R$

Table 3. Sub-keys for 192/256-bit keys

NO	Operation	subkey	value	NO	Operation	subkey	value
1	Prewhitening	$kw_1(KE_0, KE_1)$	$(K_L<<<0)_L$	18	FL^{-1}	$k_{14}(KE_{34}, KE_{35})$	$(K_L<<<60)_R$
2	Prewhitening	$kw_2(KE_2, KE_3)$	$(K_L<<<0)_R$	19	F (Round13)	$k_{13}(KE_{36}, KE_{37})$	$(K_R<<<60)_L$
3	F (Round1)	$k_1(KE_4, KE_5)$	$(K_B<<<0)_L$	20	F (Round14)	$k_{14}(KE_{38}, KE_{39})$	$(K_R<<<60)_R$
4	F (Round2)	$k_2(KE_6, KE_7)$	$(K_B<<<0)_R$	21	F (Round15)	$k_{15}(KE_{40}, KE_{41})$	$(K_B<<<60)_L$
5	F (Round3)	$k_3(KE_8, KE_9)$	$(K_R<<<15)_L$	22	F (Round16)	$k_{16}(KE_{42}, KE_{43})$	$(K_B<<<60)_R$
6	F (Round4)	$k_4(KE_{10}, KE_{11})$	$(K_R<<<15)_R$	23	F (Round17)	$k_{17}(KE_{44}, KE_{45})$	$(K_L<<<77)_L$
7	F (Round5)	$k_5(KE_{12}, KE_{13})$	$(K_A<<<15)_L$	24	F (Round18)	$k_{18}(KE_{46}, KE_{47})$	$(K_L<<<77)_R$
8	F (Round6)	$k_6(KE_{14}, KE_{15})$	$(K_A<<<15)_R$	25	FL	$kw_3(KE_{48}, KE_{49})$	$(K_A<<<77)_L$
9	FL	$kl_1(KE_{16}, KE_{17})$	$(K_R<<<30)_L$	26	FL^{-1}	$kw_4(KE_{50}, KE_{51})$	$(K_A<<<77)_R$
10	FL^{-1}	$kl_2(KE_{18}, KE_{19})$	$(K_R<<<30)_R$	27	F (Round19)	$k_{19}(KE_{52}, KE_{53})$	$(K_R<<<94)_L$
11	F (Round7)	$k_7(KE_{20}, KE_{21})$	$(K_B<<<30)_L$	28	F (Round20)	$k_{20}(KE_{54}, KE_{55})$	$(K_R<<<94)_R$
12	F (Round8)	$k_8(KE_{22}, KE_{23})$	$(K_B<<<30)_R$	29	F (Round21)	$k_{21}(KE_{56}, KE_{57})$	$(K_A<<<94)_L$
13	F (Round9)	$k_9(KE_{24}, KE_{25})$	$(K_L<<<45)_L$	30	F (Round22)	$k_{22}(KE_{58}, KE_{59})$	$(K_A<<<94)_R$
14	F (Round10)	$k_{10}(KE_{26}, KE_{27})$	$(K_L<<<45)_R$	31	F (Round23)	$k_{23}(KE_{60}, KE_{61})$	$(K_L<<<111)_L$
15	F (Round11)	$k_{11}(KE_{28}, KE_{29})$	$(K_A<<<45)_L$	32	F (Round24)	$k_{24}(KE_{62}, KE_{63})$	$(K_L<<<111)_R$
16	F (Round12)	$k_{12}(KE_{30}, KE_{31})$	$(K_A<<<45)_R$	33	Postwhitening	$kw_3(KE_{64}, KE_{65})$	$(K_B<<<111)_L$
17	FL	$k_{13}(KE_{32}, KE_{33})$	$(K_L<<<60)_L$	34	Postwhitening	$kw_4(KE_{66}, KE_{67})$	$(K_B<<<111)_R$

3 Attack Model

3.1 General Attack Model

Modern block ciphers usually use many large S-boxes to access Cache so as to improve the encrypt/decrypt efficiency. During S-box lookup procedure, it always has the following formula:

$$\alpha \odot \beta = \gamma \tag{1}$$

Then, formula (1) can be transferred to:

$$\alpha \odot \gamma = \beta \tag{2}$$

α : Part of the plaintext (first round analysis) or ciphertext (last round analysis), even the known inner state.

β : Parameter related with the key, usually an expression composed of a set of key and expand-key (*K*,*E*).

γ : Parameter related with the S-box lookup indices or results.

$\odot$: Denote one or more certain logical operations between α and β , such as $\oplus$ for AES, SMS4, and Camellia.

From Section 2.2, we know that the adversary can get γ through the measuring phase of Cache timing attack, as α is usually known, then, it's not difficult to compute β , finally predict the correct key *K*. It usually adopts the following analysis strategy:

Strategy 1: Analyze the un-accessed Cache addresses of S-box lookup

Suppose the adversary get the impossible value of γ , α sometimes is known, thus he can get the impossible value of β , combing analyzing the relations between β and *K*, then deduce the impossible value of *K* and finally recover the key.

Strategy 2: Analyze the accessed Cache addresses of S-box lookup

First the adversary fixes value α , but generates different values of plaintext *P* or ciphertext *C*, so the Cache access profiles are different, as we know that β is fixed, if the adversary can get the accessed Cache traces, then predict some candidates of γ , from formula (2), he can directly get a possible set of β candidates, the correct candidate β is always belong to this set, then fixes another value for α , using the same methods above to improve the predict frequency of correct β , after several times of analyzing, the one with the highest frequency is always the correct value for β . Combing analyzing the relations between β and the key *K*, correct key *K* can also be predicted.

As every un-accessed Cache set is related with δ (usually δ=16) impossible S-box lookup indices and results, using analysis strategy 1 the adversary can eliminate δ impossible β candidates, it's quite effective. In the following paper, we adopt the strategy 1 to analyze the Camellia F function.

3.2 Camellia F Function Attack Model

After analyzing on Camellia algorithm, it's clear to see that only F function executes 8 times S-box lookup operations, specifically speaking, 2 times for every 4 S-boxes during one F function call. Fig 3 displays C code of F function of Camellia in OPENSSL-1.0.0-beta3.

```
#define Camellia_Feistel(_s0,_s1,_s2,_s3,_key) do {\
    1    register u32 _t0,_t1,_t2,_t3;\
    2    \
    3    _t0   = _s0 ^ (_key)[0];\
    4    _t3   = S4[_t0&0xff];\
    5    _t1   = _s1 ^ (_key)[1];\
    6    _t3 ^= S3[(_t0 >> 8)&0xff];\
    7    _t2   = S1 [_t1&0xff];\
    8    _t3 ^= S2[(_t0 >> 16)&0xff];\
    9    _t2 ^= S4[(_t1 >> 8)&0xff];\
    10   _t3 ^= S1[(_t0 >> 24)];\
    11   _t2 ^= _t3;\
    12   _t3   = RightRotate(_t3,8);\
    13   _t2 ^= S3[(_t1 >> 16)&0xff];\
    14   _s3 ^= _t3;\
    15   _t2 ^= S2[(_t1 >> 24)];\
    16   _s2 ^= _t2; \
    17   _s3 ^= _t2;\
    } while(0)
```

Fig.3 The C code of Camellia F function in openssl-1.0.0-beta3

According to the specification of Camellia, we suppose that the input _s0,_s1,_s2,_s3 can be expressed as:

$$\_s=\alpha\oplus\beta_1 \tag{3}$$

$\_s$: _s0,_s1,_s2,_s3; α : part of the plaintext or some known inner state; β_1 : parameter related with *K* or *KE*;

From Code 4,6,7,8,9,10,13,15 line, we can clear to see that the index of S_n entry can be expressed as:

$$\gamma = \varphi(\ \alpha \oplus \beta_1 \oplus \beta_2, n) \tag{4}$$

φ denotes a function to return the *n-st* byte of a 32-bit value; γ denotes the index of S_n entry.

The adversary can gather γ through Cache timing attack measuring phase, α and n is also known, so the candidates for $\beta_1 \oplus \beta_2$ can be predicted. If the encrypt plaintext changed, γ and α is changed, so $\beta_1 \oplus \beta_2$ can be predicted more efficiently.

The adversary can finally recover the correct $\beta_1 \oplus \beta_2$ value, and use it for further analysis.

4 Camellia-128 Attack

4.1 First Round Attack

First the plaintext P is XORed with the 128-bit encryption key K (also KE_0, KE_1, KE_2, KE_3), the output is X. Before X is fed to the Feistel network, it is separated into two 64-bit data blocks. The left half X_L goes into the F function together with the 64-bit round key (KE_4, KE_5) and the output of the F function is XORed with the right half block X_R. At the end of F function, the right and left half block will be exchanged.

According to Fig.1 and formula (4), we can see that X can be expressed as $P \oplus K$, so obviously, β_1 in formula (3) and (4) can be expressed as KE_0, KE_1 in the first round Camellia F function call. Then, we have 8 equations as follows:

$$\begin{array}{ll} \gamma_1 = P_0 \oplus KE_{0,0} \oplus KE_{4,0} & \gamma_1 = P_7 \oplus KE_{1,3} \oplus KE_{5,3} \\ \gamma_2 = P_1 \oplus KE_{0,1} \oplus KE_{4,1} & \gamma_2 = P_4 \oplus KE_{1,0} \oplus KE_{5,0} \\ \gamma_3 = P_2 \oplus KE_{0,2} \oplus KE_{4,2} & \gamma_3 = P_5 \oplus KE_{1,1} \oplus KE_{5,1} \\ \gamma_4 = P_3 \oplus KE_{0,3} \oplus KE_{4,3} & \gamma_4 = P_6 \oplus KE_{1,2} \oplus KE_{5,2} \end{array} \tag{5}$$

γ_n denotes the index of S$_n$ entry, its candidates can be gathered from measuring phase, suppose we get the impossible value of γ_n, as P is known, so according to formula (5), we can get the impossible value for $KE_0 \oplus KE_4$ and $KE_1 \oplus KE_5$, after analysis of more samples, finally recover the correct value for $KE_0 \oplus KE_4$ and $KE_1 \oplus KE_5$.

If $KE_0 \oplus KE_4$ and $KE_1 \oplus KE_5$ is known after analyzing, as P is known, we can get every S-box lookup index γ, so obviously, we can also get every S-box lookup result. From the F function code of Fig 3, we can see that the output s$^1{}_2$, s$^1{}_3$ can be expresses as the input s$^1{}_2$, s$^1{}_3$ XORed with many S-box lookup results. So, finally, the adversary can know that the output s$^1{}_2$, s$^1{}_3$ can be expressed the input s$^1{}_2$, s$^1{}_3$ XORed with a known variable $c1$, output s$^1{}_2$, s$^1{}_3$ can be expressed as follows:

$$\begin{aligned} s^1{}_2 = s^1{}_2 \,\hat{}\, & S_2[P_4 \oplus KE_{1,0} \oplus KE_{5,0}] \,\hat{}\, S_3[P_5 \oplus KE_{1,1} \oplus KE_{5,1}] \,\hat{}\, S_4[P_6 \oplus KE_{1,2} \oplus KE_{5,2}] \,\hat{}\, \\ & S_1[P_7 \oplus KE_{1,3} \oplus KE_{5,3}] \,\hat{}\, S_1[P_0 \oplus KE_{0,0} \oplus KE_{4,0}] \,\hat{}\, S_2[P_1 \oplus KE_{0,1} \oplus KE_{4,1}] \,\hat{}\, \\ & S_3[P_2 \oplus KE_{0,2} \oplus KE_{4,2}] \,\hat{}\, S_4[P_3 \oplus KE_{0,3} \oplus KE_{4,3}] \\ s^1{}_3 = s^1{}_3 \,\hat{}\, & S_2[P_4 \oplus KE_{1,0} \oplus KE_{5,0}] \,\hat{}\, S_3[P_5 \oplus KE_{1,1} \oplus KE_{5,1}] \,\hat{}\, S_4[P_6 \oplus KE_{1,2} \oplus KE_{5,2}] \,\hat{}\, \\ & S_1[P_7 \oplus KE_{1,3} \oplus KE_{5,3}] \,\hat{}\, S_1[P_0 \oplus KE_{0,0} \oplus KE_{4,0}] \,\hat{}\, S_2[P_1 \oplus KE_{0,1} \oplus KE_{4,1}] \,\hat{}\, \\ & S_3[P_2 \oplus KE_{0,2} \oplus KE_{4,2}] \,\hat{}\, S_4[P_3 \oplus KE_{0,3} \oplus KE_{4,3}] \,\hat{}\, \mathrm{RightRotate}(S_1[P_0 \oplus KE_{0,0} \oplus KE_{4,0}] \,\hat{}\, \\ & S_2[P_1 \oplus KE_{0,1} \oplus KE_{4,1}] \,\hat{}\, S_3[P_2 \oplus KE_{0,2} \oplus KE_{4,2}] \,\hat{}\, S_4[P_3 \oplus KE_{0,3} \oplus KE_{4,3}], 8) \end{aligned} \tag{6}$$

So, after analyzing the first round Camellia F function call, we can get $KE_0 \oplus KE_4$, $KE_1 \oplus KE_5$ $((K_L <<< 0)_L \oplus (K_B <<< 0)_L)$ and compute the output s$^1{}_2$, s$^1{}_3$, it's clear to see that we can't recover the $KE_0, KE_1((K_L <<< 0)_L)$ directly.

4.2 Second Round Attack

From Section 4.1 we know that, during the first round F function, the output s$^1{}_0$, s$^1{}_1$ are the same as the input s$^1{}_0$, s$^1{}_1$, the output s$^1{}_2$, s$^1{}_3$ can be expressed the input s$^1{}_2$, s$^1{}_3$ XORed with a known variable c, finally the left and right half block (s$^1{}_0$, s$^1{}_{1,}$ s$^1{}_2$, s$^1{}_3$) will be exchanged as the input of the second round F function.(s^1_2, s^1_3, s^1_0, s^1_1, also can be expressed as s^2_0, s^2_1, s^2_2, s^2_3). Note that the second round F function uses the other 64-bit round key (KE_6, KE_7).

Also, according to Fig.3 and formula (4), we have 8 equations as follows:

$$\begin{aligned} \gamma_1 &= P_8 \oplus KE_{2,0} \oplus KE_{6,0} & \gamma_1 &= P_{15} \oplus KE_{3,3} \oplus KE_{7,3} \\ \gamma_2 &= P_9 \oplus KE_{2,1} \oplus KE_{6,1} & \gamma_2 &= P_{12} \oplus KE_{3,0} \oplus KE_{7,0} \\ \gamma_3 &= P_{10} \oplus KE_{2,2} \oplus KE_{6,2} & \gamma_3 &= P_{13} \oplus KE_{3,1} \oplus KE_{7,1} \\ \gamma_4 &= P_{11} \oplus KE_{2,3} \oplus KE_{6,3} & \gamma_4 &= P_{14} \oplus KE_{3,2} \oplus KE_{7,2} \end{aligned} \tag{7}$$

Using attack model in Section 3.2, we can recover the correct value for $KE_2 \oplus KE_6$ and $KE_3 \oplus KE_7$, then compute the output s^2_2, s^2_3 can be expressed the input s^2_2, s^2_3 XORed with a known variable $c2$.

$$\begin{aligned} s^2{}_2 = s^2{}_2 \,^\wedge S_2[P_{12} \oplus KE_{3,0} \oplus KE_{7,0}] \,^\wedge S_3[P_{13} \oplus KE_{3,1} \oplus KE_{7,1}] \,^\wedge S_4[P_{14} \oplus KE_{3,2} \oplus KE_{7,2}] \,^\wedge \\ S_1[P_{15} \oplus KE_{3,3} \oplus KE_{7,3}] \,^\wedge S_1[P_8 \oplus KE_{2,0} \oplus KE_{6,0}] \,^\wedge S_2[P_9 \oplus KE_{2,1} \oplus KE_{6,1}] \,^\wedge \\ S_3[P_{10} \oplus KE_{2,2} \oplus KE_{5,2}] \,^\wedge S_4[P_{11} \oplus KE_{2,3} \oplus KE_{6,3}] \\ s^2{}_3 = s^2{}_3 \,^\wedge S_2[P_{12} \oplus KE_{3,0} \oplus KE_{7,0}] \,^\wedge S_3[P_{13} \oplus KE_{3,1} \oplus KE_{7,1}] \,^\wedge S_4[P_{14} \oplus KE_{3,2} \oplus KE_{7,2}] \,^\wedge \\ S_1[P_{15} \oplus KE_{3,3} \oplus KE_{7,3}] \,^\wedge S_1[P_8 \oplus KE_{2,0} \oplus KE_{6,0}] \,^\wedge S_2[P_9 \oplus KE_{2,1} \oplus KE_{6,1}] \,^\wedge \\ S_3[P_{10} \oplus KE_{2,2} \oplus KE_{5,2}] \,^\wedge S_4[P_{11} \oplus KE_{2,3} \oplus KE_{6,3}] \,^\wedge \mathrm{RightRotate}(S_1[P_8 \oplus KE_{2,0} \oplus KE_{6,0}] \,^\wedge \\ S_2[P_9 \oplus KE_{2,1} \oplus KE_{6,1}] \,^\wedge S_3[P_{10} \oplus KE_{2,2} \oplus KE_{5,2}] \,^\wedge S_4[P_{11} \oplus KE_{2,3} \oplus KE_{6,3}], 8) \end{aligned} \tag{8}$$

After analyzing the second round Camellia F function call, we can get $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$ ($((K_L <<< 0)_R \oplus (K_B <<< 0)_R$) and compute the output s^2_2, s^3_3, but we can't recover KE_2 ,KE_3 $((K_L <<< 0)_R)$. In order to recover part or full key *K*, we have to expand Cache timing attack to further rounds.

4.3 Further Rounds Attack

In order to further reduce the searching space for *K*, we need to get more information about *K* and *KE*. So, this section will display how to get more information about *K* and *KE*, but note that we still can not get *K* directly from attacking the encryption procedure.

From section 4.1 we know that, the input $S^{r+1}{}_0$, $S^{r+1}{}_1$ can be expressed as the input $S^r{}_2$, $S^r{}_3$ XORed with a known variable c_r(a value we can deduce from the *r-st* round attack). As the initial two rounds input $S^r{}_0$, $S^r{}_1$(r=1,2) is X(X=$P \oplus K$), so if *r* is an odd number, the input $S^{r+1}{}_0$, $S^{r+1}{}_1$ can be expressed as the left half X_L XORed with a known variable C_1^…^C_r(*r* is an odd number); if *r* is an even number, the input $S^{r+1}{}_0$, $S^{r+1}{}_1$ can be expressed as the right half X_R XORed with a known variable C_2^…^C_r. But note that Camellia with FL/FL^{-1} functions inserted every 6 rounds are used to provide non-regularity between the rounds, so it's very hard for the adversary to compute the input $S^r{}_0$, $S^r{}_1$ (r>6) as *X* XORed with *c*. But for Camellia without FL/FL^{-1} function, the adversary can compute the input $S^r{}_0$, $S^r{}_1$ (r=1, …, 18) as *X* XORed with *c*.

According to formula (4), after analyzing on first 6 rounds F function calls (Camellia with FL/FL^{-1}), what we can get is shown in Table 4.

Table 4. Attack results of first 6 rounds Camellia-128

Round	Result	Round	Result
1	$KE_0 \oplus KE_4 \Vert KE_1 \oplus KE_5$ $((K_L{<<<}0)_L \oplus (K_A{<<<}0)_L)$	4	$KE_2 \oplus KE_{10} \Vert KE_3 \oplus KE_{11}$ $((K_L{<<<}0)_R \oplus (K_L{<<<}15)_R)$
2	$KE_2 \oplus KE_6 \Vert KE_3 \oplus KE_7$ $((K_L{<<<}0)_R \oplus (K_A{<<<}0)_R)$	5	$KE_0 \oplus KE_{12} \Vert KE_1 \oplus KE_{13}$ $((K_L{<<<}0)_L \oplus (K_A{<<<}15)_L)$
3	$KE_0 \oplus KE_8 \Vert KE_1 \oplus KE_9$ $((K_L{<<<}0)_L \oplus (K_L{<<<}15)_L)$	6	$KE_2 \oplus KE_{14} \Vert KE_3 \oplus KE_{15}$ $((K_L{<<<}0)_R \oplus (K_A{<<<}15)_R)$

4.4 Key Prediction

From the first 4 rounds attack, we can get C=(K_L<<<0) ⊕ (K_L<<<15)(C is a 128-bit value), this is enough for us to recover K_L. Specific analysis method is as follows:

```
K_L Searching Algorithm: SearchingK_L(S_K,C)
unsigned char K_P[128],cTemp
S_K ← ø
Foreach i from 0x00 to 0x01
{
  K_P[0] ← i
```

```
    Foreach $j$ from 0 to 127
    {
      cTemp ← $(K_P[(15*j)\%128]$ ^ $C[(15*j)\%128])\&0x01$
      If($j$!=127)
        $K_P[(15*(j+1))\%128]$ ← cTemp;
      Else if($j$==127)
      {
        If(cTemp==$K_P[0]$)
        Add $K_P$ to $S_K$
      }
    }
  }
```

Using Algorithm above, we can get at most 2 candidates of K_L, sometimes even directly just get the unique value of K_L.

4.5 Experiment

We have implemented our Camellia access driven cache timing attack against OPENSSL-1.0.0-beta3 running on AMD 64 machine, and use RDTSC instruction to measure high-precision receipt timestamp obtained from the CPU's cycle counter in terms of clock cycles. To have a initial “clean” testing environment, we started out using OPENSSL library calls as black-box functions, pretending we have no access to the key. Fig.4 displays the relationship between $KE_0 \oplus KE_4$ key byte searching space and sample size. It’s obviously to see that more than 400 samples are enough to recover every key byte of $KE_0 \oplus KE_4$.

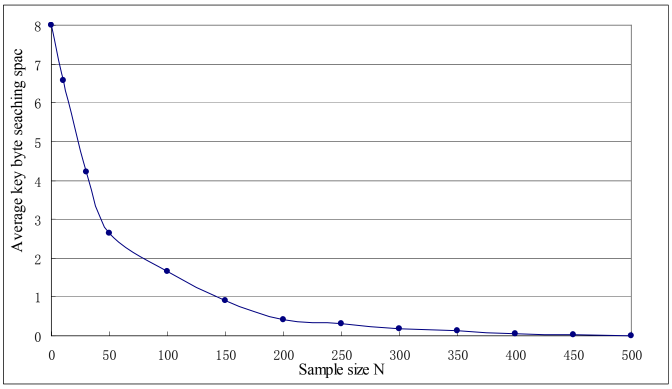


Fig.4 Average $KE_0 \oplus KE_4$ key byte searching space and Attack sample size

After we get the first 4 rounds attack results in Table 4, we can analyze the encryption key with the method of Section 4.4, then recover the encryption key. Our experiments demonstrate that for about 500 samples are enough to recover the full Camellia-128 key.

5 Camellia-192/256 Attack

The encryption procedure for Camellia-192/256 is almost the same as Camellia-128, the only thing different is that instead of calling 18 times of F function and 2 times FL/FL^{-1} function, Camellia-192/256 calls 24 times of F function and 3 times FL/FL^{-1} function. The key schedule for Camellia-192/256 is more complicating than Camellia-128. Methods on analysis the key schedule of Camellia-128 are not applicable to Camellia-192/256. Does this means Camellia-192/256 is much more secure than Camellia-128?

The answer is No! Just because of the particular design of Camellia-192/256 key schedule, it’s quite easy to recover the key though a simple analysis after first 6 rounds Cache timing attack. In order to emphasize the strong applicability of our attack on Camellia-192/256, we take Camellia-192/256 with FL/FL^{-1} as our target, which is the most secure algorithm currently, and we just analyze the first 6 rounds of Camellia encryption and the key schedule part.

5.1 First 6 rounds Attack

Using methods of Section 4, after attacking the first 6 rounds of Camellia-192/256, what we can get is shown in Table 5.

Table 5. Attack results of first 6 rounds Camellia-192/256

Round	Result	Round	Result
1	$KE_0 \oplus KE_4\|\|KE_1 \oplus KE_5$ $((K_L<<<0)_L \oplus (K_B<<<0)_L)$	4	$KE_2 \oplus KE_{10}\|\|KE_3 \oplus KE_{11}$ $((K_L<<<0)_R \oplus (K_R<<<15)_R)$
2	$KE_2 \oplus KE_6\|\|KE_3 \oplus KE_7$ $((K_L<<<0)_R \oplus (K_B<<<0)_R)$	5	$KE_0 \oplus KE_{12}\|\| KE_1 \oplus KE_{13}$ $((K_L<<<0)_L \oplus (K_A<<<15)_L)$
3	$KE_0 \oplus KE_8\|\|KE_1 \oplus KE_9$ $((K_L<<<0)_L \oplus (K_R<<<15)_L)$	6	$KE_2 \oplus KE_{14}\|\| KE_3 \oplus KE_{15}$ $((K_L<<<0)_R \oplus (K_A<<<15)_R)$

5.2 Key Schedule Analysis

From Table 5, after first 6 rounds attack, we can get $(K_A{<}{<}{<}15) \oplus (K_R{<}{<}{<}15)$, then recover $K_A \oplus K_R$, which is also the input state of the 5-th F function of Camellia key schedule, as $\sum_5$ is known, so after the 5-th and 6-th F function call, we can get the correct value K_B.

K_B is also the value of (KE_4, KE_5, KE_6, KE_7), as from Table 5, we can recover $KE_0 \oplus KE_4$, $KE1 \oplus KE_5$, $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$, so obviously, we can recover KE_0, $KE1$, KE_2, KE_3, which is also the left 128-bit of the encryption key K, as $KE_0 \oplus KE_8$, $KE1 \oplus KE_9$, $KE_2 \oplus KE_{10}$, $KE_3 \oplus KE_{11}$ is also known to us, so we can deduce the correct value of KE_8, KE_9, KE_{10}, KE_{11}, finally, K_R can be recovered.

From Section 2.3, we know that the right 64-bit key of Camellia-192 equals to the left 64-bit of K_R, and the right 128-bit key of Camellia-256 equals to K_R. As K_R is known, the right 64-bit key value of Camellia-192 and right 128-bit key of Camellia-256 is also recovered. Combing previous left 128-bit key value, we can recover full key of Camellia-192/256.

5.3 Experiment

Fig.5 displays the relationship between $KE_0 \oplus KE_4$ key byte searching space and sample size for Camellia 128 and 192/256. It's obviously to see that more than 600 samples are enough to recover every key byte of $KE_0 \oplus KE_4$ of Camellia-192/256, which is bigger than 400 of Camellia-128.

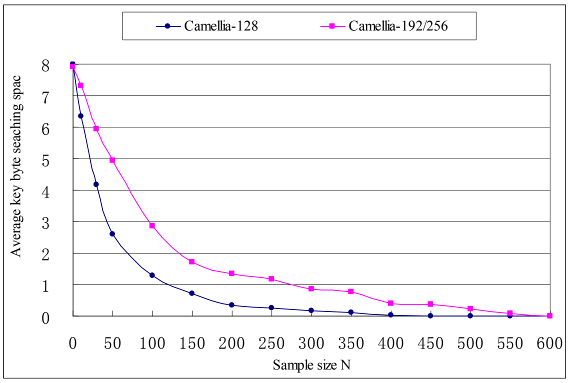


Fig.5 Average $KE_0 \oplus KE_4$ key byte searching space and Attack sample size for Camellia 128 and 192/256

After we get the first 6 rounds attack results in Table 5, we can analyze the encryption key with the method of Section 5.2, then recover the encryption key. Our experiments demonstrate that for about 800-900 samples are enough to recover the full Camellia-192/256 key.

6 Known Ciphertext Attack

In previous Section 4,5, due to the relationships among known plaintext, some accessed Cache set addresses of Camellia, the encryption key, we provided several known plaintext Cache timing attacks on Camellia. As we all known, Camellia is a symmetric block cipher, so due to the relationship among known ciphertext, Camellia accessed Cache set addresses, the encryption key, it's quite possible to design some known ciphertext attacks on Camellia.

From formula (4) in Section 3.2, we know that the input of the F function($\alpha \oplus \beta_1$) XORed with an expand-key β_2 is the input index of a single lookup table, which means that it's possible to recover $\beta_1 \oplus \beta_2$ by single lookup table related Cache sets γ and α . Then the output of F function can be expressed as $\alpha \oplus \beta_1$ XORed with c, c is the result of multi table lookup contents XORed with each other. As to the last round of Camellia encryption, the output is the known ciphertext, but parameter c is unknown, so it's impossible to recover $\beta_1 \oplus \beta_2$. But note that due to the symmetric feature of block cipher, we can attack the Camellia decrypt procedure. So, the input of the decrypt procedure is the ciphertext, just the same as the plaintext in the encrypt procedure. In this section we will show some known ciphertext attacks on Camellia.

6.1 Camellia-128 Attack

If we have measured the Cache profiles of Camllia-128 decrypt procedure, according to Formula (4), during the first two rounds decryption procedure analysis, the ciphertext is α , we can recover $KE_{48} \oplus KE_{44}||KE_{49} \oplus KE_{45}||$ $KE_{50} \oplus KE_{46}||KE_{51} \oplus KE_{47}$ and compute the following rounds F function input α , what we can get is shown in Table 6. Then, according to Section 4, if we analysis the 3rd and 4th decryption rounds attack results, we can recover at most 2 candidates of K_A, as $K_A \oplus K_L$ is also known to us after the first two decryption rounds attack, so we can recover K_L.

Table 6. Attack results of first 6 rounds Camellia-128 decryption

Round	Result	Round	Result
1	$KE_{48} \oplus KE_{44} \Vert KE_{49} \oplus KE_{45}$ $((K_A{<}{<}{<}111)_L \oplus (K_L{<}{<}{<}111)_L)$	4	$KE_{50} \oplus KE_{42} \Vert KE_{51} \oplus KE_{43}$ $((K_A{<}{<}{<}111)_R \oplus (K_A{<}{<}{<}94)_R)$
2	$KE_{50} \oplus KE_{46} \Vert KE_{51} \oplus KE_{47}$ $((K_A{<}{<}{<}111)_R \oplus (K_L{<}{<}{<}111)_R)$	5	$KE_{48} \oplus KE_{36} \Vert KE_{49} \oplus KE_{37}$ $((K_A{<}{<}{<}111)_L \oplus (K_L{<}{<}{<}94)_L)$
3	$KE_{48} \oplus KE_{40} \Vert KE_{49} \oplus KE_{41}$ $((K_A{<}{<}{<}111)_L \oplus (K_A{<}{<}{<}94)_L)$	6	$KE_{50} \oplus KE_{38} \Vert KE_{51} \oplus KE_{39}$ $((K_A{<}{<}{<}111)_L \oplus (K_L{<}{<}{<}94)_L)$

6.2 Camellia-192/256 Attack

If we measured the Cache profiles of Camllia-192/256 decrypt procedure, according to Formula (4), during the first two rounds decryption procedure analysis, the ciphertext is α , we can recover $KE_{64} \oplus KE_{60} \| KE_{65} \oplus KE_{61} \|$ $KE_{66} \oplus KE_{62} \| KE_{67} \oplus KE_{63}$ and compute the following rounds F function input α , what we can get is shown in Table 7. We can also recover $K_A \oplus K_R$, using methods in Section 5.2, we can get K_B,K_L,K_R, finally recover full Camllia-192/256 key K.

Table 7. Attack results of first 6 rounds Camellia-192/256 decryption

Round	Basic deduce result	Round	Basic deduce result
1	$KE_{64} \oplus KE_{60} \Vert KE_{65} \oplus KE_{61}$ $((K_B{<}{<}{<}111)_L \oplus (K_{\boldsymbol{L}}{<}{<}{<}111)_L)$	4	$KE_{66} \oplus KE_{58} \Vert KE_{67} \oplus KE_{59}$ $((K_{\boldsymbol{B}}{<}{<}{<}111)_R \oplus (K_A{<}{<}{<}94)_R)$
2	$KE_{66} \oplus KE_{62} \Vert KE_{67} \oplus KE_{63}$ $((K_{\boldsymbol{B}}{<}{<}{<}111)_R \oplus (K_{\boldsymbol{L}}{<}{<}{<}111)_R)$	5	$KE_{64} \oplus KE_{52} \vert KE_{65} \oplus KE_{53}$ $((K_{\boldsymbol{B}}{<}{<}{<}111)_L \oplus (K_R{<}{<}{<}94)_L)$
3	$KE_{64} \oplus KE_{56} \Vert KE_{65} \oplus KE_{57}$ $((K_{\boldsymbol{B}}{<}{<}{<}111)_L \oplus (K_A{<}{<}{<}94)_L)$	6	$KE_{66} \oplus KE_{54} \Vert KE_{67} \oplus KE_{55}$ $((K_{\boldsymbol{B}}{<}{<}{<}111)_L \oplus (K_R{<}{<}{<}94)_L)$

6.3 Experiment

We have implemented several known ciphertext attacks on Camellia-128/192/256(with and without FL/FL^{-1}), using the analysis methods in Section 6.1 and 6.2, for about 500 samples are enough to recover full Camellia-128 key, and 900 samples are enough to recover Camellia-192/256 key, and all the analysis procedure can be done within 1 second.

7 Remote Attack

In local attack experiments, we started out using OPENSSL library calls as black-box functions, pretending we have no access to the key, there is no real interaction between the attacker program and Camellia server program. In order to demonstrate our attack's strong applicability, we have implemented it into the remote environment, such as local and campus network, and it works quite efficiently.

There are three programs in the remote attack experiment: the attacker (AP), the Camellia server (CSP) and the unprivileged spy program (SP), SP and CSP are deployed on the same computer, SP just executes simple access self data to access Cache, which is possible on the vast development of spy technologies.

Step 1: AP notifies SP to access self data clearing the data in Cache, initiating the Cache into a known state

Step 2: AP sends encryption request to CSP and trigger Camellia encryption. After encryption, CSP sends the ciphertext to AP.

Step3: AP notifies SP to revisit Cache and find out which Cache set CSP has accessed and un-accessed.

Step4: Using certain analysis model, AP can make some offline analysis on encryption Key.

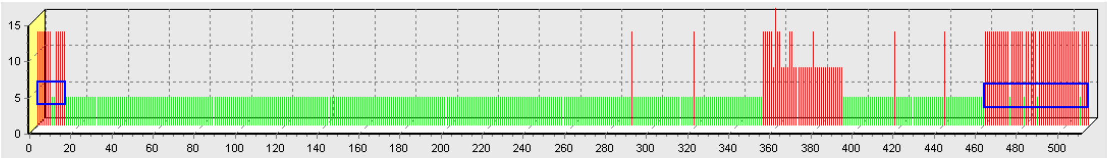


Fig.6 One sample measurement in Local Attack

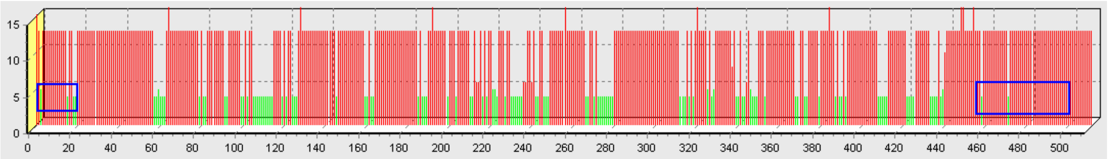


Fig.7 One sample measurement in remote Attack

During the remote attacks, we find out that, due to the frequent network interaction among AP, CSP, SP, there are many sending and receiving packet operations to access data Cache, so there are much more noises than local experiment condition. One sample measurement of local attack and remote attack is shown in Fig.6 and Fig.7, the horizontal axis denotes the 512 Cache sets, and the vertical axis is the accessed time cycles of each Cache set by SP. It's clear to see layout of Camellia's 4 lookup tables (Cache set

461,477,493,509), measurement in remote attack has rather more noises than local attack, but we can still gather the un-accessed Cache sets(less time cycles ones) during Camellia encryption, which is enough for us to recover the Camellia key. Finally, using analysis methods in Section 4 and 5, we successfully recover full Camellia 128/192/256 key for about 3000 samples in both local and campus network environment.

8 Discussions and Countermeasures

8.1 Discussions

In summary, the F function of the Camellia can leak information about the result of encryption key XORed with expand-key; the left circular rotating operation of the key schedule in Camellia has serious designing problem.

Comparisons between different Cache timing attacks on ciphers

We have tested Cache timing attacks on different symmetric ciphers, such as AES, SMS4 and Camellia, and proved that our attack model of Section 3.1 works quite efficiently.

1 AES[41]: During the first AES round, the input index of the lookup table is the plaintext XORed with the initial key, once we have gathered that Cache trace profiles of AES encryption procedure, the initial key can be recovered directly. During the last AES round, the output of the lookup table XORed with the expand key is just the ciphertext, once we have gathered the Cache profiles, the expand key can be recovered, due to the invertible key expansion structure, the initial AES key can be recovered very easily.

2 SMS4[42]: During the first SMS4 round, the input index of the lookup table is 3 plaintext bytes XORed with the 32-bit key rk_0, once we gathered the Cache profiles, rk_0 can be recovered, then we can iterated rk_0 into the first round encryption and recovered the output of the first round, which also is the input of the second round, using the same methods, we can recover rk_1, rk_2, rk_3, so the full 128-bit SMS4 initial key can be recovered. During the last SMS4 round, the output of the lookup table XORed with the expand key rk_{31} is just 32-bit partial ciphertext, once we have gathered the Cache profiles, then we can iterated rk_{31} into the last round and compute the input of the last round, which also is the output of the 31 round, using the same methods to analysis 29,30,31 round, we can recover rk_{28},rk_{29},rk_{30}. Also, due to the invertible key expansion structure, the initial SMS4 key can be recovered very easily.

3 Camellia: Comparably speaking, attacking Camellia is much more complicated than AES and SMS4, because unlike AES and SMS4, the input index of the Camellia lookup table is related with two secret variables (the encryption key XORed with expand-key), so we can't recover the initial key directly and have to make some further analysis of the key schedule to recover the key. But thanks to the left circular rotating operation of the Camellia key schedule, it enables us to recovered full Camellia key very efficiently.

Why and how does this attack works?

The dis-alignment feature of Camellia S-boxes in Cache enables us to recover full secret key about $K_i \oplus KE_j$, and the simple left circular rotating operation of the key schedule provides us an efficient way to recover Camellia encryption key.

1 The dis-alignment feature of Camellia S-boxes in Cache:

Suppose O denotes the offset of the first Camellia lookup table element in the Cache block, and the Cache block size is 64 bytes, then we can have O=0 and O≠0 two cases.

a O=0: In this case, each lookup table line is perfected aligned in one Cache set. Every un-accessed Cache set can eliminate 16 high 4-hit identical and low 4-bit different but consecutive key bytes, as the correct key byte is impossible to be eliminated, so the 15 other candidates belongs to the same Cache set as the correct key byte is also impossible to be eliminated, in totally, there will be 16 candidates left for each key byte, which are maximal information we can get during one round analysis.

b O≠0: Very interesting things happened in this case, when the first lookup element is dis-aligned in Cache, which also means that each lookup table line is related with two different but consecutive Cache sets and each Cache set is related with two different but consecutive lookup table lines. So every un-accessed Cache set will related two lookup table lines, the first table line related with 16-O indices (high 4 bit identical and low 4 bit different but consecutive), the second table line related with O indices(also high 4 bit identical and low 4 bit different but consecutive), so this Cache set can eliminate 16 both high 4 bit(2 values) and low 4 bit(16 values) different candidates. Increased sample size can enable us to eliminate other 255 wrong key byte candidates and recover the correct key byte. It's easy to find out that O=7 or O =8 has the best elimination results.

From analysis above, we can see that when O=0, it's impossible to recover full key candidates, but fortunately, in most cases, O≠0(the probability is about 15/16 when δ=16), and as to Cache timing attack on Camellia, just this dis-alignment feature enabled us to recover secret key about $K_i \oplus KE_j$.

2 Key schedule design problem:

Many block ciphers used many left circular rotating operations of in the key schedule to generate the sub-keys, such as Camellia and ARIA[43].

As for Camellia-128, sub-keys are generated by left rotating K_L and K_A, and as for Camellia-192/256, sub-keys are generated by left rotating K_L, K_R, K_A, and K_B. Usually, some certain sub-keys are generated by left rotating different bits of the same 128-bit variable, such as $kw_1||kw_2$ (K_L)and $k_3||k_4$ (K_L<<<15) in Table 2. This is quite efficient for sub-key generations, but it also raises some dangerous problems. Once the adversary successfully recovered the XOR results of these sub-keys, it's quite efficient to recover the encryption key using analysis method in Section 4.4.

As to the specific design of Camellia encryption procedure, the FL and FL^{-1} functions are inserted by every 6 rounds to provide non-regularity between the rounds and enhance the security of the cipher. This brought some difficulties for the adversary to attack more rounds on Camellia with FL/FL^{-1}, he may can only attack the first Camellia 6 rounds. But unfortunately, this was enough for further analysis. As to Camellia-128, after just attacking the first 4 rounds, we can find a XOR collision of left rotating different bits of K_L , which enable us to recover the encryption key K_L directly. As to Camellia-192/256, after attacking the 6 rounds, also we can not find any XOR collisions of the same variable (K_L, K_R, K_A, K_B), but we can recover $K_A \oplus K_R$, which is just one input of the 5th F function in Camellia key schedule, this enabled us to directly recover K_B and indirectly recover full encryption key.

The ARIA algorithm is a Korean Standard block cipher, which is optimized for lightweight environments. It is an SPN block cipher with 128/192/256-bit keys, during the encryption procedure, the input index of the four lookup tables is 8-bit state variable XORed with one byte of the 128-bit key rk_r. Using the attack model in this paper, it's quite efficient for the adversary to recover every ARIA sub-key. But it's still difficult to directly recover the encryption key. Thanks to the rotating operations of the ARIA key schedule, the 13 sub-keys of ARIA-128 are generated by four 128-bit variables W_0, W_1, W_2, W_3 , such as ($W_{i\%4}$<<<n) ⊕ ($W_{(i+1)\%4}$ <<<m). After just attack the first 4 rounds, the adversary can found a XOR collision of the same W_0, using analysis methods in Section 4.4, the encryption key W_0 can be recovered efficiently.

So, the cipher designer should take the weaknesses of rotate operation into account during the designing phase of ciphers.

Comparisons between previous remote timing attacks on ciphers

There are only Bernstein[7] and O. Acıiçmez[18] two cases of remote Cache based timing attacks on block cipher in the published papers, both of which are belonged to timing driven Cache attack on AES.

1 Bernstein attack: The attack program and encryption server are deployed on different machines, but it is the encryption server that taking charge of measuring the AES encryption time and sending it back to the attack program, which is impractical in the real environments, so actually speaking, Bernstein attack is a type of local Cache timing attack.

2 O. Acıiçmez remote attack: The attack program and AES server are deployed on the same machines, the attack program is taking charge of measuring the time between sending the plaintext to the server and receiving the ciphertext from the sever, at the condition of eliminating the network transmit delay, successfully recover the full AES key for about $2^{26.66}$ samples, thus verify the correctness of the practicality of remote timing driven Cache attack. In the real condition, the remote network delay even the dithered delay noises are more than the encryption time, it is quite hard to measure the accurate AES encryption time.

The access driven Cache remote timing attacks of this paper put aside the network delay problem at a certain degree, deploys the attack program and the encryption server at different machines in the local network environment, and deploys a spy process running synchronously with the Camellia server to measure the Cache information of the encryption process by executing normally self-data accessing operations, combines the plaintext/ciphertext information to analysis the Camellia key, finally recover the full Camellia-128/192/256 key efficiently, the sample size and time of our attack are much smaller.

8.2 Countermeasures

This section mainly talks about countermeasures against Cache timing attacks from algorithm design. Due to the weakness of F function and <<< left circular rotating operation, following are some advices for cipher designers to make such attacks more difficult.

1 Use multivariable secret keys (at least 3 dimensions) as the input of the S-box lookup.

Block cipher such as AES and SMS4, its S-box lookup input index is related with only one secret information (K_i or KE_j), so through Cache timing analysis, the adversary can directly recover K_i or KE_j. However, this doesn't work in Camellia, as the S-box lookup input index of Camellia encryption procedure can leak secret information ($K_i \oplus KE_j$, full recovery) about two secret key variables K_i and KE_j, the key schedule might leak one secret variable K_i or KE_j (limited candidates). In order to get the correct K_i or KE_j, the adversary has to analyze Camellia key schedule (Camellia-192/256), sometimes even need to attack the key schedule (Camellia-128). So, if Camellia using multivariable secret keys (at least 3 dimensions) as the input of the S-box lookup might be an efficient countermeasure to Cache timing attacks.

2 Insert FL/FL^{-1} functions between F function calls more frequently.

In Camellia, the FL and FL^{-1} functions inserted every 6 rounds are used to provide non-regularity between the rounds so that the security of the cipher is increased and these two functions are similarly constructed by logical operations including AND, OR, XOR, and rotations. Indeed, it's quite effective to improve the security of Camellia against Cache timing attack. In Camellia with FL and FL^{-1} functions, the adversary get nothing from the following rounds after FL and FL^{-1} functions, but just first 4 rounds analysis results of $K_i \oplus KE_j$ (Table 4) is enough to recover Camellia-128 key and first 6 rounds analysis results of $K_i \oplus KE_j$(Table 5) was enough to recover Camellia-192/256 key.

Note that all of the further key analysis is based on the successful attack on first 6 rounds Camellia F function analysis, if we insert *FL/FL*$^{-1}$ functions more frequently between F function call, like 2 rounds instead of current 6 rounds, it would be much harder to apply Cache timing attacks.

3 Use more linear logical operations besides <<< circular rotating operation, such as XOR, mix Columns etc.

It's quite efficient to generate the sub-key KE_j by simply rotating K_L, K_R, K_A, and K_B and taking the left or right-half of them, but it's not a secure way against Cache timing attacks. Once the adversary find a XOR collision about two variables generated by the same secret variable, it's very easy to recover the correct secret variable. Combing the cipher algorithm design, the adversary can recover encryption key quite efficiently. Our experiments demonstrate that just analyzing one <<< 15 operation can narrow down the key searching space from2^{128} to 2.

As so far rotating operation is the only direct connection between two Camellia sub-keys, if we insert more other logical operations like AND, OR, XOR between the generation of different round sub-keys, it will increase the difficulty for Cache timing analysis. As to Camellia-192/256, we specially suggest the cipher designer to rotate different n bits of K_A and K_R during the generation of the 3rd to 6th round sub-key.

4 White-Box Cryptography.

White-box cryptography aims to protect secret keys by embedding them into a software implementation of a block cipher. In such a context, it is assumed that the attacker (usually a legitimate user or malicious software) may also control the execution environment， such as static and dynamic analysis of the implementation, altering of computation, and modification of internal variables. This is in contrast with the more traditional security model where the attacker is only given a black-box access (i.e., inputs/outputs) or given a grey-box access (use side channel information such as power consumption, and timing information) to the cryptographic algorithm under consideration. In White-box Cryptography model, it was mainly composed of the following part: transform the cipher into a network of key dependant lookup tables, randomized behavior of all network nodes, extend the cryptographic border.

Chow et al. introduce this idea and propose a white box implementation of DES in [44] by interleaving affine transformations and using de-linearization techniques. An improvement is explained in [45]. An implementation of AES is also given in [46] by representing it with a set of key-dependent look-up tables, also an improvement is explained in [47]. All of the implementation above (DES represents Feistel structure, AES represents SPN structure.) can prove that this idea can efficient protect secret keys being analyzed by hidden encryption key into the S-box of a block cipher, thus can defend almost all of the current side channel analysis on block ciphers. But although the security improves, the white-box implementations make a program much larger and slower. The AES implementation of Daemen and Rijmen requires 4352 bytes for lookup tables, thus the expected increase in size is about 177×. The performance slowdown is approximately 55× compared to a normal implementation of AES in [46]. None of the published papers have tried to implement white-box cryptography on Camellia so far, so it's hopeful to design new white-box cryptography on Camellia to secure against Cache timing attacks in the nearby future.

9 Conclusion

This paper makes some researches on access driven Cache timing attacks on the most secure block cipher Camellia, the results demonstrate that Camellia is facing serious threats from Cache timing attacks. First, Camellia has been widespread as the dominate block cipher in both Japan and European, so the effect of these attacks are wide and deeply; second, the adversary needn't to gain the encryption platform to physically measure the leaked side channel information, so it has strong applicability under remote environment such as local network, campus network, even the internet network; last but not the least, this attack is applicable to all software implementation for "Cache-Memory" structure of Computer equipment, thus can threaten the security of server, desktop, embedded Operating System. So, we should put strong concerns to this type of attack.

In this paper, we discover that the fast implementation of frequency S-box lookup operations in F function and left circular rotating operation of the key schedule have serious designing problem, it's very easy to be used by the adversary to implement the attack and recover full encryption key. Note that access driven Cache timing attacks on Camellia is decided by the fast implementation of frequency S-box lookup operations and the intrinsic mechanism of Cache, so it's very hard to defend these attacks. Different from traditionally countermeasures by modifying cipher hardware and OS implementation environments, we point that the cipher designer should burden more responsibilities due to the countermeasure cost and cipher applicability, provide several advices to the cipher designer about how to make such attacks more difficult. But all the countermeasures has to be at the cost of sacrifice the encryption speed, so how to make better balance between efficiency and speed are the big challenges for cryptographic systems.

Acknowledgements

We are indebted to ZHENG Tian-ming for his interests and insightful discussions about Camellia key analysis methods, Eran Tromer of the MIT for discussions on the reason why does this attack works and remote attack suggestions, and the (anonymous)

referees for their suggestions.

References:

[1] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. CRYPTO, volume 1109 of Lecture Notes in Computer Science, pages: 104–113, Springer, 1996.

[2] John Kelsey, Bruce Schneier, DavidWagner, and Chris Hall. Side channel cryptanalysis of product ciphers. Journal of Computer Security, volume 1485 of Lecture Notes in Computer Science, pages: 97-110, Springer, 1998.

[3] Dan Page. Theoretical use of cache memory as a cryptanalytic side-channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, pages: 1-23, June 2002.

[4] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. Cryptographic Hardware and Embedded Systems - CHES 2003, volume 1109 of Lecture Notes in Computer Science, pages: 62–76. Springer, 2003.

[5] Y. Tsunoo, T. Suzaki, T. Saito, T. Kawabata, and H. Miyauchi, "Timing Attack on Camellia Using Cache Delay in S-Boxes (in Japanese)," Proceedings of the 2003 Symposium on Cryptography and Information Security, SCIS2003, 3D-4, pages: 179--184, Jan. 2003.

[6] IKEDA YOSHITAKA, KANEKO TOSHINOBU. A study on the effect of cache structure to the cache timing attack for a block cipher(in Japanese). IEIC Technical Report, VOL.103, NO.714(WBS2003 174-190), pages:37-42,2004.

[7] Daniel J. Bernstein. Cache-timing attacks on AES, 2004. Available online at http://cr.yp.to/papers.html\#cachetiming.

[8] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and Countermeasures: the Case of AES. Topics in Cryptology – CT-RSA 2006, volume 3860 of Lecture Notes in Computer Science, pages: 1-20, Springer, 2006.

[9] Eran Tromer, Dag Arne Osvik, Adi Shamir, and. Efficient Cache Attacks on AES, and Countermeasures. Journal of Cryptology, Online Version of Lecture Notes in Computer Science, Springer, July 2009.

[10] M. Neve, J. Seifert, and Z. Wang. Cache time-behavior analysis on AES. http://www.cryptologie.be/document/Publications/AsiaCSS full 06.pdf, 2006.

[11] M. Neve, J. Seifert, and Z. Wang. A refined look at bernstein's AES side-channel analysis. In Proc. AsiaCSS 2006, page 369. ACM, 2006.

[12] M. O'Hanlon and A. Tonge. Investigation of cache-timing attacks on AES. http://www.computing.dcu.ie/research/papers/2005/0105.pdf, 2005.

[13] R. Salembier. Analysis of cache timing attacks against AES. Scholarly Paper, ECE Department, George Mason University, Virginia; available from: http://ece.gmu.edu/courses/ECE746/project/F06 Project resources/Salembier Cache Timing Attack.pdf, May 2006.

[14] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo. AES power attack based on induced cache miss and countermeasure. In International Symposium on Information Technology: Coding and Computing (ITCC 2005), volume 1, pages 586–591. IEEE Computer Society, 2005.

[15] C. Percival. Cache missing for fun and profit. Paper accompanying a talk at BSDCan 2005; available at http://www.daemonology.net/papers/htt.pdf, 2005.

[16] M. Neve and J. Seifert. Advances on access-driven cache attacks on AES. In E. Biham and A. Youssef, editors, Proc. SAC 2006, volume 4356 of LNCS, pages 147–162, Springer, 2006.

[17] J. Bonneau and I. Mironov. Cache-collision timing attacks against AES. In L. Goubin and M. Matsui, editors, Proc. CHES 2006, volume 4249 of LNCS, pages 201–215, Springer, 2006.

[18] O. Ac_içmez, W. Schindler, and Ç.K. Koç, Cache-Based Remote Timing Attack on the AES, Topics in Cryptology – CT-RSA 2007, volume 4377of Lecture Notes in Computer Science, pages 271–286,Springer, 2007.

[19] ZHAO Xinjie, WANG Tao, MI Dong. Robust First Two Rounds Access Driven Cache Timing Attack on AES, Volume 3 of International Conference on Computer Science and Software Engineering(CSSE 2008), pages: 785-788, 2008.

[20] E. Brickell, G. Graunke, M. Neve, and S. Seifert. Software mitigations to hedge AES against cache-based software side-channel vulnerabilities. http://eprint.iacr.org/2006/052.pdf, 2006.

[21] Johannes Blömer and Volker Krummel. Analysis of countermeasures against access driven cache attacks on AES. In C. Adams, A. Miri, and M. Wiener, editors, Proc. SAC 2007, volume 4876 of LNCS, pages 96–109, Springer, 2007.

[22] Z. Wang and R. Lee. New cache designs for thwarting software cache-based side channel attacks. In Proc. ISCA 2007, pages :494–505, ACM, June 2007.

[23] Yeping He and Sihan Qing, Square attack on reduced Camellia cipher, Proceedings of ICICS '01, Lecture Notes in Computer Science 2229, pages: 238-245, Springer- Verlag, 2001.

[24] Seonhee Lee, Seokhie Hong, Sangjin Lee, Jongin Lim, and Seonhee Yoon, Truncated differential cryptanalysis of Camellia, Proceedings of ICISC '01, Lecture Notes in Computer Science 2288, pages: 32-38, Springer-Verlag, 2002.

[25] Makoto Sugita, Kazukuni Kobara, and Hideki Imai, Security of reduced version of the block cipher Camellia against truncated and impossible differential cryptanalysis, Advances in Cryptology|ASIACRYPT'01, Lecture Notes in Computer Science 2248, pages: 193-207, Springer-Verlag, 2001.

[26] Taizo Shirai, Differential, linear, boomerang and rectangle cryptanalysis of reduced-Round Camellia, Proceedings of The Third NESSIE Workshop, 2002.

[27] Wenling Wu, Dengguo Feng, and Hua Chen, Collision attack and pseudorandom-ness of reduced-round Camellia, Proceedings of SAC '04, Lecture Notes in Computer Science 3357, pages: 256-270, Springer-Verlag, 2005.

[28] Duo Lei, Li Chao, and Keqin Feng, New observation on Camellia, Proceedings of SAC'05, Lecture Notes in Computer Science 3897, pages: 51-64, Springer-Verlag, 2006.

[29] Wenling Wu, Wentao Zhang, and Dengguo Feng, Impossible differential crypt-analysis of reduced-round ARIA and Camellia, Journal of Computer Science and Technology, Vol. 22(3), pages: 449-456, Springer, 2007. A preliminary version appears as the Cryptology ePrint Archive, Report 2006/350.

[30] Yongjin Yeom, Sangwoo Park, and Iljun Kim, On the security of Camellia against the square attack, Proceedings of FSE '02, Lecture Notes in Computer Science 2356, pages: 89-99, Springer-Verlag, 2002.

[31] Yongjin Yeom, Sangwoo Park, and Iljun Kim, A study of integral type crypt-analysis on Camellia, Proceedings of The 2003 Symposium on Cryptography and Information Security, pages: 453-456, 2003.

[32] Yasuo Hatano, Hiroki Sekine, and Toshinobu Kaneko, Higher order differential attack of Camellia(II), Proceedings of SAC '02, Lecture Notes in Computer Science 2595, pages:39-56, Springer-Verlag, 2003.

[33] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Improving the Efficiency of Impossible Differential Cryptanalysis of Reduced Camellia and MISTY1. Topics in Cryptology-CT-RSA 2008, pages: 370-386, Springer Berlin/Heidelberg, 2008.

[34] Duo Lei, Li Chao, and Keqin Feng, New observation on Camellia, Proceedings of SAC '05, Lecture Notes in Computer Science 3897, pp. 51{64, Springer-Verlag, 2006.

[35] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita, Camellia: a 128-bit block cipher suitable for multiple platforms | design and analysis, Proceedings of SAC '00, Lecture Notes in Computer Science 2012, pages: 39-56, Springer-Verlag, 2001.

[36] K. Aoki, T. Ichikawa, M. Kansa, M. Matsui, S. Moriai, Nakajima, and T. Tokita, “Specification of Camellia - a 128-bit Block Cipher”, http://www.cosic.esat.kuleuven.be/nessie/workshop/submissions, 2000.

[37] D. Eastlake, “Additional XML Security Uniform Resource Indentifiers (URIs)”, RFC4051, 2005.

[38] S. Moriai, A. Kato, M. Kanda, “Addition of Camellia Cipher Suites to Transport Layer Security (TLS)”, RFC4132, 2005.

[39] A. Kato, S. Moriai, M.Kanda, “The Camellia Cipher Algorithm and Its Use With IPsec”, RFC4312, 2005.

[40] OpenSSL the open-source toolkit for SSL / TLS [EB/OL], 2005. Available online at http://www.openssl.org/.

[41] Joan Daemen and Vincent Rijmen. The design of Rijndael: AES—the advanced encryption standard. Springer-Verlag, 2002.

[42] Office of state commercial cipher administration. Block cipher for WLAN products—SMS4[EB/OL]. http:// www.oscca.gov.cn/ UpFile/ 200622026423297990.pdf.

[43] D. Kwon, J. Kim, S. Park et al., New block cipher: ARIA, in: Proceedings of the Information Security and Cryptology-ICISC’03, LNCS, vol. 2971, 2003, pages. 432–445.

[44] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, A White-Box DES Implementation for DRM Applications, Proceedings of ACM CCS-9 Workshop DRM 2002 (J. Feigenbaum Ed.), LNCS vol. 2696, Springer, 2003, pages: 1-15.

[45] Hamilton E. Link and William D. Neumann, Clarifying Obfuscation: Improving the Security of White-Box Encoding, Cryptology ePrint Archive, Report 2004/025, http://eprint.iacr.org/2004/025, 2004.

[46] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, White-Box Cryptography and an AES Implementation, Proccedings of SAC'02 (K. Nyberg and H. M. Heys, Eds.), LNCS vol. 2595, Springer, 2003, pages: 250-270.

[47] Julien Bringer, Herve Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. Cryptology ePrint Archive, Report 2006/468, 2006. http://eprint.iacr.org/.