Differential Fault Analysis by Reducing AES Round

[1]KiSeok Bae, [2]SangJae Moon, [3]DooHo Choi, [4]YongJe Choi, [5]DooSik Choi, [6*]JaeCheol Ha
[1,2]Graduate School of Electrical Engineering and Computer Science, Kyungpook National Univ., Daegu, 702-701, Korea, gith@ee.knu.ac.kr, sjmoon@ee.knu.ac.kr
[3,4]Information Security Research Division, ETRI, Daejeon, 305-700, Korea dhchoi@etri.re.kr, choiyj@etri.re.kr
[5,6* Corresponding Author] *Dept. of Information Security, Hoseo Univ., Asan, Chungnam, 336-795, Korea, entlr6424@nate.com, jcha@hoseo.edu*

Abstract

This paper presents a practical differential fault analysis method for the Advanced Encryption Standard (AES) with a reduced round using a semi-invasive fault injection. We adapt the round reduction fault technique on AES in order to skip the last iterations of 'for' loop procedure. We can deduce the AES 128-bit secret key using 2^{16} exhaustive searches with two pairs of correct and faulty ciphertexts. We also verified the feasibility of our proposed DFA by a fault injection experiment on an ATmega128 microcontroller chip.

Keywords: *Fault Injection Attack, AES, Round Reduction*

1. Introduction

Since the discovery of side-channel leakage, a considerable amount of research has been performed on techniques to retrieve secret information through physical access to a device [1-3]. Among physical attacks on hardware implementation of cryptographic algorithm, the Differential Fault Analysis (DFA) attack extracts the secret key by analyzing the differences between correct and faulty ciphertexts. A faulty ciphertext can be obtained from malicious fault injection during execution of an algorithm. Since Biham and Shamir introduced the DFA attacks on an implementation of DES with errors induced by fault injection [4], many researchers have mounted several DFAs on symmetric key encryption algorithms [5-7]. In particular, AES algorithm has been considered a target of DFA attack because it is a FIPS standard of encryption algorithm [8-13].

Most DFAs for the AES can be roughly classified into two types according to the assumption of the fault model. In the first type of DFA, the intermediate data of the encryption or key expansion process can be corrupted by fault injection [8-11]. Then, the difference of the fault ciphertexts which injected a fault and diffused the corruption of data during the encryption or key expansion process is analyzed to obtain the secret key.

The second type of DFA does not assume data corruption during the round function process but does assume a fault at instruction code. The DFA by H. Choukri and M. Tunstall reduces the number of AES rounds to single round [12]. They injected a fault to corrupt iterative operation of AES round instead of data corruption. However, their implementation of AES on the target device is a naive and weak structure. Their implementation structure of AES is not the one recommended in the FIPS 197 standard documentation [13]. Recently, Park et al. proposed a DFA method for AES recommended in FIPS 197 by reducing the number of the AES rounds [14]. They successfully extracted the 128-bit secret key using 10 ciphertexts obtained from round-reduced AES.

In this paper we present a practical DFA method for a faulty AES with reduced round by omitting just last iteration of 'for' loop procedure by fault injection. To verify the feasibility of the proposed method, we implemented two types of AES algorithm recommended in FIPS 197 on the target ATmega128 microcontroller. Then, we conduct a fault occurrence in iterative operation of AES encryption using a laser beam on the decapsulated-surface of target chip. Using just two pairs of correct and faulty ciphertexts, our proposed attack can retrieve the AES 128-bit secret key with about 2^{16} exhaustive searches.

2. Preliminaries

2.1. The Advanced Encryption Standard (AES)

The FIPS-approved cryptographic algorithm AES can be used to protect electronic data [13]. The AES is a symmetric block cipher algorithm that can encrypt and decrypt a data block of 128 bits. The AES algorithm is able to use secret keys of 128, 192 or 256 bits. We focus on the 128-bit key AES due to its popularity and simple description. The total number of rounds of this AES is 10, and the intermediate 16-byte values of the encryption or decryption process are called by the state. This 128-bit intermediate data of AES is usually represented by a 4x4 matrix.

AES Encryption Process Each round of AES encryption process is composed of the following 4 transformation functions:

- *SubBytes*: This is a nonlinear byte substitution that operates independently on each byte of the state. Here, we denote this function as **SB**. And we denote Inverse SubBytes as **SB^{-1}**.
- *ShiftRows*: Each row of the state is cyclically shifted with different offsets. We now denote ShiftRows and its inverse as **SR** and **SR**$^{-1}$ respectively.
- *MixColumns*: This is a linear transformation of each column of the state. We denote this function as **MC** and its inverse as **MC**$^{-1}$.
- *AddRoundKey*: This is a bitwise XOR operation with each 128-bit round key.

AES Key Expansion Process The 128-bit AES algorithm takes the 128-bit master secret key, denoted as SK, and performs a key expansion process to generate 11 round keys. Each 128-bit round key is composed of 4-byte words, denoted *W*[*i*]. The AES key expansion process consists of three transformation functions as follows:

- *RotWord*: This function takes a word $[a_0,a_1,a_2,a_3]$ as an input, performs a cyclic permutation, and returns the word $[a_1,a_2,a_3,a_0]$.
- *SubWord*: This is a function that takes a word composed of 4 bytes and applies the **SB** to each byte.
- *Rcon*[*i*]: This is a round constant word given by $[x^{i-1}$,{00},{00},{00}]. Here, is representing powers of *x* (*x* is denoted as {02}) in the field GF(28).

```
state = M
AddRoundKey(state, &w[0])
for i = 1 step 1 to 9
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, &w[i*4])
end for
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, &w[40])
```

Figure 1. Pseudo code for 128-bit AES encryption (Type-1)

Fig. 1 shows the pseudo code of the AES encryption algorithm described in the standard document [13]. The important feature of the pseudo code is that the last round of AES was not included in the "for" loop procedure.

2.2. DFA for AES with a Fault to Iterative Operation

In FDTC 2005, H. Choukri and M. Tunstall proposed a new DFA on AES by reducing the number of rounds using a fault injection method [12]. In their experiment on a smart card using PIC16F877, they chose the simplest case of reducing the number of rounds to one in order to facilitate subsequent cryptanalysis. As shown in Fig. 2, their implementation of AES algorithm is slightly different from the one described in the AES FIPS standard document. The last round for omitting *MixColumns* using a conditional branch is included into a 'for' loop procedure which is the main programming structure to execute the AES rounds.

```
state = M
AddRoundKey(state, &w[0])
for i = 1 step 1 to 9
    SubBytes(state)
    ShiftRows(state)
    If( i != 10) MixColumns(state)
    AddRoundKey(state, &w[i*4])
end for
```

Figure 2. Pseudo code for 128-bit AES encryption (Type-2)

Thus, their AES implementation could be reduced to the initial *AddRoundKey* process and first round by a fault and is composed of the following 5 sequential functions:

- *AddRoundKey*(); //the initial round key addition,
- *SubBytes*(); //the first round SB,
- *ShiftRows*(); //the first round SR,
- *MixColumns*(); //the first round MC,
- *AddRoundKey*(); //the first round key addition.

During the above round-reduced algorithm, an attacker inputs two plaintexts (M_1 and M_2) respectively, then he can obtain two faulty ciphertexts ($C_1^{'}$ and $C_2^{'}$). In order to extract the initial round key RK^0, an attacker checks the following equation (1) per byte.

$$SB(M_1 \oplus RK^0) \oplus SB(M_2 \oplus RK^0) = MC^{-1}(C_1^{'} \oplus C_2^{'}) \qquad (1)$$

Here, the *SR* function is not taken into account because it is a byte-wise permutation.

Recently, Park *et al.* proposed a practical DFA on AES using similar round reduction method by fault [14]. In their fault attack, since an attacker injects a fault on the target device using the pseudo code as shown in Fig. 1, the simplest case of reducing the number of rounds is two rounds: the first and last round.

- *AddRoundKey*(); //the initial round key addition,
- *SubBytes*(); //the first round SB,
- *ShiftRows*(); //the first round SR,
- *MixColumns*(); //the first round MC,
- *AddRoundKey*(); //the first round key addition.
- *SubBytes*(); //the last round SB,
- *ShiftRows*(); //the last round SR,
- *AddRoundKey*(); //the last round key addition.

Thus, they developed a new cryptanalysis method of the two-round AES with 10 pairs of plaintext and faulty ciphertexts. As a result, their fault attack could extract the last round key with 2^{40} exhaustive searches in less than 10 hours.

3. The Proposed DFA on Round-reduced AES

In this section we define the strategy to perform a fault analysis. We assume that an attacker can input plaintext and get the corresponding ciphertext during the AES encryption process. Furthermore, he can also reduce the number of rounds by injecting a fault into the target device. The validation of our fault assumption is given in next section with experimental results.

3.1. Attack method for AES recommended in FIP 197 (Type-1)

As shown in Fig. 2, the important feature of the pseudo code is that the last round of AES was not included in the 'for' loop procedure. In our attack method, we assume that an attacker can corrupt the counter index of the 'for' instruction by a fault injection. An attacker has induced a fault on the target device which was implemented the pseudo code as Fig. 1. After waiting for performing up to 8-th round, we inject a fault in the counter index of the 'for' instruction. Then the remained 'for' loop procedure for encryption is skipped and the last round is performed. That is, only the 9-th round is omitted by a round reduction fault.

Using a fault assumption described above, we can get the ciphertext *C* and faulty ciphertext *C'* according to plaintext *M*. The *C* and *C'* is formed as (2) and (3). Here, the 8-th round output state that is, the 9-th round input is M^8 shown as Fig. 3.

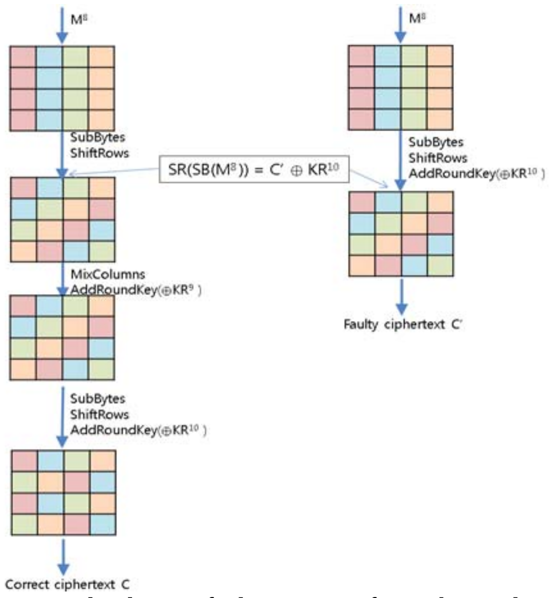


Figure 3. Round reduction fault injection after 8-th round operation.

$$C = SR(SB(MC(SR(SB(M^8))) \oplus RK^9)) \oplus RK^{10}, \tag{2}$$

$$C' = SR(SB(M^8)) \oplus RK^{10}, \tag{3}$$

where RK^9 is the 9-th round key. From (3), we can observe a simple transformation as follows

$$SR(SB(M^8)) = C' \oplus RK^{10}. \tag{4}$$

Thus, we can be rewritten the form of *C* in (2) as

$$C = SR(SB(MC(C' \oplus RK^{10}) \oplus RK^9)) \oplus RK^{10}. \tag{5}$$

In this case, we can guess the round keys from (5), but exhaustive search for key candidates contains two unknown variables RK^9 and RK^{10}. In order to reduce only one unknown variable, it is necessary to eliminate the RK^9 from (5). For this purpose, we input other plaintext and obtain corresponding faulty output *D'* by the same round reduction fault process. Therefore, *D* and *D'* can be written as follows:

$$D = SR(SB(MC(D' \oplus RK^{10}) \oplus RK^9)) \oplus RK^{10}. \tag{6}$$

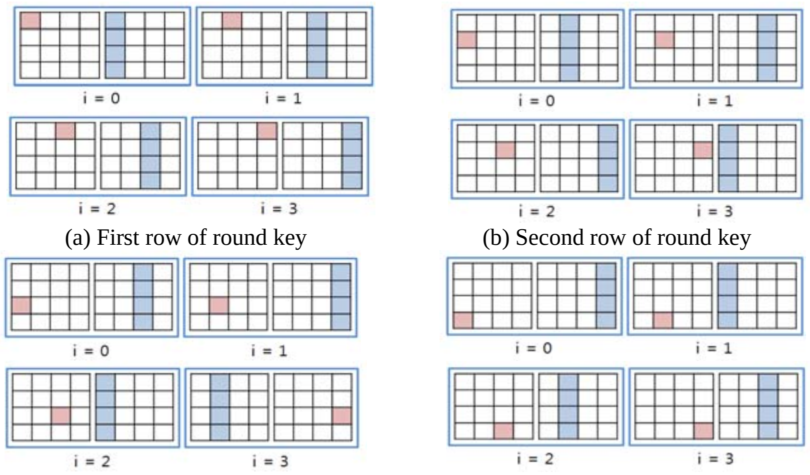
The difference between (5) and (6) still contain the RK^9, since RK^9 is performed XORing with $MC(C'\oplus RK^{10})$ and then it should be operated by Subbytes function. For cancelling the effect of RK^9, we recover the 9-th round output (last round input) from C and D by reversing calculation as in (7) and (8). Therefore, the difference between (7) and (8) can eliminate not only RK^9 but one of RK^{10} which is XORing with faulty ciphertexts based on the linearity of the *MixColumns* operation. We obtain the XORing result of (7) and (8) as (9).

$$\begin{aligned} & SR^{-1}(SB^{-1}(C \oplus RK_{10})) = MC(C'\oplus RK_{10}) \oplus RK_9 \\ & \qquad \Leftrightarrow (SR^{-1}(SB^{-1}(C \oplus RK_{10}))) \oplus (MC(C'\oplus RK_{10}) \oplus RK_9) = 0 \end{aligned} \tag{7}$$

$$\begin{aligned} & SR^{-1}(SB^{-1}(D \oplus RK_{10})) = MC(D'\oplus RK_{10}) \oplus RK_9 \\ & \qquad \Leftrightarrow (SR^{-1}(SB^{-1}(D \oplus RK_{10}))) \oplus (MC(D'\oplus RK_{10}) \oplus RK_9) = 0 \end{aligned} \tag{8}$$

$$(SR^{-1}(SB^{-1}(C \oplus RK^{10}))) \oplus (MC(C')) \oplus (SR^{-1}(SB^{-1}(D \oplus RK^{10}))) \oplus (MC(D')) = 0 \tag{9}$$

Now, we can find the relation of (9) from two pairs of correct and faulty ciphertexts. Here, *C, C', D* and *D'* are known to an attacker, but only RK^{10} is unknown. We can reduce the key candidates per byte by key exhaustive search of $RK^{10}_{0,0} \sim RK^{10}_{3,3}$ to be satisfied (9). A detailed description of extracting key candidates is given as below.



(a) First row of round key (b) Second row of round key

(c) Third row of round key (d) Fourth row of round key

Figure 4. The states of faulty ciphertext related with one byte round key

- **Step 1.** Analyze process on the first row.

In this step, we try to extract the last round key RK^{10} using two pairs of correct and faulty ciphertexts. Firstly, to find the first row round key $RK^{10}_{0,0} \sim RK^{10}_{0,3}$ from (9), we can ignore the *InvShiftRow* operation. The location of faulty ciphertext affected by $RK^{10}_{0,0} \sim RK^{10}_{0,3}$ is shown as Fig. 4(a). The one byte round key shown in left side of Fig. 4(a) is guessed using computation results of the four bytes in right side by *MixColumns* operation of faulty ciphertexts. Therefore, the key candidates of $RK^{10}_{0,0} \sim RK^{10}_{0,3}$ should be satisfied as:

$$\begin{aligned} & SB^{-1}(C_{0,i} \oplus RK^{10}_{0,i}) \oplus (2(C^{'}_{0,i}) \oplus 3(C^{'}_{1,i}) \oplus (C^{'}_{2,i}) \oplus (C^{'}_{3,i})) \\ & \quad \oplus SB^{-1}(D_{0,i} \oplus RK^{10}_{0,i}) \oplus (2(D^{'}_{0,i}) \oplus 3(D^{'}_{1,i}) \oplus (D^{'}_{2,i}) \oplus (D^{'}_{3,i})) = 0 \end{aligned} \tag{10}$$

Therefore, we can reduce the four byte round keys candidates satisfying (10).

- **Step 2**. Analyze process on the second row.

The second row satisfying (9) needs to perform the *InvShiftRow* process once. The location of faulty ciphertext affected by $RK_{1,0}^{10} \sim RK_{1,3}^{10}$ is shown in Fig. 4(b). The one byte round key shown in left side on Fig. 4(b) is guessed using the four bytes of the right side computed by *MixColumns* operation of faulty ciphertexts. In this case, the key candidates of $RK_{1,0}^{10} \sim RK_{1,3}^{10}$ should be satisfied as:

$$\begin{aligned} &SB^{-1}(C_{1,i} \oplus RK_{1,i}^{10}) \oplus ((C_{0,i+1\bmod 4}^{'}) \oplus 2(C_{1,i+1\bmod 4}^{'}) \oplus 3(C_{2,i+1\bmod 4}^{'}) \oplus (C_{3,i+1\bmod 4}^{'})) \\ &\quad \oplus SB^{-1}(D_{1,i} \oplus RK_{1,i}^{10}) \oplus ((D_{0,i+1\bmod 4}^{'}) \oplus 2(D_{1,i+1\bmod 4}^{'}) \oplus 3(D_{2,i+1\bmod 4}^{'}) \oplus (D_{3,i+1\bmod 4}^{'})) = 0 \end{aligned} \quad (11)$$

- **Step 3**. Analyze process on the third row.

The third row satisfying (9) needs to perform the *InvShiftRow* process twice. The location of faulty ciphertext affected by $RK_{2,0}^{10} \sim RK_{2,3}^{10}$ is shown in Fig. 4(c). The one byte round key shown in left side of Fig. 4(c) is guessed using computation results of the four bytes in right side by *MixColumns* operation of faulty ciphertexts. Thus, the key candidates of $RK_{2,0}^{10} \sim RK_{2,3}^{10}$ should be satisfied as:

$$\begin{aligned} &SB^{-1}(C_{2,i} \oplus RK_{2,i}^{10}) \oplus ((C_{0,i+2\bmod 4}^{'}) \oplus (C_{1,i+2\bmod 4}^{'}) \oplus 2(C_{2,i+2\bmod 4}^{'}) \oplus 3(C_{3,i+2\bmod 4}^{'})) \\ &\quad \oplus SB^{-1}(D_{2,i} \oplus RK_{2,i}^{10}) \oplus ((D_{0,i+2\bmod 4}^{'}) \oplus (D_{1,i+2\bmod 4}^{'}) \oplus 2(D_{2,i+2\bmod 4}^{'}) \oplus 3(D_{3,i+2\bmod 4}^{'})) = 0 \end{aligned} \quad (12)$$

- **Step 4**. Analyze process on the last row.

The fourth row satisfying (9) needs to perform the *InvShiftRow* process 3 times. The location of faulty ciphertext affected by $RK_{3,0}^{10} \sim RK_{3,3}^{10}$ is shown in Fig. 4(d). The one byte round key shown on left side of Fig. 4(d) is guessed using computation results of the four bytes on right side by *MixColumns* operation of faulty ciphertexts. Thus, the key candidates of $RK_{3,0}^{10} \sim RK_{3,3}^{10}$ should be satisfied as:

$$\begin{aligned} &SB^{-1}(C_{3,i} \oplus RK_{3,i}^{10}) \oplus (3(C_{0,i+3\bmod 4}^{'}) \oplus (C_{1,i+3\bmod 4}^{'}) \oplus (C_{2,i+3\bmod 4}^{'}) \oplus 2(C_{3,i+3\bmod 4}^{'})) \\ &\quad \oplus SB^{-1}(D_{3,i} \oplus RK_{3,i}^{10}) \oplus (3(D_{0,i+3\bmod 4}^{'}) \oplus (D_{1,i+3\bmod 4}^{'}) \oplus (D_{2,i+3\bmod 4}^{'}) \oplus 2(D_{3,i+3\bmod 4}^{'})) = 0 \end{aligned} \quad (13)$$

During the exhaustive search for RK^{10} satisfying above equations, the values of $MC(C_{k,j}') \oplus MC(D_{k,j}')$ for $0 \le k,i, \le 3$ are known. Since the round key $RK_{k,j}^{10}$ is a byte, the values of $SB^{-1}(C \oplus RK_{k,j}^{10}) \oplus SB^{-1}(D \oplus RK_{k,j}^{10})$ have 256 key hypotheses.

There are two cases to satisfy the equivalent using two pairs of correct and faulty ciphertexts. For example, if the case of $MC(C_{k,j}') \oplus MC(D_{k,j}')$ is *0×F9*, then the set of possible values of $SB^{-1}(C \oplus RK_{k,j}^{10})$ and $SB^{-1}(D \oplus RK_{k,j}^{10})$ are $\{(0xEA, 0xDD), (0xDD, 0xEA)\}$. Therefore, we can obtain two key candidates of $RK_{k,j}^{10}$ by checking the equality of above four equations for $0 \le k,i, \le 3$ from two pairs of correct and faulty ciphertexts. As a result, we can extract the last round key composed of 4x4 bytes through at least 2^{16} exhaustive searches. And we can compute the 128-bit master secret key from last round key.

Therefore, if we inject a fault in the iterative operation of AES encryption using fault injection equipment like a laser beam on the decapsulated-surface of target chip, our proposed attack can retrieve the AES 128-bit secret key with about 2^{16} complexity using just two pairs of correct and faulty ciphertexts.

3.2. Attack method for Type-2

In the Fig. 2 the pseudo code for AES describes that 'for' loop procedure includes entire round functions. Similar with the fault assumption model described in attack for type-1 implementation, an attacker injects a fault in the counter index of 'for' instruction after the execution of the 9-th round. Then the last round is skipped and the algorithm returns the faulty output by a round reduction fault. That is, the output is equivalent to the result of 9-th round before executing the last round.

Using a fault assumption described above, we can get one pair of correct and faulty ciphertext. The relationship between correct and faulty ciphertext is expressed as follow

$$C = (SR(SB(M^9))) \oplus RK^{10} = (SR(SB(C'))) \oplus RK^{10} \qquad (14)$$

where RK^{10} is the last round key. From (14), we can extract RK^{10} with simple substitution as follows

$$RK^{10} = (SR(SB(C'))) \oplus C \qquad (15)$$

Therefore, the attack method against type-2 implementation can deduce the last round key using only one pair of correct and faulty ciphertexts.

4. Result of Experiment and Computer simulation

4.1. Round reduction technique

The method of a round reduction fault assumption is demonstrated by a simple example of the 'for' loop procedure on the target chip ATmega128 microcontroller. Fig. 5 shows the assembly code of the programmed example.

Step 1	**I/O = 5V**	
	354: e2 e3 ldi r30, 0x32	// 1 clock
	356: f0 e0 ldi r31, 0x00	// 1 clock
	358: 8f ef ldi r24, 0xFF	// 1 clock
	35a: 80 83 st Z, r24	// 2 clock
Step 2	**for (i=0;i<100;i++)**	
	35c: 18 8a std Y+25, r1	// 2 clock
	35e: 06 c0 rjmp .+12 ; 0x36c	// 2 clock
Step 3	**a = a+1;**	
	360: 8f 85 ldd r24, Y+27	// 2 clock
	362: 8f 5f subi r24, 0xFF	// 1 clock
	364: 8f 87 std Y+27, r24	// 2 clock
Step 4.1	**for (i=0;i<100;i++)**	
	366: 88 89 ldd r24, Y+25	// 2 clock
	368: 8f 5f subi r24, 0xFF	// 1 clock
	36a: 88 8b std Y+16,r24	// 2 clock
Step 4.2	36c: 88 89 ldd r24, Y+25	// 2 clock
	36e: 84 36 cpi r24, 0x64	// 1 clock
	370: b8 f3 brcs .-18; 0x360	// 1 or 2 clock
Step 5	**I/O = 0V**	
	372: e2 e3 ldi r30, 0x32	// 1 clock
	374: f0 e0 ldi r31, 0x00	// 1 clock
	376: 10 82 st Z, r1	// 2 clock

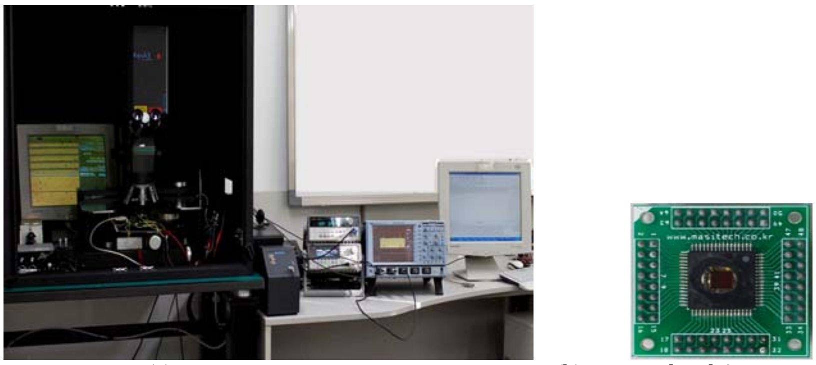
Figure 5. Assembly code of a simple 'for' loop procedure

Step 1 and step 5 change the I/O state to define the execution period of the 'for' loop. Step 2 initialize the counter value *i* to 0; subsequently, the program address jumps to step 4.2 (*0x36c*). In step 4.2, the counter value *i* is compared with the loop bound 100, and then the program address jumps to step 3 if the counter is lower than the loop bound. Step 3 executes addition instruction, which is computed iteratively in the 'for' loop procedure. In the case of AES, step 3 represents the round functions such as *SB*, *SR*, *MC*, *AddRoundkey*. After performing the iterative computation, the counter *i* is increased by 1 in step 4.1.

If a fault is injected at step 4.2, then all remaining iterations are skipped. That is, the corruption of the 'cpi' instruction in step 4.2 leads to an incorrect decision of the comparison instruction determining whether or not to jump back to step 3. To reduce the number of iterative instructions, we try to inject a fault at step 4.2 during the execution of the target number of iterative procedures. To emanate a laser beam at the target step, we count the number of clock cycles, e.g., the 'ldi' instruction consumes 1 clock and the 'st' instruction consumes 2 clocks as described in [15]. Therefore, our fault assumption is practical and easy to implement.

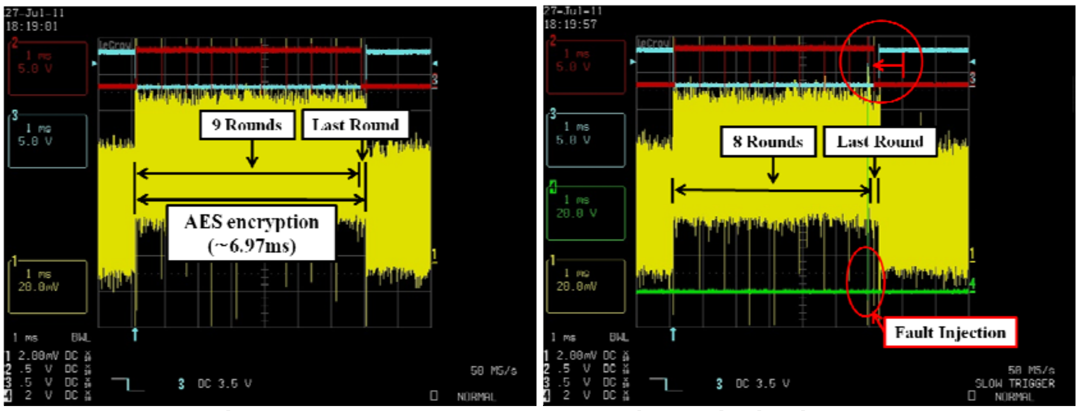
4.2. Fault Injection Experiment

To verify above round-reduction techniques, we implemented two types of AES software on the ATmega128 microcontroller. Our implementation follows the pseudo codes are shown as Fig.1 and Fig.2. Then we omitted the last round operation by injection of a fault at the 'for' loop instruction in the type 1 case, and the 9-th round operation in the type-2 case. We use a fault injection tool EZ Laze 3, which can target a laser beam on the surface of the decapsulated chip. Fig. 6 shows our experimental setup [14].



(a) Experiment setup (b) Decapsulated ATmega128

Figure 6. Experiment setup and target chip



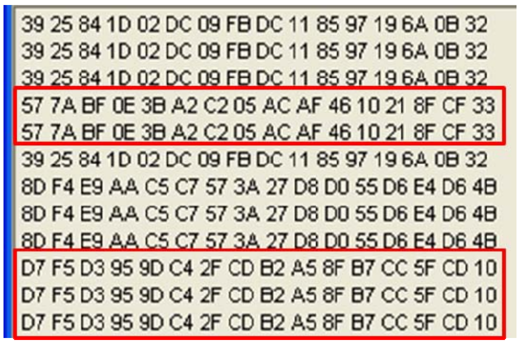
(a) 10 round AES encryption (b) Round-reduced AES encryption

Figure 7. Captured power traces of AES encryption (Type-1)

4.2.1. Experimental result of Type-1 implementation

After several trial and error runs, we were able to obtain two faulty ciphertexts from the round-reduced AES. Fig. 7 shows the captured power signals from target chip. The upper circled area of Fig. 7(b) indicates the end of 8-th round AES and lower one indicates the fault injection trigger. The second I/O signal shows the last round operation signal which is followed after the 8-th round operation. We were able to obtain the faulty ciphertext of the round-reduced AES as Fig. 8(a). As shown in Fig. 8(a), "39

25 … 0B 32” is the correct ciphertexts with input “32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34” and the master key “2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C”, which are the given example in hexadecimal form in FIPS 197. And we use another input “00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF” with same master key to obtain another correct cipher text “8D F4 … D6 4B”. However, “57 7A … CF 33” and “D7 F5 … CD 10” are the faulty ciphertexts of round-reduced AES omitted 9-th round using each input.


```
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
57 7A BF 0E 3B A2 C2 05 AC AF 46 10 21 8F CF 33
57 7A BF 0E 3B A2 C2 05 AC AF 46 10 21 8F CF 33
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
8D F4 E9 AA C5 C7 57 3A 27 D8 D0 55 D6 E4 D6 4B
8D F4 E9 AA C5 C7 57 3A 27 D8 D0 55 D6 E4 D6 4B
8D F4 E9 AA C5 C7 57 3A 27 D8 D0 55 D6 E4 D6 4B
D7 F5 D3 95 9D C4 2F CD B2 A5 8F B7 CC 5F CD 10
D7 F5 D3 95 9D C4 2F CD B2 A5 8F B7 CC 5F CD 10
D7 F5 D3 95 9D C4 2F CD B2 A5 8F B7 CC 5F CD 10
```
(a) Type-1 implementation

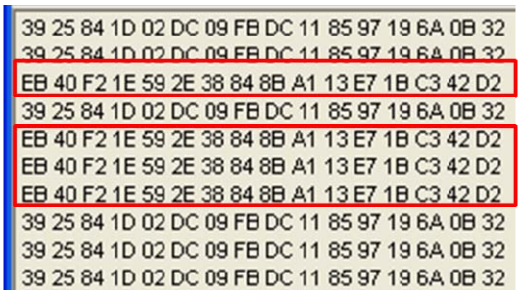
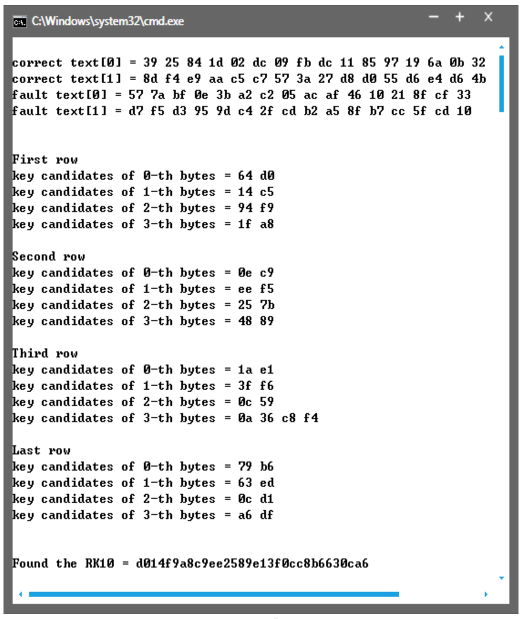
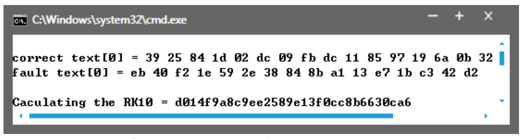

```
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
EB 40 F2 1E 59 2E 38 84 8B A1 13 E7 1B C3 42 D2
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
EB 40 F2 1E 59 2E 38 84 8B A1 13 E7 1B C3 42 D2
EB 40 F2 1E 59 2E 38 84 8B A1 13 E7 1B C3 42 D2
EB 40 F2 1E 59 2E 38 84 8B A1 13 E7 1B C3 42 D2
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32
```
(b) Type-2 implementation

Figure 8. Ciphertexts from target chip



```
correct text[0] = 39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32
correct text[1] = 8d f4 e9 aa c5 c7 57 3a 27 d8 d0 55 d6 e4 d6 4b
fault text[0] = 57 7a bf 0e 3b a2 c2 05 ac af 46 10 21 8f cf 33
fault text[1] = d7 f5 d3 95 9d c4 2f cd b2 a5 8f b7 cc 5f cd 10


First row
key candidates of 0-th bytes = 64 d0
key candidates of 1-th bytes = 14 c5
key candidates of 2-th bytes = 94 f9
key candidates of 3-th bytes = 1f a8

Second row
key candidates of 0-th bytes = 0e c9
key candidates of 1-th bytes = ee f5
key candidates of 2-th bytes = 25 7b
key candidates of 3-th bytes = 48 89

Third row
key candidates of 0-th bytes = 1a e1
key candidates of 1-th bytes = 3f f6
key candidates of 2-th bytes = 0c 59
key candidates of 3-th bytes = 0a 36 c8 f4

Last row
key candidates of 0-th bytes = 79 b6
key candidates of 1-th bytes = 63 ed
key candidates of 2-th bytes = 0c d1
key candidates of 3-th bytes = a6 df


Found the RK10 = d014f9a8c9ee2589e13f0cc8b6630ca6
```
(a) Type-1 implementation



```
correct text[0] = 39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32
fault text[0] = eb 40 f2 1e 59 2e 38 84 8b a1 13 e7 1b c3 42 d2

Caculating the RK10 = d014f9a8c9ee2589e13f0cc8b6630ca6
```
(b) Type-2 implementation

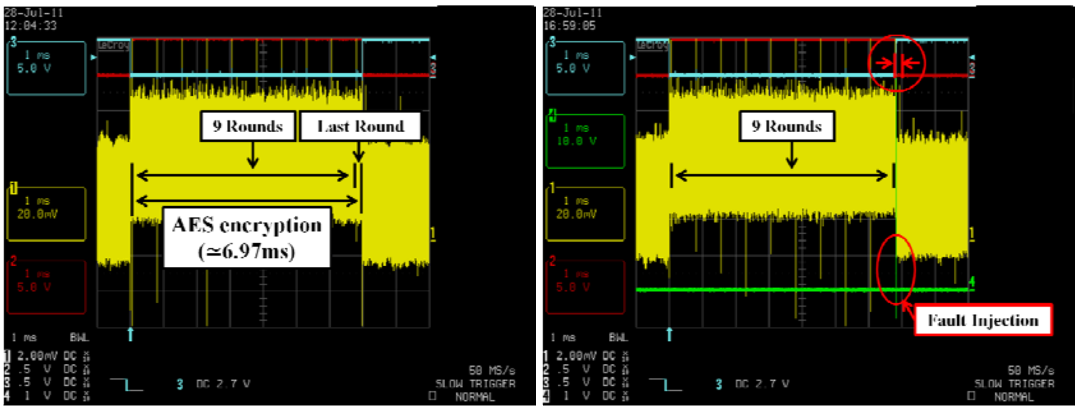
Figure 9. Key search simulation result

Firstly, to deduce the last round key using faulty ciphertexts, we perform exhaustive key search on a PC with a 3.0 GHz CPU with 4GB memory using Visual C++ software. Fig. 9(a) shows the search result of 10-th round key described steps in the previous section. As shown in Fig. 9(a), we successfully deduced the key candidates of each step and were able to recover the last round key by about 2^{16} key exhaustive searches.

4.2.2. Experimental result of Type-2 implementation

In the attack of type-2 implementation, we needed only one-pair of correct-faulty ciphertexts. Similar with the above result of type-1 implementation, the power signal is shown as Fig. 10(b) indicates the last round is omitted. As shown in Fig. 8(b), we also were able to obtain the faulty ciphertext of the round-reduced AES using the given input and the master key in FIPS 197.

We also calculated the last round key from computing the equation (15) as shown in Fig 9(b). Using our attack method, an attacker can easily recover the master key using only one pair of correct- faulty cipher text.



(a) 10 round AES encryption (b) Round-reduced AES encryption

Figure 10. Captured power traces of AES encryption (Type-2)

5. Conclusion

This paper proposed a new round-reduction fault attack based DFA on AES. To extract the 10-th round key of AES, we found candidates of possible key bytes using two pairs of plaintext and faulty ciphertexts obtained from the round-reduced AES and then performed the exhaustive search with 2^{16} complexity. To verify the proposed DFA method, we implemented the AES algorithm recommended in FIPS 197 on the ATmega128 microcontroller. To obtain faulty ciphertexts with omitting the 9-th round operation, we injected a laser beam during the execution of AES. After obtaining just two ciphertexts from the reduced-round AES, we successfully extracted 128-bit secret key. To prevent round reduction fault attack described in this literature, chip developer should check the number of round performed for AES encryption.

6. Acknowledgment

This work was supported by SCARF project which is the R&D program of KCC/KCA [Development of the Technology of Side Channel Attack Countermeasure Primitives and Security Validation]

6. References

[1] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton, “On the Importance of Checking Cryptographic Protocols for Faults”, In Proceeding of EUROCRYPT 1997, LNCS 1233, pp. 37–51, 1997.
[2] Paul Kocher, Joshua Jaffe, and Benjamin Jun, “Differential Power Analysis”, In Proceeding of Advances in Cryptology - CRYPTO '99 , LNCS 1666, pp. 388-397, 1999.
[3] Liu Shubo, Tang ming, Gao Si, and Zhang Huanguo, "Evolution Cipher against Differential Power Attack", JDCTA, Vol. 4, No. 7, pp. 177 ~ 189, 2010
[4] Eli Biham and Adi Shamir, “Differential fault analysis of secret key cryptosystems”, In Proceeding of Advances in Cryptology - CRYPTO '97, LNCS 1294, pp.513-525, 1997.
[5] Chong Hee Kim and Jean-Jacques Quisquater, “Faults, injection methods, and fault attacks”, IEEE Design&Test of Computers, IEEE, vol. 24, no.6, pp. 544-545, 2007.

[6] Wei Li, Dawu Gu, and Juanru Li, “Differential fault analysis on the ARIA algorithm”, Information Sciences, Elsevier Inc., pp.3727–3737, 2008.
[7] Chunhui Sun, Hui Li, Yang Yang, and Jie Chen, "Research on Fault-Electromagnetic Attack on Block Cipher", JCIT, Vol. 6, No. 11, pp. 409 ~ 417, 2011
[8] Johannes Blömer and Jean-Pierre Seifert, “Fault based cryptanalysis of the advanced encryption standard (AES)”, In Proceeding of FC 2003, LNCS 2742, pp.162-181, 2003.
[9] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo, “Differential fault analysis on AES”, In Proceeding of ACNS'03, LNCS 2846, pp.293-306, 2003.
[10] M. Tunstall, “Practical complexity differential cryptanalysis and fault analysis of AES,” Journal of Cryptographic Engineering, Springer, pp.219-230, 2011.
[11] Wei Li, Xiaoling Xia, Dawu Gu, Zhiqiang Liu, Juanru Li, and Ya Liu, “A New Differential Fault Attack on SPN Structure, with Application to AES Cipher”, Journal of Computers, Academy Publicher, vol. 6, no. 2, pp. 216-223, 2011
[12] Hamid Choukri and Michael Tunstall, “Round reduction using faults”, In Proceeding of FDTC'05, 2005, pp. 13-24, 2005.
[13] NIST, “Announcing the Advanced Encryption Standard,” FIPS 197, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.
[14] JeaHoon Park, SangJae Moon, DooHo Choi, YouSung Kang, and JaeCheol Ha, "Differential fault analysis for round-reduced AES by fault injection”, ETRI Journal, vol. 33, no. 3, pp. 434-442, 2011.
[15] ATMEL Corp. webpage, http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf