Fault Detection of the MacGuffin Cipher against Differential Fault Attack

Wei Li[1,2,⋆], Dawu Gu[3], Zhiqiang Liu[3], Ya Liu[3], and Xiaohu Huang[1]

[1] School of Computer Science and Technology, Donghua University, Shanghai 201620, China

[2] Shanghai Key Laboratory of Integrate Administration Technologies for Information Security, Shanghai 200240, China

[3] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract. Since the early work of Biham and Shamir on differential fault attack against block ciphers at CRYPTO 1997, much work has been devoted to reducing the number of faults and to improving the time complexity of this attack. This attack is very efficient when a single fault is injected on the last several rounds, and it allows to recover the whole secret key. Thus, it is an open question whether detecting the faults injected into a block cipher against this attack with low overhead of space and time tolerance. The MacGuffin cipher, a representative of the Unbalanced Feistel Network(UFN) structure, is vulnerable to fault attack at the last four rounds. In this paper, we give an answer to this problem by presenting a fault detection of the MacGuffin block cipher. Our result in this study could detect the faults with negligible cost when faults are injected into the last four rounds.

1 Introduction

During the last years a new class of attacks against cryptographic devices has become public. These attacks exploit easily accessible information like power consumption, running time, input–output behavior under malfunctions, and can be mounted by anyone only using low–cost equipment. These side–channel attacks amplify and evaluate leaked information with the help of statistical methods, and are often much more powerful than classical cryptanalysis. Examples show that a very small amount of side–channel information is enough to completely break a cryptosystem. While many previously–known cryptanalytic attacks can be analyzed by studying algorithms, the vulnerabilities of side–channel attacks result from electrical behavior of transistors and circuits of an implementation. This ultimately compromises cryptography, and shifts the top priority in cryptography from the further improvement of algorithms to the prevention of such attacks by reducing variations in timing, power and radiation from the hardware, reduction of observability of system behavior after fault injection. Therefore, it

⋆ Corresponding author.

extends theoretically the current mathematical models of cryptography to the physical setting which takes into consideration side–channel attacks.

As one type of side–channel attacks, differential fault analysis (DFA) was proposed by Biham and Shamir as an attack on DES in 1997 [1]. The similar attacks have been applied to other block ciphers [2–6]. The DFA attack is based on deriving information about the secret key by examining the differences between a cipher resulting from a correct operation and a cipher of the same initial message resulting from a faulty operation.

MacGuffin is a block cipher which was proposed by Blaze and Schneier [7]. Its fundamental structure is the contracting Unbalanced Feistel Network, and supports 64–bit block size and 128–bit key size. Up to now, some literature is available on the security of MacGuffin against the classical cryptanalysis, such as differential attack, and linear attack [8]. MacGuffin is vulnerable to Differential Fault Analysis(DFA)[9]. The secret key of MacGuffin could be obtained by inducing faults into the computation of the last four rounds. This method requires 355 and 165 faulty ciphertexts in two byte–oriented fault models, respectively.

In this paper, we focus on the security application of MacGuffin against the fault analysis. In the literature, countermeasures against fault attacks could help a cryptographic algorithm to avoid, detect or correct faults. In practice, many proposed schemes are based on fault detection, including code–based technique and redundancy–based technique [10–18].

Code based detections are divided into coding method and error detection code (EDC). Coding method means encoding message before encryption and checking errors after decryption. Its overhead depends on encoding and decoding progress to translate plaintexts and ciphertexts into codes. Its time redundancy also depends on the code processes. As for block ciphers, the EDC approach is often used in each rounds' inner parts with the implementation of parity–based EDC. The parity of linear layers is easy to implement since permutations do not change the parity. More consideration should be given to the nonlinear layers. Whether the parity of input joins in encryption determines how the parity constructs. Approximately, 10%$\sim$20% overhead is required, and so does time tolerance.

The redundancy–based solution for implementing fault detection in the encryption module is to perform a test decryption immediately after the encryption, and then check whether the original data block is obtained. If a decryption module is already present in the implementation, the hardware overhead reduces to the cost of a comparator for two data blocks of 128 bits. Otherwise, the overhead is close to 100 percent since the decryption module is very similar to the encryption one. The overall time penalty in either of these two cases is the time required to decrypt a data block, plus the time required for the comparison. This technique is independent of the adopted fault model.

The above techniques of fault detection seem to ensure a high level of security. However, only checking the correctness of the computed encryption result may not be enough to prevent fault analysis since an attacker may destroy the detector.

In order to resist the differential fault analysis with low cost, we propose a fault detection technique to protect MacGuffin against the previous attacks. Our work not only helps to detect the errors with low overhead of space and time tolerance, but also can be applied in all kinds of software implementation. The idea of this attack and the related countermeasure are naturally suitable for other block ciphers.

The rest of this paper is organized as follows. Section 2 briefly introduces the MacGuffin cryptosystem. The next section shows the previous differential fault analysis on MacGuffin. Then section 4 presents our fault detection on MacGuffin. Finally section 5 concludes the paper.

2 Description of MacGuffin

MacGuffin is a 64–bit block cipher, which supports 128–bit key lengths [7]. It has 32–round unbalanced Feistel structure. The input of MacGuffin is partitioned into four registers from left to right (See Figure 1). Every register is composed of double bytes. In every round, the three rightmost registers comprise the control block and are bitwise exclusive–ORed with 48 bits derived from the subkey. These 48 bits are split eight branches to provide input to eight functions of six bits (the S–boxes), and then output two bits for every S–box. The 16–bit S–boxes output are then XORed with the bits in the leftmost register. Finally, the leftmost register is rotated into the rightmost register. Figure 1 shows the block diagram of the MacGuffin cipher.

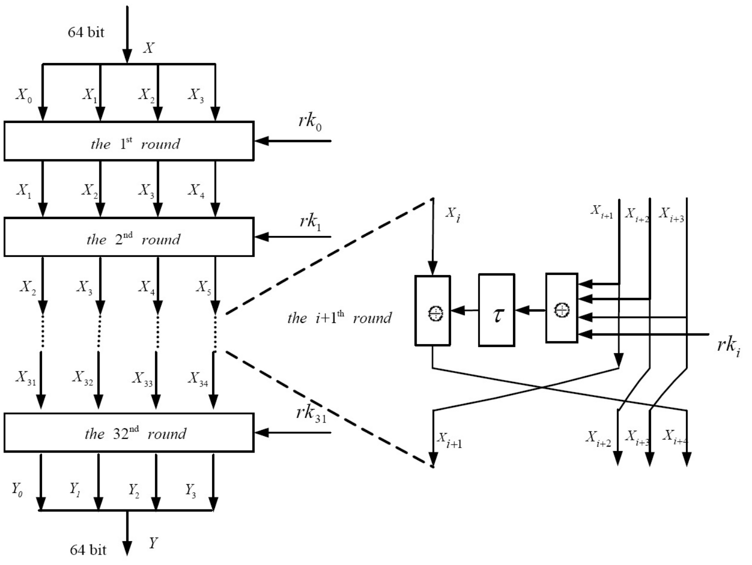

Fig. 1. The MacGuffin cipher

2.1 Encryption Process

Let $X = (X_0, X_1, X_2, X_3) \in (\{0,1\}^{16})^4$ be the plaintext and $Y = (Y_0, Y_1, Y_2, Y_3) \in (\{0,1\}^{16})^4$ be the ciphertext. Let $rk_i \in (\{0,1\}^{16})^3$ denote the i-th subkey, $(X_i, X_{i+1}, X_{i+2}, X_{i+3})$ denote the $i+1$-th round inputs, and R_i denote the $i+1$-th round $(i = 0, 1, \cdots, 31)$. Then the MacGuffin scheme can be written as

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i),$$

$$(Y_0, Y_1, Y_2, Y_3) = (X_{32}, X_{33}, X_{34}, X_{35}),$$

where $i \in \{0, 1, \cdots, 31\}$, F is the i–th round function defined below:

$$F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) = X_i \oplus \tau(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i).$$

Here τ are defined as follows.

τ–function is a nonlinear transformation layer with 8 parallel 6×2 S-boxes, which are specified in [7]. That is,

$$\tau : (\{0,1\}^6)^8 \rightarrow (\{0,1\}^2)^8.$$

2.2 Decryption Process

The decryption procedure of MacGuffin can be done in the same way as the encryption procedure by reversing the order of the subkeys.

2.3 Key Schedule

In the MacGuffin cryptosystem, the key schedule generates a total of 32 subkeys $(rk_0, rk_1, \cdots, rk_{31})$. Each round of the cipher uses the secret key parameter to perturb the S–boxes by bitwise XOR against the S–box inputs. Each round thus requires 48 key bits. To covert the 128–bit secret key to a sequence of 48–bit values for each round, MacGuffin uses an iterated version of its own block encryption function. In our fault detection, all errors are injected in the encryption procedure. Thus, we could omit the structure of the key schedule.

3 The Previous Differential Fault Analysis on MacGuffin

The previous differential fault analysis on the security of MacGuffin adopts two basic assumptions as follows:

(1) The attacker can induce a single byte error to a 16–bit register. However, the location of this byte in this register and the value of the error are both unknown.
(2) The attacker has the capability to obtain the right and the corresponding faulty ciphertexts when encrypting one plaintext with the same secret key.

On the above basic assumptions, they induce a random error in the last four rounds at the beginning of the attack, and thus obtain a faulty ciphertext. By differential fault analysis, part or all bytes of the subkeys in the last round can be recovered. The location of fault injection may be not the location of subkeys which will be recovered. For example, to recover the subkeys in the last round, they induce errors in the penultimate round. This kind of fault injection could derive multiple bytes of one subkey and avoids decreasing the efficiency of fault injection. Repeat this procedure until the subkey is obtained. Then they decrypt the right ciphertext to obtain the input of the last round, which is the output of the penultimate round. Repeat the above procedure until the secret key is obtained by the key schedule.

4 Our Proposed Fault Detection of MacGuffin

Our objective is to develop fault detection techniques which will be independent of the particular hardware implementation. To this end, we make the following assumptions:

(1) The MacGuffin algorithm is partitioned into three basic modules: encryption, decryption, and key schedule.
(2) All the modules have in common the same basic operations; hence, only the encryption module is examined in detail since most conclusions will hold for the remaining modules as well.

Thus, a fault injected into the first round is comparable to encoding a different input. The injection of a fault in one of the inner rounds is more complicated and it is necessary to follow the errors as they propagate along the execution path.

Every round of MacGuffin consists of the round function, which is composed of two transformations: subkey addition(SA), S–boxes. Different from the other block ciphers, MacGuffin has no linear transformation. The propagation of a single fault is influenced by the execution of the round components. The result can be classified into only one cases: the fault affects only one byte in the output. The situation includes the S–boxes and SA transformations, where the error is only moved within a byte, respectively. When using a specific input and injecting lots of a single–byte fault into every different round, the average number of erroneous bytes in the ciphertext has the following characteristic (See Fig. 2):

(1) If there are less than 8 nonzero erroneous bytes, the fault must occur in the last four rounds. The average number of erroneous bytes is 7.75, 6, 4.25 and 3.25, respectively.
(2) If all bytes are erroneous, most faults may occur before the last four rounds.

To date, little research has been done on the related attacking method when the faults are induced before the four rounds. Thus, MacGuffin is secure even if the errors have been induced before the four rounds. We put emphasis on the

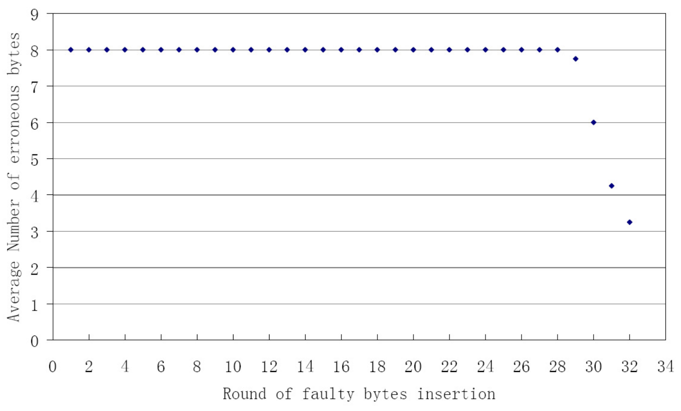

Fig. 2. Erroneous bytes in the ciphertext of MacGuffin

research of the errors induced into the last four rounds. In the DFA analysis, the attacker must capture at least two ciphertexts, including one right ciphertext and one faulty ciphertext. On the basis of this assumption, we propose a pattern–based technique to infer whether the attacker induce faults into the encryption module.

For MacGuffin, let Y, Y^*, $\triangle Y$ be the correct ciphertext, the faulty ciphertext, and ciphertext difference. Let $|\triangle Y|$, $|\triangle Y_0|$, $|\triangle Y_1|$, $|\triangle Y_2|$ and $|\triangle Y_3|$ be the number of erroneous bytes in $\triangle Y$, $\triangle Y_0$, $\triangle Y_1$, $\triangle Y_2$ and $\triangle Y_3$. The pattern is defined within the bounds of remote possibility as the result of the XOR operation between two right ciphertexts (See Table 1). If the distribution of a ciphertext difference satisfies these patterns, then we could derive that the attacker has induced faults into the encryption module and at least one ciphertext is faulty. In other words, if the ciphertext difference satisfies the distribution of some patterns in Table 1, it shows that the error has been induced into the encryption module. Otherwise, it is not feasible for DFA to derive the secret key of MacGuffin. In Table 1, the pattern 0001 denotes $\triangle Y$ has one nonzero byte, which locates in the register Y_3. The ciphertext pair with this pattern could be one correct ciphertext and one faulty ciphertext, or two faulty ciphertext, since the two correct ciphertexts with the pattern has the remote probability of 1.31E–34%.

Depending on the pattern of the ciphertext difference between one correct ciphertext and one faulty ciphertext, we could detect the fault location as Table 2 shows. For example, if the pattern is 0001, the fault must be injected in the register X_{31} of the 32nd round.

When some patterns are within the bounds of average possibility as the result of the XOR operation of one correct ciphertext and one faulty ciphertext. For example, when an error is injected into the last four round, the ratio of 1, 2, 3, 4, 5, 6, 7, and 8 erroneous bytes occur at 6.25%, 0%, 25&, 6.25%, 18.75%, 12.5%, 12.5%, and 18.75%, respectively(See Figure 3).

Table 1. The relationship between some patterns and ciphertext pairs

Pattern		Percentage(%)	A ciphertext pair
$\|\triangle Y\|$	$\|\triangle Y_0\|, \|\triangle Y_1\|, \|\triangle Y_2\|, \|\triangle Y_3\|$		
1	0001	1.31E–34	(Y, Y^*), (Y^*, Y^*)
2	0002	1.23E–36	(Y^*, Y^*)
3	1002, 0102, 0012	1.91E–25	(Y, Y^*), (Y^*, Y^*)
4	0022, 0202, 2002, 1102, 1012, 0112	1.78E–23	(Y, Y^*), (Y^*, Y^*)
5	1022, 0122	3.78E–24	(Y, Y^*), (Y^*, Y^*)
6	0222, 1122, 2022	4.15E–22	(Y, Y^*), (Y^*, Y^*)
7	1222	5.17E–22	(Y, Y^*), (Y^*, Y^*)
8	2222	99%	

Table 2. The relationship between the pattern and fault locations

Pattern		Location of one-byte fault injection			
$\|\triangle Y\|$	$\|\triangle Y_0\|, \|\triangle Y_1\|, \|\triangle Y_2\|, \|\triangle Y_3\|$	R_{32}	R_{31}	R_{30}	R_{29}
1	0001	X_{31}	/	/	/
3	1002	X_{32}	/	/	/
	0102	X_{33}	/	/	/
	0012	X_{34}	X_{30}	/	/
4	0022	/	X_{31}	/	/
5	1022	/	X_{32}	/	/
	0122	/	X_{33}	X_{29}	/
6	0222	/	/	X_{30}, X_{31}	/
7	1222	/	/	X_{32}	X_{28}
8	2222	/	/	/	X_{29}, X_{30}, X_{31}

In real application, one correct ciphertext and one faulty ciphertext as a ciphertext pair is ideal. However, there exist two faulty ciphertexts as Table 1 shows. On the basis of Table 2, we build up the pattern of ciphertext difference between two faulty ciphertexts(See Table 3). Thus, we derive the relationship between the pattern of two faulty ciphertext and the fault locations in Table 4.

When two error are injected independently into the last four round, the ratio of 1, 2, 3, 4, 5, 6, 7, and 8 erroneous bytes occurs at 0.20%, 0.20%, 5.08%, 6.84%, 16.21%, 19.53%, and 26.95%, respectively(See Figure 4).

If the distribution of a ciphertext difference satisfies these above patterns, then the attacker has induced faults into the encryption module and at least one ciphertext is faulty. It is helpful for the MacGuffin cipher to be secure against the differential fault analysis. In other words, if the ciphertext difference satisfies the distribution of some patterns in Table 1, it shows that the error has been induced into the encryption module. Otherwise, it is not feasible for DFA to derive the secret key of MacGuffin.

For example, if the ciphertext difference has 2 nonzero bytes, there are only one pattern which is 0002. It shows that the attacker injects two faults, whose

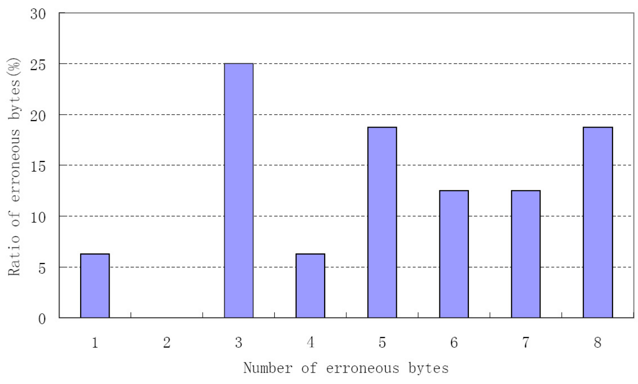

Fig. 3. Ratio of Erroneous bytes in one correct ciphertext and one faulty ciphertext in the last four rounds

Table 3. Patterns of two faulty ciphertexts

	0001	1002	0102	0012	0022	1022	0122	0222	1222	2222
0001	0001,0002	1002	0102	0012	0022	1022	0122	0222	1222	2222
1002	1002	2002	1102	1012	1022	1022,2022	1122	1222	1222,2222	2222
0102	0102	1102	0102,0202	0112	0122	1122	1122	0222	1222	2222
0012	0012	1012	0112	0012,0022	0022	1022	0122	0222	1222	2222
0022	0022	1022	0122	0022	0022	1022	0122	0222	1222	2222
1022	1022	1022,2022	1122	1022	1022	1022,2022	1122	1222	1222,2222	2222
0122	0122	1122	1122	0122	0122	1122	0122,0222	0122	1222	2222
0222	0222	1222	0222	0222	0222	1222	0122	0222	1222	2222
1222	1222	1222,2222	1222	1222	1222	1222,2222	1222	1222	1222,2222	2222
2222	2222	2222	2222	2222	2222	2222	2222	2222	2222	2222

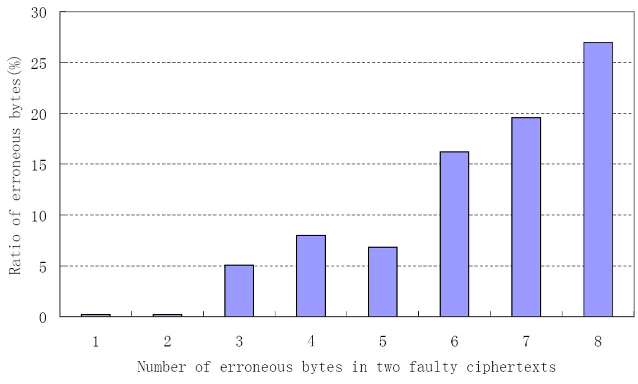

Fig. 4. Ratio of Erroneous bytes of two faulty ciphertexts in the last four rounds

Table 4. The relationship between pattern of two faulty ciphertexts and fault injection

Pattern		Location of fault injections
$\|\triangle Y\|$	$\|\triangle Y_0\|, \|\triangle Y_1\|, \|\triangle Y_2\|, \|\triangle Y_3\|$	
1	0001	$(R_{32}X_{31}, R_{32}X_{31})$
2	0002	$(R_{32}X_{31}, R_{32}X_{31})$
3	1002	$(R_{32}X_{31}, R_{32}X_{32})$
	0102	$(R_{32}X_{31}, R_{32}X_{33})$, $(R_{32}X_{33}, R_{32}X_{33})$
	0012	$(R_{32}X_{31}, R_{32}X_{34})$, $(R_{32}X_{31}, R_{32}X_{30})$, $(R_{32}X_{34}, R_{32}X_{30})$ $(R_{32}X_{34}, R_{32}X_{34})$,$(R_{32}X_{30}, R_{32}X_{30})$
4	0202	$(R_{32}X_{33}, R_{32}X_{33})$
	0022	$(R_{32}X_{31}, R_{31}X_{31})$, $(R_{31}X_{31}, R_{31}X_{31})$,$(R_{32}X_{34}, R_{32}X_{34})$, $(R_{32}X_{34}, R_{31}X_{31})$, $(R_{32}X_{30}, R_{31}X_{31})$,$(R_{32}X_{30}, R_{32}X_{30})$, $(R_{32}X_{34}, R_{32}X_{30})$
	2002	$(R_{32}X_{32}, R_{32}X_{32})$
	1102	$(R_{32}X_{32}, R_{32}X_{33})$
	1012	$(R_{32}X_{32}, R_{32}X_{34})$, $(R_{32}X_{32}, R_{32}X_{30})$
	0112	$(R_{32}X_{33}, R_{32}X_{34})$, $(R_{32}X_{33}, R_{32}X_{30})$
5	1022	$(R_{32}X_{31}, R_{31}X_{32})$, $(R_{32}X_{32}, R_{31}X_{31})$, $(R_{31}X_{31}, R_{32}X_{32})$, $(R_{32}X_{34}, R_{31}X_{31})$, $(R_{32}X_{30}, R_{31}X_{31})$,$(R_{31}X_{32}, R_{31}X_{32})$, $(R_{32}X_{32}, R_{31}X_{32})$
	0122	$(R_{32}X_{31}, R_{31}X_{33})$, $(R_{32}X_{31}, R_{30}X_{29})$, $R_{30}X_{29}, R_{31}X_{31})$, $(R_{32}X_{33}, R_{31}X_{31})$, $(R_{31}X_{33}, R_{31}X_{31})$, $(R_{32}X_{34}, R_{31}X_{32})$, $(R_{32}X_{30}, R_{31}X_{32})$, $(R_{32}X_{34}, R_{30}X_{29})$, $(R_{32}X_{31}, R_{32}X_{31})$
6	0222	$(R_{32}X_{31}, R_{30}X_{30})$,$(R_{32}X_{31}, R_{30}X_{31})$, $(R_{30}X_{29}, R_{30}X_{29})$ $(R_{32}X_{33}, R_{30}X_{30})$, $(R_{32}X_{33}, R_{30}X_{31})$, $(R_{30}X_{31}, R_{32}X_{34})$, $(R_{30}X_{30}, R_{32}X_{34})$, $(R_{30}X_{30}, R_{32}X_{30})$, $(R_{30}X_{31}, R_{32}X_{30})$ $(R_{30}X_{30}, R_{31}X_{31})$, $(R_{30}X_{31}, R_{31}X_{31})$, $(R_{30}X_{30}, R_{30}X_{31})$, $(R_{30}X_{30}, R_{30}X_{30})$, $(R_{30}X_{31}, R_{30}X_{31})$, $(R_{31}X_{33}, R_{30}X_{29})$ $(R_{31}X_{33}, R_{31}X_{33})$
	1122	$(R_{32}X_{33}, R_{31}X_{32})$, $(R_{32}X_{32}, R_{31}X_{33})$, $(R_{32}X_{32}, R_{30}X_{29})$, $(R_{32}X_{32}, R_{30}X_{29})$, $(R_{31}X_{32}, R_{31}X_{33})$, $(R_{32}X_{33}, R_{31}X_{33})$ $(R_{30}X_{29}, R_{31}X_{32})$
	2022	$(R_{32}X_{32}, R_{31}X_{32})$, $(R_{31}X_{32}, R_{31}X_{32})$
7	1222	$(R_{32}X_{31}, R_{30}X_{32})$, $(R_{32}X_{31}, R_{29}X_{28})$, $(R_{32}X_{32}, R_{30}X_{32})$, $(R_{32}X_{32}, R_{30}X_{30})$, $(R_{32}X_{32}, R_{30}X_{31})$,$(R_{32}X_{32}, R_{29}X_{28}$ $(R_{32}X_{33}, R_{30}X_{32})$, $(R_{32}X_{33}, R_{29}X_{28})$, $(R_{32}X_{34}, R_{30}X_{32})$, $(R_{32}X_{31}, R_{30}X_{32})$, $(R_{32}X_{31}, R_{29}X_{28})$,$(R_{32}X_{30}, R_{29}X_{28})$ $(R_{32}X_{30}, R_{30}X_{32})$, $(R_{32}X_{34}, R_{29}X_{28})$, $(R_{31}X_{31}, R_{30}X_{32})$ $(R_{30}X_{30}, R_{31}X_{32})$, $(R_{30}X_{31}, R_{31}X_{32})$, $(R_{31}X_{31}, R_{29}X_{28})$ $(R_{31}X_{32}, R_{30}X_{32})$, $(R_{31}X_{32}, R_{29}X_{28})$, $(R_{30}X_{29}, R_{30}X_{32})$, $(R_{31}X_{33}, R_{30}X_{32})$, $(R_{30}X_{29}, R_{29}X_{28})$,$(R_{31}X_{33}, R_{29}X_{28})$ $(R_{30}X_{30}, R_{30}X_{32})$, $(R_{30}X_{31}, R_{29}X_{28})$, $(R_{30}X_{32}, R_{30}X_{32})$ $(R_{30}X_{31}, R_{30}X_{32})$, $(R_{30}X_{30}, R_{29}X_{28})$, $(R_{29}X_{28}, R_{29}X_{28})$ $(R_{29}X_{28}, R_{30}X_{32})$
8	2222	

locations are both in the register X_{31} of the 31st round. If the ciphertext difference has 3 nonzero bytes and its pattern is 1002, the attacker might inject one or two faults. The locations might be in the register X_{31} of the 32nd round and the register X_{32} of the 32nd round.

We implemented the experiment on a PC using Visual C++ on a 1.60 GHz centrino with 2GB memory. The fault induction was simulated by computer software. In this situation, we ran the attacking algorithm to 1000 encryption unit with different random generated keys. And then we could detect about 77.15% errors into the last four rounds of MacGuffin. Unless the errors' pattern is 2222, we could detect 100% errors.

Compared with the previous techniques, the overhead and time tolerance of required for the comparison in our method is negligible (see Table 5). As one countermeasure of MacGuffin against DFA, the pattern–based technique could not only help to detect the errors with low overhead of space and time tolerance, but also be applied in hardware or software implementation.

Table 5. Comparison of overhead and tolerance

Approaches	Overhead	Time tolerance
Duplication	100%	100%
Coding method	Encoding dependent	Encoding dependent
EDC method	10-20%	Parity dependent
Proposed method	Negligible	Negligible

5 Conclusion

In this study, we present a fault injection of MacGuffin in software implementation. This method adopts the special pattern of ciphertext pairs in the attacking assumption and procedure of differential fault analysis. It is simple to detect errors in real applications and provides a practical approach for fault detection on block ciphers.

Future analysis should be able to detect differential fault analysis when the faults are injected into deeper rounds and the ciphertext difference has no special patterns. For the hardware situation, we will leave it for the future research.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61003278, the Opening Project of Shanghai Key Laboratory of Integrate Administration Technologies for Information Security, and the Fundamental Research Funds for the Central Universities. The authors wish to acknowledge the anonymous referees for helpful suggestions.

References

[1] Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)

[2] Moradi, A., Manzuri Shalmani, M.T., Salmasizadeh, M.: A Generalized Method of Differential Fault Attack Against AES Cryptosystem. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 91–100. Springer, Heidelberg (2006)
[3] Hemme, L.: A Differential Fault Attack Against Early Rounds of (Triple-)DES. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 254–267. Springer, Heidelberg (2004)
[4] Clavier, C., Gierlichs, B., Verbauwhede, I.: Fault Analysis Study of IDEA. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 274–287. Springer, Heidelberg (2008)
[5] Li, W., Gu, D., Li, J.: Differential fault analysis on the ARIA algorithm. Information Sciences 178(19), 3727–3737 (2008)
[6] Li, W., Gu, D., Li, J., Liu, Z., Liu, Y.: Differential fault analysis on Camellia. Journal of Systems and Software 83, 844–851 (2010)
[7] Blaze, M., Schneier, B.: The MacGuffin Block Cipher Algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 97–100. Springer, Heidelberg (1995)
[8] Rijmen, V., Preneel, B.: Cryptanalysis of MacGuffin. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 353–358. Springer, Heidelberg (1995)
[9] Li, W., Gu, D., Wang, Y.: Differential Fault Analysis on the Contracting UFN Structure, with Application to SMS4 and MacGuffin. Journal of Systems and Software 82(2), 346–354 (2009)
[10] Karpovsky, M., Kulikowski, K.J., Taubin, A.: Differential fault analysis attack resistant architectures for the Advanced Encryption Standard. In: International Conference on Smart Card Research and Advanced Applications – CARDIS 2004, pp. 177–192. IEEE Computer Society (2004)
[11] Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes for fault–based side–channel cryptanalysis of symmetric block ciphers. IEEE Transactions on Computer–Aided Design 21(12), 1509–1517 (2002)
[12] Karpovsky, M., Kulikowski, K.J., Taubin, A.: Robust protection against fault injection attacks on smart cards implementing the Advanced Encryption Standard. In: International Conference on Dependable Systems and Networks–DSN 2004, pp. 93–101. IEEE Computer Society (2004)
[13] Malkin, T.G., Standaert, F.-X., Yung, M.: A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 159–172. Springer, Heidelberg (2006)
[14] Wu, K., Karri, R., Kuznetsov, G., Goessel, M.: Low cost error detection for the Advanced Encryption Standard. In: International Test Conference–ITC 2004, pp. 1242–1248. IEEE Computer Society (2004)
[15] Knudsen, L.: Truncated and Higher Order Differentials. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 196–211. Springer, Heidelberg (2003)
[16] Karri, R., Gössel, M.: Parity–based concurrent error detection in symmetric block ciphers. In: International Test Conference–ITC 2003, pp. 919–926. IEEE Computer Society (2003)
[17] Joshi, N., Wu, K., Karri, R.: Concurrent Error Detection Schemes for Involution Ciphers. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 400–412. Springer, Heidelberg (2004)
[18] Karri, R., Kuznetsov, G., Gössel, M.: Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 113–124. Springer, Heidelberg (2003)