# Cache Timing Attacks on Camellia Block Cipher*

ZHAO Xin-Jie, WANG Tao

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

zhaoxinjieem@163.com, twangdrsjz@yahoo.com.cn

**Abstract**: Camellia, as the final winner of 128-bit block cipher in NESSIE, is the most secure block cipher of the world, none of the published paper has claimed to extract full Camellia key through experiment within limited time and data complexity. However, our research shows that, due to its frequent S-box lookup operations, Camellia is quite vulnerable to access driven Cache timing 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 three attacks on Camellia-128 and one attack on Camellia-192/256. Experiment results demonstrate: for about 500 random plaintexts, our basic attack on Camellia-128 can reduce the key searching space from $2^{128}$ to $2^{107.8}$, expand attack I can further reduce it to $2^{58.4}$ for Camellia with $FL/FL^{-1}$ and 1 for camellia without $FL/FL^{-1}$; besides basic attack, after analyzing just one simple left circular rotating operation of the key schedule, expand attack II can recover full 128-bit key for both Camellia with and without $FL/FL^{-1}$; for about 900 random plaintexts are enough to recover the full 192/256-bit key for Camellia-192/256 with and without $FL/FL^{-1}$. 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 attack.

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

## 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 2003. The first actual cache attacks on DES were implemented by Tsunoo et al.[4]. The main focus of those papers was on DES. In 2005, Bernstein[5] and Osvik, Shamir, and Tromer[6] 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[7][8][9][10][11], improving the attack[12][13][14][15], and devising and analyzing countermeasures[16][17][18].

However, 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 analyzed the Camellia with Cache timing attacks. Camellia was thought to be the most secure block cipher of the world, none of the published paper has claimed to recover full encryption key through experiment within limited time and data complexity. In this paper, we analyzed Cache timing attacks on Camellia, our research shows that it's possible to recover full encryption key of Camellia-128/192/256 within less than $2^{10}$ samples and

* 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.

1 second analysis, even Camellia in OPENSSL-1.0.0-beta2 with *FL*/*FL*-1 function, which is the latest version published by May 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, 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 get partial candidates of the data block before and after executing <<< operation $W^1$, $W^2$, the candidates count for $W^1{}_i$ and $W^2{}_i$ (one byte) can be further reduced through F function analysis. As both $W^1{}_i$ and $W^2{}_i$ have close relations with $K$, usually, $W^1{}_i$ and $W^2{}_i$ are related with different encryption key byte $K_i$; also, $W^1{}_i$ and $W^2{}_j$ has some other relations between themselves because of the <<< n operation. We find out that: combing certain analysis methods, the adversary can eliminate wrong candidates and recover correct key very efficiently. Experiments demonstrate that just analyzing one <<< operation can narrow down the key searching space from$2^{107.8}$ to 1.

4) Propose and realize several Cache timing attacks on Camellia.

After analyzing on the F function and <<< operation during Camellia implementation, we have successfully designed and realized 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 $2^{10}$ samples and less than 1 seconds analysis; besides, improving key length and design complexity of Camellia can not enhance its security against Cache timing attacks, as to attack on Camellia-128, the adversary need to measure, analyze both the encrypt procedure and key schedule to recover full key, however, in the case of Camellia-192/256, after the encrypt procedure measurement and analysis, the adversary even needn't to measure its key schedule, just a simple derivation of the Camellia key schedule is enough to deduce full key directly. 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 | [19] | Square attack | 6 | x | $2^{111.7}$ | $2^{112}$ | 128-bit |
| Camellia-128 | [20] | Truncated differential | 8 | x | $2^{83.6}$ | $2^{55.6}$ | 128-bit |
| Camellia-128 | [21] | Impossible differential | 7 | x | --- | --- | 128-bit |
| Camellia-128 | [29] | Impossible differential | 11 | x | $2^{120}$ | $2^{83.4}$ | 128-bit |
| Camellia-128 | Section 4 | Cache Timing attack | 2 | √ | $2^{8.97}$ | 1s | 20.2-bit |
| Camellia-128 | Section 5 | Cache Timing attack | 18 | x | $2^{8.97}$ | 1s | 128-bit |
| Camellia-128 | Section 5 | Cache Timing attack | 6 | √ | $2^{8.97}$ | 1s | 69.6-bit |
| Camellia-128 | Section 5 | Cache Timing attack | 6 | x / √ | $2^{8.97}$ | 1s | 128-bit |
| Camellia-192/256 | [29] | Boomerang attack | 9 | √ | $2^{124}$ | $2^{170}$ | 192/256-bit |
| Camellia-192/256 | [23] | Collision attack | 9 | x | $2^{13}$ | $2^{175.6}$ | 192/256-bit |
| Camellia-192/256 | [24] | Square attack | 10 | x | --- | $2^{186}$ | 192/256-bit |
| Camellia-192/256 | [25] | Impossible differential | 12 | x | $2^{120}$ | $2^{181}$ | 192/256-bit |
| Camellia-192/256 | [29] | Impossible differential | 12 | x | $2^{119}$ | $2^{147.3}$ | 192/256-bit |
| Camellia-192/256 | Section 6 | Cache Timing attack | 6 | x / √ | $2^{9.81}$ | 1s | 192/256-bit |
| Camellia-256 | [26] | Square attack | 9 | √ | $2^{60}$ | $2^{202}$ | 256-bit |
| Camellia-256 | [27] | Integral cryptanalysis | 9 | √ | $2^{60.5}$ | $2^{202.5}$ | 256-bit |
| Camellia-256 | [22] | Rectangle attack | 10 | √ | $2^{127}$ | $2^{241}$ | 256-bit |
| Camellia-256 | [23] | Collision attack | 10 | x | $2^{14}$ | $2^{239.9}$ | 256-bit |
| Camellia-256 | [22] | Differential | 11 | x | $2^{104}$ | $2^{232}$ | 256-bit |
| Camellia-256 | [28] | High-order differential | 11 | x | $2^{21}$ | $2^{255}$ | 256-bit |
| Camellia-256 | [28] | High-order differential | 11 | √ | $2^{93}$ | $2^{256}$ | 256-bit |
| Camellia-256 | [30] | Square attack | 11 | x | --- | $2^{250}$ | 256-bit |
| Camellia-256 | [22] | Linear cryptanalysis | 12 | x | $2^{119}$ | $2^{247}$ | 256-bit |
| Camellia-256 | [29] | Impossible differential | 13 | x | $2^{120}$ | $2^{211.7}$ | 256-bit |

5) Discuss why and how these attacks work on Camellia, and provide 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 7.

### 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 Cache timing attack model and how it can be used into Camellia F function attack. Section 4 gives the basic attack on Camellia-128, Section 5 presents several expanded attacks on Camellia-128, and Section 6 displays attacks on Camellia-192/256. Section 7 discusses on the reason why Camellia is vulnerable to this type of attack and provides several advices to the cipher designer. Section 8 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-beta2.

$X_L$: 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

$\delta$: 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 $\delta$ 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 $[a/B] \bmod S$.

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, so it's much more efficiently 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.

Summary: 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[31]. It was chosen as a recommended algorithm by the NESSIE (New European Schemes for Signatures, Integrity and Encryption) project in 2003 [32] and was certified as the IETF (Internet Engineering Task Force) standard cipher for XML security URIs, SSL/TLS cipher suites and IPsec in 2005[33][34][35]. In March 2009, Camellia was integrated into the OPENSSL-1.0.0-beta1, which is the most widespread cryptographic library of the world. A full description of the Camellia cipher is provided in [31][32], 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.

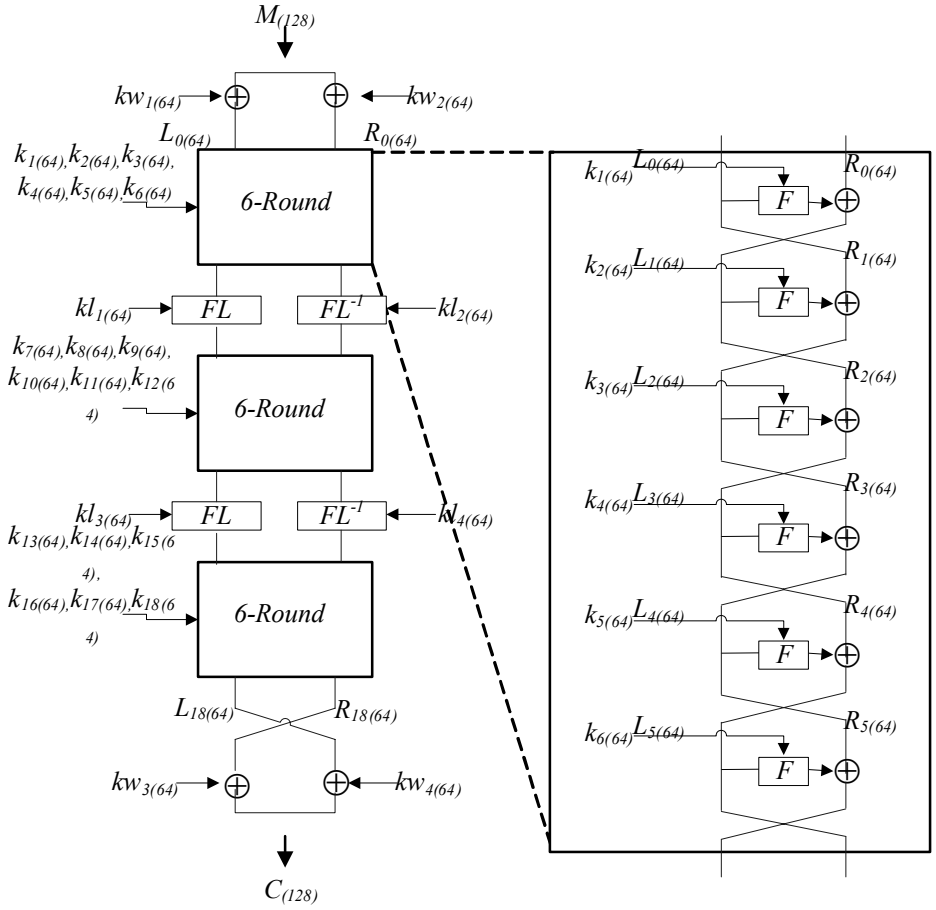


Fig.1 Encryption process of Camellia

**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 subkeys $k_{wt}$, $k_u$, and $k_{lv}$ are generated from $K_L$, $K_R$, $K_A$, and $K_B$. The subkeys 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.

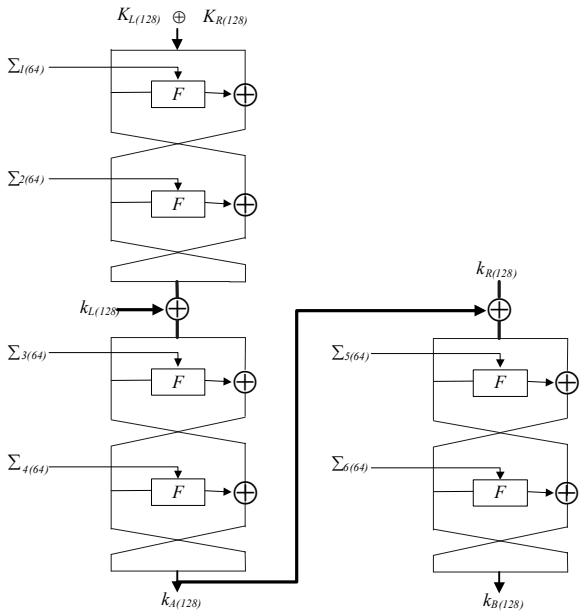


Fig.2 key schedule of Camellia

Table 2. Subkeys 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. Subkeys 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 | $kw_4(KE_{66}, KE_{67})$ | $(K_B<<<111)_R$ |

From Fig.1 and Fig.2 we can clearly see that both Camellia encryption procedure and key schedule uses the F function, every F function has 2 times table lookup for each 4 S-boxes to access Cache, so the adversary is possible to deduce some information about $K$ from both Camellia encryption procedure and key schedules, this is very important for the coming analysis in this paper.

## 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:

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

$\alpha$ : Part of the plaintext (first round analysis) or cipher (last round analysis), even the known inner state.

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

$\gamma$ : Parameter related with the S-box lookup indices or results.

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

From Section 2.2, we know that the adversary can get $\gamma$ through the measuring phase of Cache timing attack, as $\alpha$ is usually known, then, it's not difficult to compute $\beta$ , 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 $\gamma$ , $\alpha$ sometimes is known, thus he can get the impossible value of $\beta$ , combing analyzing the relations between $\beta$ 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 $\alpha$ , but generates different values of plaintext *P* or cipher *C*, so the Cache access profile are different, as we know that $\beta$ is fixed, if the adversary can get the accessed Cache traces, then predict some candidates of $\gamma$ , from formula (2), he can directly get a possible set of $\beta$ candidates, the correct candidate $\beta$ is always belong to this set, then fixes another value for $\alpha$ , using the same methods above to improve the predict frequency of correct $\beta$ , after several times of analyzing, the one with the highest frequency is always the correct value for $\beta$ . Combing analyzing the relations between $\beta$ and the key *K*, correct key *K* can also be predicted.

As every un-accessed Cache set is related with $\delta$ (usually $\delta$=16) impossible S-box lookup indices and results, using analysis strategy 1 the adversary can eliminate $\delta$ impossible $\beta$ candidates, so it's rather 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, 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-beta2.

```
#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-beta2

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; $\alpha$ : part of the plaintext or some known inner state; $\beta_1$ : 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:

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

$\varphi$ denotes a function to return the *n-st* byte of a 32-bit value; $\gamma$ denotes the index of $S_n$ entry.

The adversary can gather $\gamma$ through Cache timing attack measuring phase, $\alpha$ and $n$ is also known, so the candidates for $\beta_1 \oplus \beta_2$ can be predicted. If the encrypt plaintext changed, $\gamma$ and $\alpha$ 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 Basic Camellia-128 Attack

### 4.1 Attack Encryption Procedure

#### 4.1.1 First Round

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, $\beta_1$ in formula (3) and (4) can be expressed as $KE_0$, $KE_1$ in first round Camellia F function call. Then, we have 8 equations as follows:

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

$\gamma_n$ denotes the index of $S_n$ entry, its candidates can be gathered from measuring phase, suppose we get the impossible value of $\gamma_n$ , 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 $\gamma$ , so obviously, we can also get every S-box lookup result. From the F function code of Fig3, 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:

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

Summary: in this section, after analyzing on the F function call in the first round Camellia encryption, we can get the correct value for $KE_0 \oplus KE_4$, $KE_1 \oplus KE_5$ and compute the output expression of $s^1_2$, $s^1_3$, $KE_0$ denotes $K_0$,$K_1$,$K_2$,$K_3$ 4 bytes, and $KE_1$ denotes $K_4$,$K_5$,$K_6$,$K_7$ 4 bytes, and it's clear to see that we can't recover the $KE_0$ and $KE_1$ directly.

4.1.2 Second Round Attack

From Section 4.1.1 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:

$$\begin{aligned} \gamma_1 &= P_8 \oplus KE_{2,0} \oplus KE_{6,0} \quad & \gamma_1 &= P_{15} \oplus KE_{3,3} \oplus KE_{7,3} \\ \gamma_2 &= P_9 \oplus KE_{2,1} \oplus KE_{6,1} \quad & \gamma_2 &= P_{12} \oplus KE_{3,0} \oplus KE_{7,0} \\ \gamma_3 &= P_{10} \oplus KE_{2,2} \oplus KE_{6,2} \quad & \gamma_3 &= P_{13} \oplus KE_{3,1} \oplus KE_{7,1} \\ \gamma_4 &= P_{11} \oplus KE_{2,3} \oplus KE_{6,3} \quad & \gamma_4 &= P_{14} \oplus KE_{3,2} \oplus KE_{7,2} \end{aligned} \tag{5}$$

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 *c*2.

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

Summary: in this section, after analyzing on the F function call in the second round Camellia encryption, we can get the correct value for $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$ and compute the output expression of $s^2_2$, $s^3_3$. $KE_2$ denotes $K_8$,$K_9$,$K_{10}$,$K_{11}$ 4 bytes, and $KE_3$ denotes $K_{12}$,$K_{13}$,$K_{14}$,$K_{15}$ 4 bytes, and it's clear to see that we can't recover the $KE_2$ and $KE_3$ directly. In order to recover part or full key K, we have to attack the Camellia key schedule.

**4.2** Key Schedule Attack

Fig.4 is partial code of Camellia-128 key schedule in OPENSSL-1.0.0-beta2. It's clear to see that Camellia-128 calls F function 4 Times (code line 4,5,7,8).

According to formula (4), we can see that the S-box lookup index is composed of *s* (s0, s1, or s2, s3) XORed with the constant $\sum_1$.

Take code line 4 as an example, we can get:

$$\begin{aligned} \gamma_1 &= K_0 \oplus \Sigma_{1,0} \quad & \gamma_1 &= K_7 \oplus \Sigma_{1,7} \\ \gamma_2 &= K_1 \oplus \Sigma_{1,1} \quad & \gamma_2 &= K_4 \oplus \Sigma_{1,4} \\ \gamma_3 &= K_2 \oplus \Sigma_{1,2} \quad & \gamma_3 &= K_5 \oplus \Sigma_{1,5} \\ \gamma_4 &= K_3 \oplus \Sigma_{1,3} \quad & \gamma_4 &= K_6 \oplus \Sigma_{1,6} \end{aligned} \tag{7}$$

As the Cache trace profile of Camellia key schedule under the same encryption key are all the same. So the adversary can only get limited un-accessed Cache sets (one time measurement) during the Camellia key Schedule. During the Camellia-128 key schedule, as it calls F function 4 times, so in all, the whole key schedule has 32 times S-box lookups (8 times for each S-box). Obviously, the key schedule can accessed 8 different Cache sets for each S-box, which also means that, theoretically speaking, the adversary can

measure about 8 un-accessed Cache sets for each S-box. As to $\delta$=16, each un-accessed Cache set is related with 16 impossible S-box lookup indices, which also means that it's related with 16 impossible $K_i$ (one byte of $K$), so generally speaking, from formula(7) we can eliminate 128 impossible values for $K_i$ ($i$=0,1,2,3,4,5,6,7).

Also, take code line 8 as an example, the S-box lookup index is composed of s (s2, s3) XORed with the constant $\sum_4$. Note that s2, s3 are equal to $KE_6$, $KE_7$. we can finally get the candidates for $KE_6$, $KE_7$.

```
int Camellia_Ekeygen(int keyBitLength, const u8 *rawKey, KEY_TABLE_TYPE k)
{
1.  register u32 s0,s1,s2,s3;
2.  k[0] = s0 = GETU32(rawKey);    k[1] = s1 = GETU32(rawKey+4);
3.  k[2] = s2 = GETU32(rawKey+8); k[3] = s3 = GETU32(rawKey+12);
4.  Camellia_Feistel(s0,s1,s2,s3,SIGMA+0);
5.  Camellia_Feistel(s2,s3,s0,s1,SIGMA+2);
6.  s0 ^= k[0], s1 ^= k[1], s2 ^= k[2], s3 ^= k[3];
7.  Camellia_Feistel(s0,s1,s2,s3,SIGMA+4);
8.  Camellia_Feistel(s2,s3,s0,s1,SIGMA+6);
9.  k[ 4] = s0, k[ 5] = s1, k[ 6] = s2, k[ 7] = s3;
10. RotLeft128(s0,s1,s2,s3,15);/* KA <<< 15 */
11. k[12] = s0, k[13] = s1, k[14] = s2, k[15] = s3;
}
```

Fig.4 Partial code of Camellia-128 key schedule

Summary: from Section 4.1 we can get the correct value about $KE_0 \oplus KE_4$, $KE_1 \oplus KE_5$, $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$, and from Section 4.2 we can get partial candidates for $K_i$ ($i$=0,1,2,3,4,5,6,7) and $KE_6$, $KE_7$. So, if we combine the attack result together, we input every candidate of $KE_6$, $KE_7$ into $KE_2 \oplus KE_6$ and $KE_3 \oplus KE_7$, we can get the possible candidates for $KE_2$, $KE_3$, which also is candidates for $K_i$ ($i$=8,9,10,11,12,13,14,15). Theoretically speaking, the candidates number for $K_i$ is about 128, which also means that we can reduce the key searching space from $2^{128}$ to $2^{112}$.

**4.3** Experiment

We have implemented our Camellia access driven cache timing attack against OPENSSL-1.0.0-beta2 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.5 displays the relationship between $KE_0 \oplus KE_4$ key byte searching space and sample size. It's obviously to see that more than 500 samples are enough to recover every key byte of $KE_0 \oplus KE_4$.

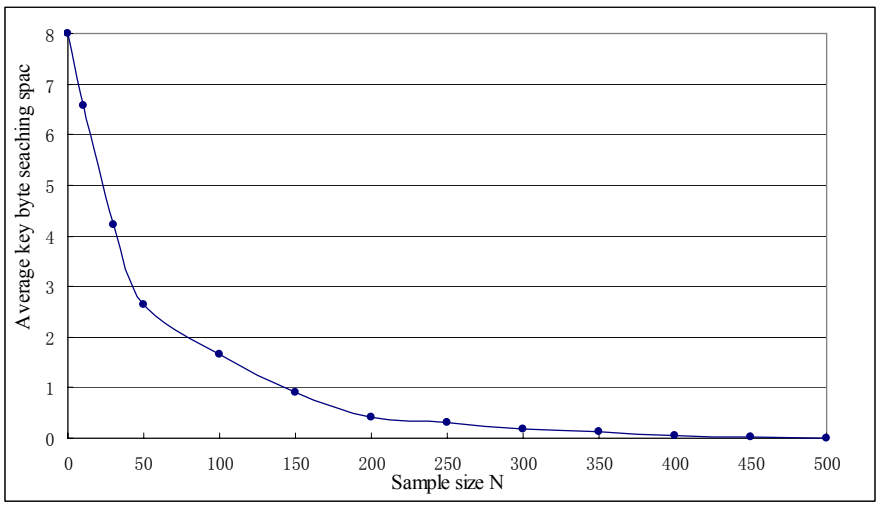


Fig.5 Average $KE_0 \oplus KE_4$ key byte searching space and Attack sample size

Fig.6 shows the key searching space decreasing after basic attack, it's obviously to see that after the basic attack, the average key searching space of Camellia-128 is reduced from $2^{128}$ to $2^{107.8}$.

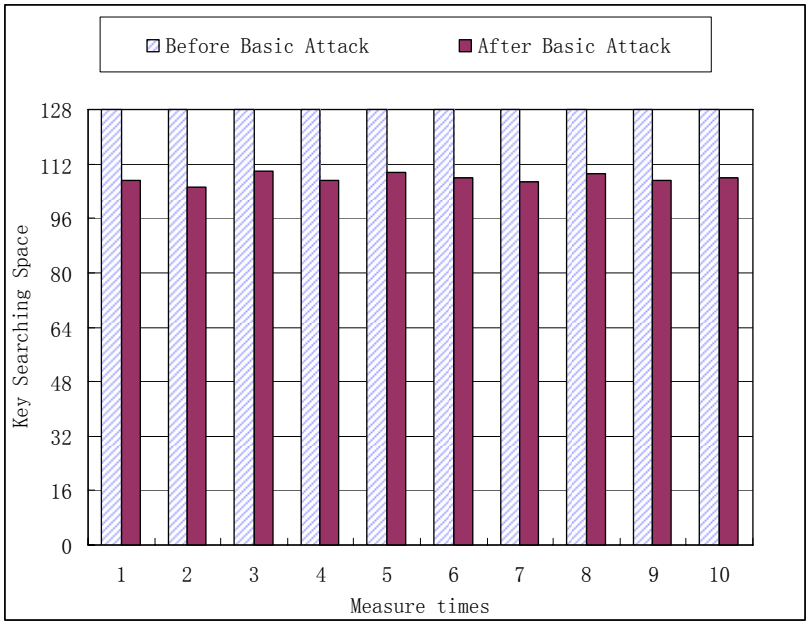


Fig.6 Key searching spaces of before and after the basic attack

## 5 Expand Camellia-128 Attack

### 5.1 Expand Encryption Procedure 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 can not get $K$ directly from attack 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 deduces from the *r-st* round encryption procedure 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； after analyzing on whole 18 rounds F function calls(Camellia without $FL/FL^{-1}$), what we can get is shown in Table 5.

Table 4. Attack results of first 6 rounds Camellia with $FL/FL^{-1}$

| Round | Result | Round | Result |
|---|---|---|---|
| 1 | $KE_0 \oplus KE_4$, $KE1 \oplus KE_5$ | 4 | $KE_2 \oplus KE_{10}$, $KE_3 \oplus KE_{11}$ |
| 2 | $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$ | 5 | $KE_0 \oplus KE_{12}$, $KE_1 \oplus KE_{13}$ |
| 3 | $KE_0 \oplus KE_8$, $KE1 \oplus KE_9$ | 6 | $KE_2 \oplus KE_{14}$, $KE_3 \oplus KE_{15}$ |

Table 5. Attack results of 18 rounds Camellia without $FL/FL^{-1}$

| Round | Result | Round | Result |
|---|---|---|---|
| 1 | $KE_0 \oplus KE_4$, $KE1 \oplus KE_5$ | 11 | $KE_0 \oplus KE_{28}$, $KE_1 \oplus KE_{29}$ |
| 2 | $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$ | 12 | $KE_2 \oplus KE_{30}$, $KE_3 \oplus KE_{31}$ |
| 3 | $KE_0 \oplus KE_8$, $KE1 \oplus KE_9$ | 13 | $KE_0 \oplus KE_{36}$, $KE1 \oplus KE_{37}$ |
| 4 | $KE_2 \oplus KE_{10}$, $KE_3 \oplus KE_{11}$ | 14 | $KE_2 \oplus KE_{38}$, $KE_3 \oplus KE_{39}$ |
| 5 | $KE_0 \oplus KE_{12}$, $KE1 \oplus KE_{13}$ | 15 | $KE_0 \oplus KE_{40}$, $KE1 \oplus KE_{41}$ |
| 6 | $KE_2 \oplus KE_{14}$, $KE_3 \oplus KE_{15}$ | 16 | $KE_2 \oplus KE_{42}$, $KE_3 \oplus KE_{43}$ |
| 7 | $KE_0 \oplus KE_{20}$, $KE1 \oplus KE_{21}$ | 17 | $KE_0 \oplus KE_{44}$, $KE1 \oplus KE_{45}$ |
| 8 | $KE_2 \oplus KE_{22}$, $KE_3 \oplus KE_{23}$ | 18 | $KE_2 \oplus KE_{46}$, $KE_3 \oplus KE_{47}$ |
| 9 | $KE_0 \oplus KE_{24}$, $KE1 \oplus KE_{25}$ | Postwhitening | $KE_0 \oplus KE_{48}$, $KE1 \oplus KE_{49}$ |
| 10 | $KE_2 \oplus KE_{26}$, $KE_3 \oplus KE_{27}$ | Postwhitening | $KE_2 \oplus KE_{50}$, $KE_3 \oplus KE_{51}$ |

Note that as to Camellia without ($FL/FL^{-1}$), if cipher $C$ is known, also we can get the expression of final round F function output ($S^r_0$, $S^r_1$, $S^r_2$, $S^r_3$), as the output results XORed with($KE_{48}$,$KE_{49}$,$KE_{50}$,$KE_{51}$) is just the cipher $C$, so we can get the correct value for $KE_0 \oplus KE_{48}$, $KE1 \oplus KE_{49}$, $KE_2 \oplus KE_{50}$, $KE_3 \oplus KE_{51}$.

**5.2** Expand Key Schedule Attack I

From Section 5.1, we can get correct results for the encryption key $K$ XORed with the expand-key $KE$, but we can not get $K$ directly. So in order to further reduce the key searching space for K, we have to analyze the key schedule of Camellia.

Besides F function, Camellia key schedule used a lot of <<< operations to generate the expand-key. The <<< operations is quite vulnerable to Cache timing attack. Take Table 2 and code line 8,9,10 in Fig.4 as an example. As we known, from Table 4 and Table 5 we can get the correct value for $KE_0 \oplus KE_4$, $KE1 \oplus KE_5$, $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$ and $KE_0 \oplus KE_{12}$, $KE_1 \oplus KE_{13}$, $KE_2 \oplus KE_{14}$, $KE_3 \oplus KE_{15}$. As limited candidates for $KE_0$, $KE_1$, $KE_2$, $KE_3$ is known after Camellia basic attack in Section 4, so it's very easy for us to reduce the possible candidates for $KE_4$, $KE_5$, $KE_6$, $KE_7$ and $KE_{12}$, $KE_{13}$, $KE_{14}$, $KE_{15}$. According to code line 9 in Fig.4, we can get:

$$(KE_{12}, KE_{13}, KE_{14}, KE_{15}) = \text{RotLeft128}\ (KE_4, KE_5, KE_6, KE_7, 15) \tag{8}$$

Specifically speaking, we have 16 equations:

$$\begin{aligned}
&KE_{12,0} = KE_{4,1[7]} \,||\, KE_{4,2[0,\ldots6]} \quad KE_{12,1} = KE_{4,2[7]} \,||\, KE_{4,3[0,\ldots6]} \\
&KE_{12,2} = KE_{4,3[7]} \,||\, KE_{5,0[0,\ldots6]} \quad KE_{12,3} = KE_{5,0[7]} \,||\, KE_{5,1[0,\ldots6]} \\
&KE_{13,0} = KE_{5,1[7]} \,||\, KE_{5,2[0,\ldots6]} \quad KE_{13,1} = KE_{5,2[7]} \,||\, KE_{5,3[0,\ldots6]} \\
&KE_{13,2} = KE_{5,3[7]} \,||\, KE_{6,0[0,\ldots6]} \quad KE_{13,3} = KE_{6,0[7]} \,||\, KE_{6,1[0,\ldots6]} \\
&KE_{14,0} = KE_{6,1[7]} \,||\, KE_{6,2[0,\ldots6]} \quad KE_{14,1} = KE_{6,2[7]} \,||\, KE_{6,3[0,\ldots6]} \\
&KE_{14,2} = KE_{6,3[7]} \,||\, KE_{7,0[0,\ldots6]} \quad KE_{14,3} = KE_{7,0[7]} \,||\, KE_{7,1[0,\ldots6]} \\
&KE_{15,0} = KE_{7,1[7]} \,||\, KE_{7,2[0,\ldots6]} \quad KE_{15,1} = KE_{7,2[7]} \,||\, KE_{7,3[0,\ldots6]} \\
&KE_{15,2} = KE_{7,3[7]} \,||\, KE_{4,0[0,\ldots6]} \quad KE_{15,3} = KE_{4,0[7]} \,||\, KE_{4,1[0,\ldots6]}
\end{aligned} \tag{9}$$

Take the first equation as an example, from it we can know that the correct value of $KE_{12,0}$ and $KE_{4,2}$ have this feature: the right 7-bit value of $KE_{12,0}$ should equal to left 7-bit value of $KE_{4,2}$.

$$KE_{12,0[1,\ldots7]} = KE_{4,1[7]} \,||\, KE_{4,2[0,\ldots6]} \tag{10}$$

If we input every candidate of $KE_{12,0}$ and $KE_{4,2}$ into formula (10), the candidates do not fulfill the equation above can be eliminated from $KE_{12,0}$ and $KE_{4,2}$ candidate sets. Take the left part of formula (10) as an example, as every candidate for $KE_{12,0}$ is related to a candidate for $K_0$, so the elimination of $KE_{12,0}$ candidates also means that the candidate count for $K_0$ is decreasing, we can reduce the candidates count for $K_0$. From formula (9), we know that there are 15 other equations like formula(10), we can reduce the candidates count for $K_i$. Also, Take the right part of formula (10) as an example, every candidate for $KE_{4,2}$ is related to a candidate for $K_2$, so the elimination of $KE_{4,2}$ candidates also means that the candidate count for $K_2$ is decreasing, using the same methods on 15 other equations like formula(10), we can further reduce the candidates count for $K_i$. So after every <<< function analysis, $K_i$ is reduced 2 times.

Considering Camellia with $FL/FL^{-1}$ function, it's clear to see that besides $KE_4$, $KE_5$, $KE_6$, $KE_7$ , $KE_{12}$, $KE_{13}$, $KE_{14}$, $KE_{15}$ related to <<< function about $K_A$, we can also analyze $KE_0$, $KE_1$, $KE_2$ , $KE_3$, $KE_8$, $KE_9$, $KE_{10}$, $KE_{11}$ related to <<< function about $K_L$, thus we can further reduce the candidates count for $K_i$.

Considering Camellia without $FL/FL^{-1}$ function, it's clear to see that besides $KE_4$, $KE_5$, $KE_6$, $KE_7$ , $KE_{12}$, $KE_{13}$, $KE_{14}$, $KE_{15}$ related to <<< function about $K_A$, we can use many other <<< operations to eliminate wrong candidates of $K_i$, specifically, 6 <<< operations about KL, 6 <<< operations about KA, so we can have (6*5)/2+(6*5)/2=30 rounds analysis points, and finally recover the correct $K_i$.

Summary: after 2 round analysis on <<< operations of Camellia with $FL/FL^{-1}$ function, we can further reduce the candidates count for $K_i$, but it's still quite hard to get the only correct value for $K_i$; after 30 rounds analysis on <<< operations of Camellia without $FL/FL^{-1}$ function, it's possible for us to recover encryption key $K$.

**5.3** Expand Key Schedule Attack II

As we know, the key schedule attack in 5.2 can not recover the whole encryption key $K$ for Camellia with $FL/FL^{-1}$ function, the reason for it is that for every byte $K_i$, we just eliminate wrong candidates of it for about 4 times (2 rounds analysis), but do we have other ways to eliminate candidates of $K_i$ for more times?

Let's also take code line 8,9,10 in Fig.4 as an example, As the rotate num $n$=15, which means that the output value after rotating is almost 2 bytes left rotate from the input value. According to formula (8), we can transfer every equation into a set of variables related

with the eliminating $K_i$.

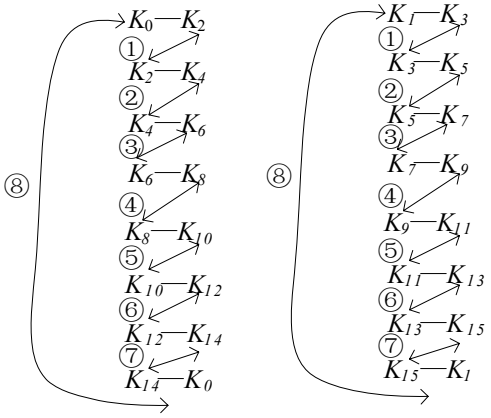


Fig.7 Elimination of $K_i$ in formula (8)

Suppose $NK_i$ denotes the number of candidates for $K_i$, from Fig.7 it's easy to find out:

1 After every time of elimination, $NK_i$ and $NK_{(i+2)\%16}$ are decreasing, $N_i = NK_{(i+2)\%16}$.

2 The elimination have a Domino chain effect.

Take left part of Fig.7 as an example:

(1) After elimination 1, $NK_0$ and $NK_2$ are decreasing, $NK_0 = NK_2$.

(2) After the elimination 2, $NK_2$ and $NK_4$ are decreasing, $NK_2 = NK_4$. Back to (1), $NK_0$ is decreasing, but $NK_0 = NK_2 = NK_4$.

(3) After the elimination 3, $NK_4$ and $NK_6$ are decreasing, $NK_4 = NK_6$. Back to (2), $NK_4$ and $NK_2$ is decreasing, back to (1), $NK_0$ is decreasing. $NK_0 = NK_2 = NK_4 = NK_6$.

……

(8) After the elimination 8, $NK_{14}$ and $NK_0$ are decreasing, $NK_{14} = NK_0$. Back to (1)(2)(3)(4)(5)(6)(7), we have $NK_0 = NK_2 = NK_4 = NK_6 = NK_8 = NK_{10} = NK_{12} = NK_{14}$.

Take right part of Fig.7 as an example, after 8 times elimination, we have $NK_1 = NK_3 = NK_5 = NK_7 = NK_9 = NK_{11} = NK_{13} = NK_{15}$.

So, every time of elimination, it can always decrease other $NK_j$. After about 8 times of decreasing, we can reduce the key searching space dramatically. Experiment demonstrates that this was enough to reduce the Camellia-128 searching space from $2^{112}$ to 1! Even if this can not recover full Camellia-128 key, we can also analyze $KE_0, KE_1, KE_2, KE_3, KE_8, KE_9, KE_{10}, KE_{11}$ related to <<< function about $K_L$ to have another 8 times of decreasing like this Section, finally recover the full 128-bit Camellia key.

Note that only analysis on first 6 rounds is enough to recover the full 128-bit Camellia key, which means that both Camellia with and without $FL/FL^{-1}$ are vulnerable to the attack of this Section.

### 5.4 Experiment

Fig.8 shows the 128-bit key searching space of basic attack and the expand attack I for Camellia with $FL/FL^{-1}$, it's clear to see that the average key searching space after basic attack is about $2^{107.8}$, after the expand attack I, it is reduced to about $2^{58.4}$.

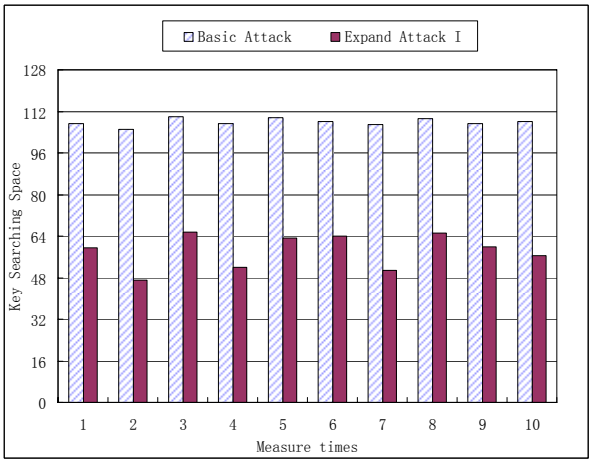


Fig.8 Key searching spaces of basic attack and expand attack I

As to Camellia without $FL/FL^{-1}$, the first expand key schedule attack is enough to recover full 128-bit key. And our experiment results demonstrate that, after the second expand key schedule attack, the key searching space is down to 1, just attacking one <<< operation of the key schedule is enough to reduce the key searching space from $2^{107.8}$ to 1. So, the <<< operation of the Camellia-128 key schedule is rather vulnerable to Cache timing attack!

## 6 Camellia-192/256 Attack

### 6.1 Key Schedule 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-129?

The answer is No! In Camellia-128 attack, we need to measure and analyze both encryption procedure and key schedule to recover the full 128-bit key, in Camellia-192/256 attack, we just need to measure and analyze the encryption procedure , combing a simple analysis on its key schedule, it's very easy to recover 192/256-bit key.

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, this is the most secure algorithm currently, and we just analyze the first 6 rounds of Camellia encryption and the key schedule part. As the first 6 rounds of Camellia encryption and the key schedule part are all the same for algorithm with and without $FL/FL^{-1}$, so the attack is also applicable to Camellia-192/256 without $FL/FL^{-1}$.

From the key schedule of Camellia-192/256(Fig.2, Table 3), we can see that:

$$(KE_8,\ KE_9,\ KE_{10},\ KE_{11}) = K_R <<< 15 \quad (KE_{12},\ KE_{13},\ KE_{14},\ KE_{15}) = K_A <<< 15 \tag{11}$$

From Table 4, we can get the value of $KE_0 \oplus KE_8$, $KE1 \oplus KE_9$, $KE_2 \oplus KE_{10}$, $KE_3 \oplus KE_{11}$ and $KE_0 \oplus KE_{12}$, $KE_1 \oplus KE_{13}$, $KE_2 \oplus KE_{14}$, $KE_3 \oplus KE_{15}$, which also means we can get the value of $KE_8 \oplus KE_{12}$, $KE_9 \oplus KE_{13}$, $KE_{10} \oplus KE_{14}$, $KE_{11} \oplus KE_{15}$, then, we can get the value of $(K_A <<< 15) \oplus (K_R <<< 15)$, so we can get the value of $K_A \oplus K_R$, 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 4, we can get the value of $KE_0 \oplus KE_4$, $KE1 \oplus KE_5$, $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$, so obviously, we can get the correct value of $KE_0$, $KE1$, $KE_2$, $KE_3$, 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 $KE_8$, $KE_9$, $KE_{10}$, $KE_{11}$, also, we can recover the value $K_R$.

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 know, 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.

### 6.2 Experiment

Fig.9 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 500 of Camellia-128.

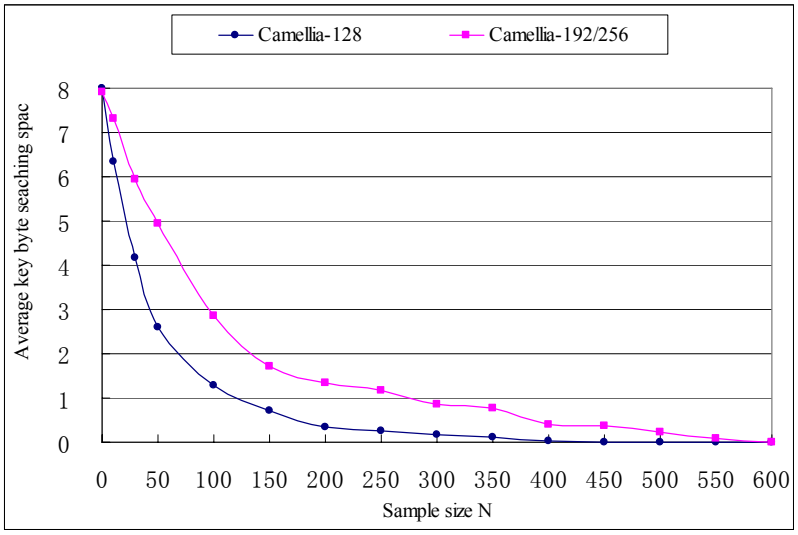


Fig.9 Average $KE_0 \oplus KE_4$ key byte searching space and Attack sample size for Camellia 128 and 192/256

After we get the first 6 rounds attack results in Table 4 or Table 5, we can analyze the encryption key with the method of Section 6.1, then recover the encryption key. Our experiments demonstrate that for about 800-900 samples are enough to recover the full Camellia-192/256 key.

## 7 Discussions and Countermeasures

### 7.1 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.

Every F function used 8 times table lookup operations, 2 times for each S-box. The adversary can get the accessed Cache sets during the F function call, these Cache addresses can be transferred into S-box lookup indices, which is also related with a known variable (plaintext, inner state or cipher) and the secret key. According to the analysis model of Section 3, it's not difficult to recover the secret key. But the secret key doesn't equal to the encryption key $K$, but the results of encryption key XORed with expand-key, so in order to recover the value of the encryption key, we have to analyze the key schedule of Camellia.

Camellia-128 key schedule calls the F function for 4 times, after measuring of the key schedule Cache access profiles, through the first call analysis, we can directly get the possible candidates for $K_i$ ($i$=0,1,…7), through the 4-th call analysis, we can get the possible candidates for $KE_6$ and $KE_7$, as the value of $KE_2 \oplus KE_6$ and $KE_3 \oplus KE_7$ is known to us after attacking the encryption procedure, so we can get the possible candidates for $K_i$ ($i$=7,8,…15), as the key schedule is fixed, we can only get very limited Cache access profile with one time measurement, but up to now, we still can't recover full 128-bit key. In order to further reduce the key searching space, we have to analysis other parts of the key schedule, that is <<< n left circular rotating operation. Through former analysis, we can get the 128-bit candidates before and after the <<< n operation, due to their linear transformations and relations with the encryption key, all of the key candidates don't fit for the <<< line relations are eliminated, as the <<< operation analysis has the domino chain effect (Section 5.3), the encryption key can be recovered very efficiently, just one <<< operation analysis is enough to narrow down the key searching space from $2^{107.8}$ down to 1.

Camellia-192/256 calls the F function 6 times, during the first F call, we can't get any information about $K_i$ ($i$=0,1,…7), but candidates about $KE_0 \oplus (KE_8 >>> 15)$, $KE_1 \oplus (KE_9 >>> 15)$, $KE_2 \oplus (KE_{10} >>> 15)$, $KE_3 \oplus (KE_{11} >>> 15)$, which is nonsense information to recover K, so as the 2-th, 3-th, 4-th F call, interesting things happens at the input of the 5-th F call, it was just the value of $((KE_8 >>> 15) \oplus (KE_{12} >>> 15)$, $(KE_9 >>> 15) \oplus (KE_{13} >>> 15)$, $(KE_{10} >>> 15) \oplus (KE_{14} >>> 15)$, $(KE_{11} >>> 15) \oplus (KE_{15} >>> 15))$, as after attacking the encryption procedure, we get the correct value of $(KE_8 \oplus KE_{12}, KE_9 \oplus KE_{13}, KE_{10} \oplus KE_{14}, KE_{11} \oplus KE_{15})$, so we can derive the input of the 5-th F call, then, after 5-th and $6^{th}$ F calls, we can get the value of $(KE_4, KE_5, KE_6, KE_7)$, as $KE_0 \oplus KE_4$, $KE1 \oplus KE_5$, $KE_2 \oplus KE_6$, $KE_3 \oplus KE_7$ is known, so $K_i$ ($i$=0,1,…15) is recovered, combing other analysis, we can recover full 192/256-bit key by directly derivations. So, obviously, attacking Camllia-192/256 is even much easier than Camellia-128.

### 7.2 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 6 rounds analysis results of $K_i \oplus KE_j$ (Table 4), unfortunately, this was enough for key Camellia-192/256 recovery directly, and after key schedule Cache timing attack, full key of Camellia-128 can be recovered very efficiently.

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 get the partial candidates before and after the rotating operation, it's very easy to recover the correct input and output of the rotating operation($KE_j$), as they are closely related with the *K*(Table 4, Table 5), so the adversary can recover K quite efficiently. Our experiments demonstrate that just analyzing one <<< operation can narrow down the key searching space from$2^{107.8}$ to 1. 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.

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 black-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 [37] by interleaving affine transformations and using de-linearization techniques. An improvement is explained in [38]. An implementation of AES is also given in [39] by representing it with a set of key-dependent look-up tables, also an improvement is explained in [40]. 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 [39]. None of the published papers have tried to implement white-box cryptography on Camellia so far, so it's hopeful to design a new white-box cryptography on Camellia to secure against Cache timing attacks in the nearby future.

## 8 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.

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

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

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

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

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

[37] 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.

[38] 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.

[39] 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.

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