Further Improved Differential Fault Analysis on Camellia by Exploring Fault Width and Depth

Xin-jie Zhao, Tao Wang

Abstract—In this paper, we present two further improved differential fault analysis methods on Camellia by exploring fault width and depth. Our first method broadens the fault width of previous Camellia attacks, injects multiple byte faults into the $r$th round left register to recover multiple bytes of the $r$th round equivalent key, and obtains Camellia-128,192/256 key with at least 8 and 12 faulty ciphertexts respectively; our second method extends fault depth of previous Camellia attacks, injects one byte fault into the $r$-2\textsuperscript{th} round left register to recover full 8 bytes of the $r$th round equivalent key, 5-6 bytes of the $r$-1\textsuperscript{th} round equivalent key, 1 byte of the $r$-2\textsuperscript{th} round equivalent key, and obtains Camellia-128,192/256 key with 4 and 6 faulty ciphertexts respectively. Simulation experiments demonstrate: due to its reversible permutation function, Camellia is vulnerable to multiple bytes fault attack, the attack efficiency is increased with fault width, this feature greatly improves fault attack’s practicalities; and due to its Feistel structure, Camellia is also vulnerable to deep single byte fault attack, 4 and 6 faulty ciphertexts are enough to reduce Camellia-128 and Camellia-192/256 key hypotheses to $2^{22\times}$ and $2^{21\times}$ respectively.

Index Terms—Differential fault analysis, Feistel structure, SPN structure, Camellia, Block cipher, Fault width and depth

1 INTRODUCTION

The idea of fault attack was first suggested Boneh et al. in 1997 [1]. Boneh et al. showed that this attack succeeded when it was applied to RSA based on the Chinese Remainder Theorem using a faulty ciphertext. Shortly after, Biham and Shamir proposed that the idea of fault analysis could be applied to symmetric-key cryptography DES and showed that the attack succeeded in obtaining an entire DES key [2]. They called this attack differential fault analysis (DFA), which is executed with some pairs of correct and faulty ciphertexts. Since that, many research papers have been published using this cryptanalysis technique to successfully attack various cryptosystems, including ECC [3], 3DES [4], AES [5,6], [7,8], [9,10], [11], Camellia [12], [13], [14], ARIA [15], CLEFIA [16,17], RC4 [18], [19], Trivium [20], [21] and so on.

Camellia is a 128-bit block cipher jointly developed by NTT and Mitsubishi Electric Corporation in 2000 [22]. It was chosen as a recommended algorithm by the NESSIE project in 2003 and certified as the IETF standard cipher for XML security URIs, SSL/TLS cipher suites and IPsec in 2005. In March 2009, Camellia was integrated into the OPENSSL-1.0.0-beta1. In Jan 2009, Zhou et al. proposed the first DFA on Camellia using approximately 64 pairs of correct and faulty ciphertexts to recover a 128-bit key and 96 pairs to recover 192/256-bit keys [12]. After one byte random fault injected into $r$\textsuperscript{th}, $r$-1\textsuperscript{th}, $r$-2\textsuperscript{th}, $r$-3\textsuperscript{th} etc round register, they simply extended DFA attack against DES in [2], many faulty pairs were needed to obtain Camellia key. In Dec 2009, Zhao et al. proposed an improved fault attack extending the fault depth of [12], and used 16 pairs of correct and faulty ciphertexts to recover a 128-bit key and 24 pairs to recover 192/256-bit keys [13]. They injected one byte fault into the $r$-1\textsuperscript{th}, $r$-2\textsuperscript{th}, $r$-3\textsuperscript{th} etc rounds, utilized the reverse feature of Camellia permutation function to recover 5-6 key byte at one time, and improved the key recovery efficiency. At the same time, Li et al. also proposed the same method as [13] in [14], used 30 pairs to recover Camellia 128/192/256-bit keys.

In this paper, we analyze the basic DFA attack principle and summarize the DFA on block ciphers with S-box into computing the S-box input and output differential problems, then present two further improved DFA methods on Camellia. Note that how to induce the specific fault is not covered in this paper, since this is not the main concern of our paper and many literatures on fault inductions are available [23].

Our first attack is based on the multiple byte faults model, which is verified by hardware fault attack experiments in [10] and [11] recently. In these two experiments, both SPN and Feistel structure block cipher can be injected fault with any width from 8-bit to 64-bit, sometimes even 128-bit. In this paper, we inject multiple byte faults into the $r$\textsuperscript{th}, $r$-1\textsuperscript{th}, $r$-2\textsuperscript{th}, $r$-3\textsuperscript{th} etc round left register instead of one byte in [12] and provide the complexity analysis of the attack. The proposed attack requires 16 pairs and 24 pairs to obtain 128-bit, 192/256 key with feasible calculation time and improve the fault analysis efficiency, attack in [12] is a specialized case of our attack when the faulty byte number is one, most importantly, multiple byte faults attack is more practical than single byte fault attack.

Our second attack further extends the fault depth of [13] and [14] from the $r$-1\textsuperscript{th} round to the $r$-2\textsuperscript{th} round. We inject multiple byte faults into the $r$-2\textsuperscript{th}, $r$-4\textsuperscript{th} etc round left register instead of the $r$-1\textsuperscript{th}, $r$-2\textsuperscript{th}, $r$-3\textsuperscript{th} etc round left register.
The proposed attack takes advantage of the generalized Feistel structure of Camellia, one byte fault in the r-th round can recover 8 bytes of the r-th round equivalent key, 5-6 bytes of the r-1th round equivalent key and 1 byte of the r-2th round equivalent key, so 4 and 6 faulty pairs are enough to recover Camellia-128,192/256 key hypotheses to 2^{22.2} and 2^{31.8} respectively. Compared with [12],[13],[14], Our DFA method not only enhances the fault depth, but also improves the fault analysis efficiency by 16 times and 4 times respectively, and decreases the faulty ciphertexts number. Besides, our second attack can be easily extended to DFA on Camellia key schedule case, while [12] can not.

As induction of faults requires high precision instruments (more the precision, more the cost!) and is harder to guarantee, an attack which requires large number of faulty pairs is impractical. We can, therefore, state that the most efficient attacks are those that require the fewest assumptions on the effect of a fault and least faulty ciphertexts. Our first attack looses the assumptions on the fault width to improve the fault injection practicability, the second attack deduces the number of faulty ciphertexts to improve the key retrieve efficiency, so we believe that both attacks are much more stronger than previous DFA attacks on Camellia.

This work is organized as follows. In Section 2, we present the basic DFA model and how it can be used into SPN and Feistel block ciphers. Section 3 presents the general overview of DFA on Camellia. Section 4 and Section 5 present several further improved DFA attacks on Camellia by broadening fault width and enhancing fault depth respectively. Section 6 displays the complexity analysis and experimental results of the attacks. Section 7 is the conclusion.

## 2 DFA Attack Model

Most block ciphers are composed of Substitution function S and Permutation function P. In DFA attacks, the adversary usually injects single byte fault before the final S function, after the S function, the S-box lookup index byte \( a \) becomes \( a' \), the S-box input differential value \( \Delta a \) (\( \Delta a=a \ominus a' \)) can be either known or unknown. But usually, the adversary can obtain the S-box output differential \( \Delta c \) by observing the correct and faulty ciphertexts. So, it always holds the following formula

\[
S[a] \oplus S[a \oplus \Delta a] = \Delta c
\]

(1)

The output of the S function usually has extra post-whitenings by Xored the last round key to generate the ciphertexts C. As C is known, if \( a \) is obtained, the last round key can be recovered. According to \( \Delta c \) is known or unknown, we present two DFA models for Feistel and SPN structure block ciphers.

### 1 \( \Delta a \) is known

This case is usually related with Feistel structure block cipher. If one byte fault \( \Delta a \) is injected into the r-th round left register, due to the feature of Feistel structure, both \( \Delta a \) and \( \Delta c \) can be obtained after analyzing the cipher differential \( \Delta c \). If we input every possible candidate of \( a \) to formula (1), we can get limited candidates of \( a \) satisfying formula (1).

Fig. 1 is the Camellia S-box and differential S-box \( (\Delta=1) \) elements sorted ascending.

### 2 \( \Delta c \) is unknown

This case is usually related with SPN structure block cipher. As to Feistel structure block cipher, when one byte fault is injected before the last permutation layer of the r-th round, and both the r-th round S-box input and output differential are known. However, as to SPN structure block cipher, when one byte fault is injected before the last permutation layer of the r-th round, after the permutation layer, \( m \) faulty state with the same differential fault value as \( \Delta a \) can be generated, but after the r-th S function, the faults are propagated into \( m \) differential faults, usuusly, the S-box output differential \( \Delta c \) is known, but the input differential \( \Delta a \) is unknown. In order to recover \( a \), \( \Delta a \) has to be guessed firstly. Suppose \( \Delta a \) is an 8-bit non-zero value, which has 255 candidates. Usually, the \( m \) different output differential values should be related with the same input differential S-box, if these 7 output values are not in the same differential S-box when \( \Delta a \neq n \), we can eliminate \( \Delta a = n \) using this technique, we can get limited candidates of \( \Delta a \), then case 2 \( (\Delta c \text{ is unknown}) \) can be transferred into Case 1 \( (\Delta a \text{ is known}) \), finally, after analyzing enough samples, \( a \) and the secret key can be obtained.

## 3 General Overview of DFA on Camellia

### 3.1 Description of Camellia

A full description of the Camellia cipher is provided in [22], but below is a brief description of the cipher’s properties that are utilized in this study.

**Encryption Procedure:** Camellia is an iterated cipher, its block length is 128-bit, and support 128,192,256-bit three key length types, it adopts the Feistel structure.
order to improve the security of the cipher, in the first and last round, the 128-bit data block is XORed with 128-bit pre-whitening and post-whitening keys, and the FL and FL₁ functions inserted every 6 rounds are used to provide non-regularity between the rounds. The Feistel structure for Camellia encryption can be written as follows:

\[
\begin{align*}
L_t &= R_{t-1} \oplus F(L_{t-1}, k_t) \\
R_t &= L_t
\end{align*}
\]

\(k_t\) denotes the \(t\)th round key, \(F = P \star S\) is the round function, the definition of the \(S\) and \(P\) function is

\[S : F^{64} \rightarrow F^{64}\]

\[l_{64}(0) || l_{64}(1) || l_{64}(2) || l_{64}(3) || l_{64}(4) || l_{64}(5) || l_{64}(6) \rightarrow l'_{64}(0) || l'_{64}(1) || l'_{64}(2) || l'_{64}(3) || l'_{64}(4) || l'_{64}(5) || l'_{64}(6) \]

\(\Delta P_{i} = s_{i}(l_{i}, k_{i}) = s_{i}(l_{i}) = s_{i}(l_{i+1})\)

\[l'_{4} = s_{i}(l'_{i}) \oplus s_{i}(l'_{i+1}) \oplus s_{i}(l'_{i+2}) = s_{i}(l_{i}) \oplus s_{i}(l_{i+1}) \oplus s_{i}(l_{i+2})\]

\[P : F^{64} \rightarrow F^{64}\]

\[z_{64}(0) || z_{64}(1) || z_{64}(2) || z_{64}(3) || z_{64}(4) || z_{64}(5) || z_{64}(6) \rightarrow z'_{64}(0) || z'_{64}(1) || z'_{64}(2) || z'_{64}(3) || z'_{64}(4) || z'_{64}(5) || z'_{64}(6) \]

\[z'_{0} = z_{0} \oplus z_{1} \oplus z_{2} \oplus z_{3} \oplus z_{4} \oplus z_{5} \oplus z_{6} \oplus z_{7} = z_{0} \oplus z_{1} \oplus z_{2} \oplus z_{3} \oplus z_{4} \oplus z_{5} \oplus z_{6} \oplus z_{7}\]

\[z'_{i} = z_{i} \oplus z_{i+1} \oplus z_{i+2} \oplus z_{i+3} \oplus z_{i+4} \oplus z_{i+5} \oplus z_{i+6} \oplus z_{i+7} = z_{i} \oplus z_{i+1} \oplus z_{i+2} \oplus z_{i+3} \oplus z_{i+4} \oplus z_{i+5} \oplus z_{i+6} \oplus z_{i+7}\]

\[z'_{18} = z_{18} \oplus z_{19} \oplus z_{20} \oplus z_{21} \oplus z_{22} \oplus z_{23} \oplus z_{24} \oplus z_{25} = z_{18} \oplus z_{19} \oplus z_{20} \oplus z_{21} \oplus z_{22} \oplus z_{23} \oplus z_{24} \oplus z_{25}\]

**Key Schedule:** Firstly, 4 12-bit variable \(K_{0}, K_{8}, K_{16}\), and \(K_{24}\) are generated from the initial key and 6 64-bit constant \(\Sigma_{i} (i = 1, 2, ..., 6)\) by several F functions, then the 64-bit sub-keys \(k_{i}, k_{i+1}\), and \(k_{i+6}\) are generated from \(K_{i}, K_{i+8}, K_{i+16}\), and \(K_{i+24}\). Specific procedure is provided in [22].

**3.2 Basic Assumption and Notation**

1. **Assumption:**
   (1) One byte or more bytes random fault is induced into the memory registers storing the intermediate results in one fault induction. Notice that the attacker knows neither the location nor the concrete value of the fault.
   (2) For any one plaintext adaptively selected, two different ciphertexts under the control of the same secret key are available, the right ciphertext and the faulty one.
   (3) The faulty ciphertexts of the required type are presumably available. How to induce the specific fault is not covered in this paper, since this is not the main concern of our paper and many literatures on fault inductions are available [23]. The attacker should be able to identify the required faulty ciphertexts from a mass of faulty ciphertexts and discard faults occurring at a wrong timing.
   (4) Only one master key is used during one attack.

2. **Notation:**
   (1) \(K_{i}\): the equivalent subkey of the \(i\)th round, also is the exclusive OR half of the post-whitening subkey and the \(i\)th subkey. In case of Camellia-128, the equivalent subkey for the 16th, 17th, 16th, 15th, 14th, 13th round is \(K_{16} = k_{00} \oplus k_{01}\), \(K_{17} = k_{02} \oplus k_{03}\), \(K_{18} = k_{04} \oplus k_{05}\), \(K_{19} = k_{06} \oplus k_{07}\), \(K_{20} = k_{08} \oplus k_{09}\), \(K_{21} = k_{10} \oplus k_{11}\), \(K_{22} = k_{12} \oplus k_{13}\), \(K_{23} = k_{14} \oplus k_{15}\), and \(K_{24} = k_{16} \oplus k_{17}\) respectively.
   (2) \(L_{r-1}, R_{r-1}\): the 64-bit left and right half of the \(r\)th round inputs.
   (3) \(k_{r}\): the 64-bit \(r\)th round subkey.
   (4) \(\Delta L_{r}, \Delta R_{r}\): the \(i\)th byte of the \(r\)th round left and right half input differential value. \(i \in [0, 7]\)
   (5) \(\Delta L_{r}^{0}, \Delta R_{r}^{0}\): the \(i\)th byte of the \(r\)th round left and right half output differential value. \(i \in [0, 7]\)
   (6) \(\Delta S_{r}^{0}, \Delta P_{r}^{0}\): the \(i\)th byte of the \(r\)th round \(S\) function and \(P\) function output differential value. \(i \in [0, 7]\)
   (7) \(\Delta CL, \Delta CR\): the \(i\)th byte of the left and right half ciphertext differential value. \(i \in [0, 7]\)
   (8) Fault: If not specially stated, fault denotes the non-zero differential value except for the faulty ciphertext.

**3.3 Main idea of DFA on Camellia**

The main idea of DFA on Camellia is as follows:

1. Choose any plaintext \(P\), and obtain the corresponding correct ciphertext \(C\).
2. Inject specific fault into the encryption procedure or key schedule, and obtain the faulty ciphertext \(C'\).
3. Deduce one byte or several bytes of \(K_{r}\) using DFA technique.
4. Repeat the above steps, until full \(K_{r}\) is recovered.
5. Proceed in the same way and attack the previous round, and deduce \(K_{r-1}, K_{r-2}, K_{r-3}, ...\), accordingly.
6. Recover Camellia-128 key by \(K_{r-5}, K_{r-4}, K_{r-3}, K_{r-2}, K_{r-1}, K_{r}\) and Camellia-192/256 key by \(K_{r-5}, K_{r-4}, K_{r-3}, K_{r-2}, K_{r-1}, K_{r}\) with key reversion techniques.
7. Verify the correctness of the recovered Camellia key. In the next Sections, two improved differential fault analysis methods on Camellia by broadening fault width and enhancing fault depth are described, and the experimental results and comparisons are given to prove the correctness of the analysis theory.

**4 IMPROVED DFA ON CAMELLIA BY BROADENING FAULT WIDTH**

**4.1 Previous Study**

Zhou’s attack [12] is a generic attack based on model of Section 2. Its main idea is to inject single byte fault on the \(r\)th round left register \(L_{r-1}\) and use equation (1) to retrieve \(K_{r}\). Let’s take recovering \(K_{18}\) as an example, the fault propagation is depicted in Fig. 2.

![Fig. 2. Fault propagation of one byte fault in L₁₇.](image)

The adversary first induces one byte fault \(\Delta IL_{18}^{0}\) to \(L_{17}^{0}\), after the \(S\) function, \(\Delta IL_{18}\) is transferred to single byte fault \(\Delta S_{i}^{0}\), after the \(P\) function, 5 or 6 bytes of \(\Delta P_{18}\) have the same fault as \(\Delta S_{i}^{0}\), after the final swap and exclusive OR of \(k_{03}\) and \(k_{04}\), \(\Delta CL\) is equal to \(\Delta IL_{18}\) and also is the \(S\) func-
tion input differential, ΔCR is equal to ΔP18 and its non-zero value is also the S function output differential. By applying DFA methods of Section 2, the adversary can recover K18. Note that one byte fault can only recover one K18 byte from 256 to 2-4 candidates, and two times of the same location single byte fault can recover one K18 byte, so at least 16 faults are needed to recover K18. By applying this method to the 17th, 16th, 15th round, K17, K16, K15 can be recovered, combing the key reverse techniques, the initial key K can be obtained.

4.2 Improved DFA on Camellia by Broadening Fault Width

In this section, we suppose the adversary has the ability of injecting multiple byte faults into L-1, this is much more practical than single byte fault attack in [12], also has been verified by hardware fault attack experiments in [11] recently. Let’s take injecting m (1≤m≤8) faults into the 18th round left register L17 to recover m bytes of K18 as an example, the fault propagation is depicted in Fig. 3.

![Fig. 3. Fault propagation of multiple byte faults in L17.](image)

Specific attacking procedure is as follows:

1. Choose random plaintext P and obtain the correct ciphertext C under the secret key K.
2. Induce m bytes random faults ΔL18 into L17, and obtain the faulty ciphertext C*.
3. Deduce the fault location.
   Different fault locations injected into L17 can propagate the same location faults into C1 according to the nonzero byte of ΔC1, the attacker can easily identify the fault location injected into L17.
4. Compute the 18th round S-box input and output differential ΔIL18 and ΔS18.
   According to Fig.3, the 18th round S-box input differential ΔIL18 is equal to the left half ciphertext differential ΔC1, the 18th round S-box output differential ΔS18 can be computed by the Camellia reverse P function

\[
\Delta S_{18}^{0i} = \Delta P_{18}^{0} \oplus \Delta P_{18}^{1} \oplus \Delta P_{18}^{2} \oplus \Delta P_{18}^{3} \oplus \Delta P_{18}^{4} \oplus \Delta P_{18}^{5} \oplus \Delta P_{18}^{6} \oplus \Delta P_{18}^{7},
\]

\[
\Delta S_{18}^{1i} = \Delta P_{18}^{0} \oplus 2 \Delta P_{18}^{1} \oplus \Delta P_{18}^{2} \oplus \Delta P_{18}^{3} \oplus \Delta P_{18}^{4} \oplus \Delta P_{18}^{5} \oplus \Delta P_{18}^{6} \oplus \Delta P_{18}^{7},
\]

\[
\Delta S_{18}^{2i} = \Delta P_{18}^{0} \oplus \Delta P_{18}^{1} \oplus \Delta P_{18}^{2} \oplus \Delta P_{18}^{3} \oplus \Delta P_{18}^{4} \oplus \Delta P_{18}^{5} \oplus \Delta P_{18}^{6} \oplus \Delta P_{18}^{7},
\]

\[
\Delta S_{18}^{3i} = \Delta P_{18}^{0} \oplus \Delta P_{18}^{1} \oplus \Delta P_{18}^{2} \oplus \Delta P_{18}^{3} \oplus \Delta P_{18}^{4} \oplus \Delta P_{18}^{5} \oplus \Delta P_{18}^{6} \oplus \Delta P_{18}^{7},
\]

(5) Recover K18.

From step (1)-(4), we can recover the m bytes input and output differential of the 18th round S function ΔIL18, ΔS18 using DFA model of Section 2, it’s easy to recover m bytes S function input value, which can be expressed as L17 ⊕ k18, as L17 ⊕ k05 = C1, C1 is known, so m bytes of K18 can be recovered. Repeat above steps to recover full 64-bit K18.

6) Recover K17, K16, K15...etc equivalent subkeys.
   Proceed in the same way and attack, in turn, deduce the equivalent subkeys K2-3, ... accordingly, and retrieve the initial Camellia-128/192/256 key with methods in [13].

5 Improved DFA on Camellia by Extending Fault Depth

5.1 Previous Study

In Dec 2009, Zhao et. al. proposed an improved fault attack extending the fault depth of [12] in [13], the same method was also proposed in [14] . They supposed the adversary has the ability of injecting single byte fault into the r-1th round left Camellia register Lr-2. Let’s take injecting one byte fault into the 17th round left register L16 to as an example, the fault propagation is depicted in Fig. 4.

![Fig. 4. Fault propagation of one byte fault in L16.](image)

Specific attacking procedure is as follows:
(1) Choose random plaintext $P$ and obtain the correct ciphertext $C$ under the secret key $K$.

(2) Induce one byte fault $\Delta L_{17}$ into $L_{16}$, and obtain the faulty ciphertext $C^*$ under the secret key $K$.

(3) Deduce the fault location.

Different fault locations injected into $L_{16}$ can generate different indices sets of the fault $C_L$, we can identify the fault location by methods in [13].

(4) Compute the 18th round S-box input and output differential $\Delta L_{18}$ and $\Delta S_{18}$.

$\Delta L_{18}$ is equal to the left half ciphertext differential $\Delta C_L$, $\Delta O_{18}$ is equal to the left half ciphertext differential $\Delta C_R$, and is generated by $\Delta S_{18}^0, \Delta S_{18}^1, \Delta S_{18}^2, \Delta S_{18}^3, \Delta S_{18}^4$ and $\Delta IL_{18}^{i/0}$ using 8 equations. It's simple to recover these 6 unknown differentials ($\Delta S_{18}^0, \Delta S_{18}^1, \Delta S_{18}^2, \Delta S_{18}^3, \Delta S_{18}^4$ and $\Delta IL_{18}^{i/0}$) by equation (3).

\[
\begin{align*}
\Delta O_{18}^0 &= \Delta L_{17}^0 \oplus \Delta S_{18}^0 \oplus \Delta S_{18}^0 \oplus \Delta S_{18}^0 \\
\Delta O_{18}^1 &= \Delta S_{18}^0 + \Delta S_{18}^1 \oplus \Delta S_{18}^1 + \Delta S_{18}^1 \\
\Delta O_{18}^2 &= \Delta S_{18}^1 + \Delta S_{18}^2 \oplus \Delta S_{18}^2 + \Delta S_{18}^2 \\
\Delta O_{18}^3 &= \Delta S_{18}^2 + \Delta S_{18}^3 \oplus \Delta S_{18}^3 + \Delta S_{18}^3 \\
\Delta O_{18}^4 &= \Delta S_{18}^3 + \Delta S_{18}^4 \oplus \Delta S_{18}^4 + \Delta S_{18}^4 \\
\Delta O_{18}^5 &= \Delta S_{18}^4 + \Delta S_{18}^5 \oplus \Delta S_{18}^5 + \Delta S_{18}^5 \\
\Delta O_{18}^6 &= \Delta S_{18}^5 + \Delta S_{18}^6 \oplus \Delta S_{18}^6 + \Delta S_{18}^6 \\
\Delta O_{18}^7 &= \Delta S_{18}^6 + \Delta S_{18}^7 \oplus \Delta S_{18}^7 + \Delta S_{18}^7 \\
\Delta IL_{18}^{i/0} &= \Delta O_{18}^0 \oplus \Delta O_{18}^1 \oplus \Delta O_{18}^2 \oplus \Delta O_{18}^3 \\
\Delta IL_{18}^{i/0} &= \Delta S_{18}^0 + \Delta S_{18}^1 \oplus \Delta S_{18}^2 + \Delta S_{18}^3 \oplus \Delta S_{18}^4 + \Delta S_{18}^5 \oplus \Delta S_{18}^6 + \Delta S_{18}^7 \\
\Delta IL_{18}^{i/0} &= \Delta S_{18}^0 + \Delta S_{18}^1 \oplus \Delta S_{18}^2 + \Delta S_{18}^3 \oplus \Delta S_{18}^4 + \Delta S_{18}^5 \oplus \Delta S_{18}^6 + \Delta S_{18}^7 \\
\end{align*}
\]

(3) Deduce the 17th round S-box lookup output differential $\Delta CL$.

$\Delta CL = \Delta L_{18} = P(\Delta S_{17}) \oplus \Delta R_{18} = P(\Delta S_{17}) \oplus \Delta IL_{18}^{i/0}$

(4) Deduce the 17th round S-box lookup output differential $\Delta S_{17}$.

Next, we should recover $\Delta IL_{17}$ from $\Delta S_{17}$. First we compute the 255 Camellia differential S-boxes, for each input differential S-box value $\epsilon (\epsilon = \Delta IL_{18}^{i/0}$ from 1 to 255, we can get 4 type of differential Camellia S-boxes, if 5 nonzero bytes of $\Delta S_{17}$ are among these 4 differential S-boxes (satisfying equation (5)), we can get one $\Delta IL_{17}^{i/0}$ candidate, else $\epsilon$ will be eliminated, after 255 iterations, the adversary can get limited $\Delta IL_{17}^{i/0}$ candidates.

\[
\begin{align*}
\Delta S_{17}^0 &= \Delta S_{17}^0 \oplus \Delta S_{17}^0 \oplus \Delta S_{17}^0 \oplus \Delta S_{17}^0 \\
\Delta S_{17}^1 &= \Delta S_{17}^0 + \Delta S_{17}^1 \oplus \Delta S_{17}^1 \oplus \Delta S_{17}^1 \\
\Delta S_{17}^2 &= \Delta S_{17}^1 + \Delta S_{17}^2 \oplus \Delta S_{17}^2 \oplus \Delta S_{17}^2 \\
\Delta S_{17}^3 &= \Delta S_{17}^2 + \Delta S_{17}^3 \oplus \Delta S_{17}^3 \oplus \Delta S_{17}^3 \\
\Delta S_{17}^4 &= \Delta S_{17}^3 + \Delta S_{17}^4 \oplus \Delta S_{17}^4 \oplus \Delta S_{17}^4 \\
\Delta S_{17}^5 &= \Delta S_{17}^4 + \Delta S_{17}^5 \oplus \Delta S_{17}^5 \oplus \Delta S_{17}^5 \\
\Delta S_{17}^6 &= \Delta S_{17}^5 + \Delta S_{17}^6 \oplus \Delta S_{17}^6 \oplus \Delta S_{17}^6 \\
\Delta S_{17}^7 &= \Delta S_{17}^6 + \Delta S_{17}^7 \oplus \Delta S_{17}^7 \oplus \Delta S_{17}^7 \\
\end{align*}
\]

(5) Recover 5 or 6 bytes of $K_{18}$.

By applying the general DFA model of Section 2, it’s easy to recover 5 bytes $S$ function input value of the 18th round, which can be expressed as $L_{17}^i \oplus k_{18}^i (i=0,1,2,4,7)$, as $L_{17}^i \oplus k_{18}^i = C_{17}$, $C_1$ is known, $k_{18}^i \oplus k_{18}^i (i=0,1,2,4,7)$ can be recovered.

(6) Recover full 8 bytes of $K_{18}$ and several bytes of $K_{17}$.

Repeat step (1)-(5) to recover full 8 bytes of $K_{18}$. Note that after $K_{18}$ is recovered, as $\Delta L_{17}$ is deduced, the output 17th round S-box differential $\Delta S_{17}$ can be obtained through $\Delta CL$, the adversary can recover several bytes of $K_{17}$.

(7) Proceed in the same way as step (1)-(6) and attack the previous round, and deduce the equivalent subkeys $K_{17}, K_{16}, K_{15}, \ldots$, accordingly, and retrieve the initial Camellia-128/192/256 key with methods in [13].

5.2 Improved DFA on Camellia by Extending Fault Depth

In this section, we describe idea of proposed DFA method to retrieve Camellia round keys by analyzing S-box input and output differentials.

Suppose the adversary has the ability of injecting single byte fault into the r-2th round left Camellia register $L_{r-3}$, let’s take injecting one byte fault into the 16th round left register $L_{15}$ as an example, the fault propagation is depicted in Fig. 5.

Specific attacking procedure is as follows:

(1) Choose random plaintext $P$ and obtain the correct ciphertext $C$ under the secret key $K$.

(2) Induce random single byte fault $\Delta L_{16}$ into $L_{15}$, obtain the faulty ciphertext $C^*$.

(3) Compute the 18th round S-box lookup input differential $\Delta IL_{18}$.

$\Delta IL_{18}$ can be computed from the left half ciphertext differential $\Delta CL$.

(4) Deduce the 17th round S-box lookup input differential $\Delta IL_{17}$.

(5) Deduce the 17th round S-box lookup output differential $\Delta CL$.

$\Delta CL = \Delta L_{18} = P(\Delta S_{17}) \oplus \Delta R_{18} = P(\Delta S_{17}) \oplus \Delta IL_{18}$

(6) Deduce the 18th round S-box lookup output differential $\Delta S_{18}$.

In order to recover $K_{18}$, the adversary should recover the 18th round S-box lookup output differential $\Delta S_{18}$.

$\Delta CR = \Delta O_{18} = P(\Delta S_{18}) \oplus \Delta R_{18} = P(\Delta S_{18}) \oplus \Delta IL_{18}$

The equation (6) can be transferred into equation (7): $\Delta S_{18} = P^{-1}(\Delta CR \oplus \Delta IL_{18})$.
6 complex Analysis and experimental Results

6.1 Complexity Analysis

Due to the limit abilities of the attacker, it's very difficult to induce accurate and effective faults for DFA, so how many faulty ciphertexts are needed to crack the cipher is also very crucial. Next, we make a sketch of this complexity analysis.

1. Complexity Analysis of one byte fault analysis

We describe the characteristics of the equation for the S-box in the deterministic methods. Let's just consider the simple one byte S-box model shown in Fig. 2. When we know the input S-box differential $\Delta L_{16}^0$ and the output differential $\Delta S_{16}^0$, we can get a set of $L_{17}^0$ satisfying equation (8).

$$S[L_{17}^0 \oplus k_{18}^0] \oplus S[L_{17}^0 \oplus k_{18}^0 \oplus \Delta L_{16}^0] = \Delta S_{16}^0$$  \hspace{1cm} (8)

The number of $K_{18}^0$ depends on $C^0$, $\Delta L_{16}^0$, $\Delta S_{16}^0$ and the S-box. In Camellia, four S-boxes $S_0$, $S_1$, $S_2$, $S_3$ are used in the $F$ function. By solving equation (8), we examine the size of the key candidates $|K_{18}^0|$ for all combinations of $C^0$, $\Delta L_{16}^0$, $\Delta S_{16}^0$ related with $S_0$. The total number of combinations is 16711680($\Delta L_{16}^0$ is a non-zero value). The results are shown in Table 1, it’s clear to see that by injecting single byte fault into $L_{17}^0$, about 2.0312 $K_{18}^0$ candidates can be obtained at one time, and the statistics of other three S-boxes are the same as $S_0$.

**Table 1**

<table>
<thead>
<tr>
<th>$K_{18}^0$</th>
<th>Number</th>
<th>$P$</th>
<th>$E(K_{18}^0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16450560</td>
<td>0.9844</td>
<td>1.9688</td>
</tr>
<tr>
<td>4</td>
<td>261120</td>
<td>0.0156</td>
<td>0.0624</td>
</tr>
<tr>
<td>Total</td>
<td>16711680</td>
<td>1</td>
<td>2.0312=2^{1.0}</td>
</tr>
</tbody>
</table>

2. Complexity analysis of attack in Section 4.2.

Suppose the adversary injects $m$ byte faults into $L_{r-1}$, according to Section 4.2, $m$ bytes of $K_r$ can be obtained. If $m=8$, one time 8-bytes faults in $L_{r-1}$ can reduce $K_r$ space from $2^{64}$ to 289.75 (2.0312$^8$). As almost two times of the same index fault byte can recover one key byte, so theoretically speaking, the relationship between faulty number $N$ and $m$ obtaining unique $K_i$ is

$$N = 16/m$$ \hspace{1cm} (9)

It’s clear to see that the key recovery efficiency is increased with the faulty bytes width, and two times full 8 faulty bytes of $L_{r-1}$ can recover $K_r$.

3. Complexity analysis of attack in Section 5.2.

Injecting one byte fault in the $r$-th round can propagate 5-6 faulty bytes in the $r$-th round and 8 faulty bytes in the $r$-th round, combined the analysis of Camellia S-box statistics and fault propagation feature, 2 faults are enough to recover $K_r$ and reduce the search space of $K_{r-1}$ from $2^{64}$ to about $2^{10.6}(1555.56)$ on average with 87.5% probabilities (if the two fault indices in the $r$-th round are not identical), 3 faults are enough to recover $K_r$ and reduce the search space of $K_{r-1}$ from $2^{64}$ to about $2^{3.8}$ on average with 98.4% probabilities (if all of the three fault indices in the $r$-th round are not identical). As to Camellia-128 with and without FL/FL$^{-1}$ layer, 2 faults in each of the 16th, 14th round, 4 faults are enough to reduce Camellia-128 key searching space to $2^{22.2}$ to $2^{21.2}$. As to Camellia-192/256 without FL/FL$^{-1}$ layer, 2 faults in each of the 16th, 14th, 12th round, 6 faults are enough to reduce Camellia-192/256 key searching space to $2^{21.8}$.

Note that if the attacker does not need to know the specific fault byte location, as for times random fault injected into $L_{15}$, there are 64 fault location combinations. Only the correct combination can recover unique $K_{18}$, and the wrong combinations usually get empty candidates of
6.2 Experimental Results and Comparisons

We have implemented simulations of the attacks in this paper written by Visual C++6.0 on Windows XP. Our simulations run on a personal computer (Athlon 64-bit 3000+ 1.81 GHz CPU and 1GB RAM) and successfully extract the Camellia-128/192/256 key.

In order to verify the complexity analysis of attack in Section 4.2 and 6.1, we implemented the 8 bytes width fault attack on Camellia 18th round, the statistics of 22 sets for 2000 sample’s average 8 byte faults in $L_{17}$ and $K_{18}$ candidat number is depicted in Fig. 6. It’s clear to see that one time 8 bytes fault on $L_{17}$, on average 289.74 candidates of $K_{18}$ are obtained, which is almost the same as the 289.75 of the theory value in Section 6.1.

Fig. 6. 22 sets of Statistics on $K_{18}$ candidate number.

In order to verify the complexity analysis of attack in Section 5.2 and 6.1, we injected two times single byte fault to $L_{15}$ (two fault indices are not identical), used the analysis method of Section 5.2, unique $K_{18}$ can be obtained, and the statistics of 10 sets for 5000 sample’s $K_{17}$ candidate number is depicted in Fig. 7. It’s clear to see that two times single byte fault on $L_{15}$, on average 1557.01 candidates of $K_{17}$ are obtained, which is almost the same as 1555.56 of the theory value in Section 6.1.

Fig. 7. 10 sets of Statistics on $K_{17}$ candidate number.

Also, the experimental results in Table 2 strongly support the complexity analysis presented in Section 6.1.

It’s clear to see that our methods are more effective than previous attacks and have the following properties:

1. Firstly, we broaden the fault depth of [10].
2. Secondly, we extend the fault depth of [10],[13],[14].
3. Thirdly, our DFA methods on Camellia encryption procedure in Section 5.2 can be easily adapted into DFA on Camellia key schedule, while [10] can not. It’s impossible for [10] to inject fault into $k_r$ to recover $K_r$ (the adversary can not get the $r$th round S-box input differential), however, according to the analysis of Section 5.2, it’s easy to inject fault into $k_{r-1}$ and $k_{r-2}$ to recover $K_r$.

<table>
<thead>
<tr>
<th>Camellia</th>
<th>Attack</th>
<th>Fault Type</th>
<th>Fault Location</th>
<th>$F/L$</th>
<th>$FL^I$</th>
<th>Fault Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>[10]</td>
<td>Single byte</td>
<td>$L_{14}$-$L_{17}$</td>
<td>×/√</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Section 4.2</td>
<td>Multiple bytes</td>
<td>$L_{14}$-$L_{17}$</td>
<td>×/√</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>[13]</td>
<td>Single byte</td>
<td>$L_{13}$-$L_{16}$</td>
<td>×/√</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Section 5.2</td>
<td>Single byte</td>
<td>$L_{13}$-$L_{16}$</td>
<td>×/√</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>[13]</td>
<td>Single byte</td>
<td>$k_{14}$-$k_{17}$</td>
<td>×/√</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>[14]</td>
<td>Single byte</td>
<td>$k_{12}$-$k_{17}$</td>
<td>×</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>Section 5.2</td>
<td>Single byte</td>
<td>$k_{14}$-$k_{16}$</td>
<td>×/√</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>[10]</td>
<td>Single byte</td>
<td>$L_{12}$-$L_{17}$</td>
<td>×/√</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>Section 4.2</td>
<td>Multiple bytes</td>
<td>$L_{12}$-$L_{17}$</td>
<td>×/√</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>[13]</td>
<td>Single byte</td>
<td>$L_{11}$-$L_{16}$</td>
<td>×</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>Section 5.2</td>
<td>Single byte</td>
<td>$L_{11}$-$L_{16}$</td>
<td>×/√</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>[13]</td>
<td>Single byte</td>
<td>$k_{12}$-$k_{17}$</td>
<td>×/√</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>[14]</td>
<td>Single byte</td>
<td>$k_{12}$-$k_{17}$</td>
<td>×</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>Section 5.2</td>
<td>Single byte</td>
<td>$k_{12}$-$k_{16}$</td>
<td>×/√</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>[13]</td>
<td>Single byte</td>
<td>$L_{11}$-$L_{16}$</td>
<td>√</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>Section 5.2</td>
<td>Single byte</td>
<td>$L_{11}$-$L_{16}$</td>
<td>√</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>192/256</td>
<td>Section 5.2</td>
<td>Single byte</td>
<td>$k_{12}$-$k_{16}$</td>
<td>√</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

7 Conclusion

Current studies of fault analysis on block ciphers are devoted to mathematical analysis on cryptographic algorithms, fault injection and detection of cryptographic algorithms in software and hardware implementation. In this paper, we examine the mathematical analysis of Camellia with its fault injection simulation in software implementation. For the hardware situation, we will leave it for the future research.

In this paper we present two improved DFA methods on Camellia. Our methods not only broaden the fault

$K_{18}$. So, we can use this feature to deduce the fault location of these 2 faults, recover unique $K_{18}$, and choose corresponding method to recover 5-6 bytes of $K_{17}$.
width, expand the fault depth, but also improve the efficiency of fault injection and decrease the number of faulty ciphertexts. Our best results demonstrate that 4 faults are enough to reduce Camellia-128 key searching space to $2^{22.2}$, 6 faults are enough to reduce Camellia-192/256 key searching space to $2^{31.8}$. Besides, our attack model can be adapted into most block ciphers with S-box, such as AES, ARIA, CLEFIA, SMS4, HYRAL, and MIBS.

Future analysis should be able to supply fault injection and detection of Camellia in hardware implementation. Moreover, we are working on the generic DFA sample size analysis model for block ciphers using S-box.

**ACKNOWLEDGMENT**

The authors wish to thank the anonymous reviewers for many helpful comments and suggestions. This work was supported in part by National Natural Science Foundation of China (Grant No. 60772082) and the Natural Science Foundation of Hebei Province. China (Grant No. 08M010).

**REFERENCES**


Xin-jie Zhao, Tao Wang: Department of Computer Engineering, Ordnance Engineering College, Shijiazhuang 050003, China, zhaox-injieem@163.com.