

Differential Fault Analysis of RECTANGLE-80

Shobhit Sinha, Sandip Karmakar

Department of Computer Science and Engineering,
Indian Institute of Information Technology Kalyani, India
{shobhit,sandip}@iiitkalyani.ac.in

Abstract. We present various differential fault attack schemes for the RECTANGLE-80 and demonstrate how initially we started from a 80-bit fault to a single word fault scheme. This was mainly due to a differential vulnerability in the S-box of RECTANGLE as a result of which the exhaustive search space for the key reduces from 2^{80} to 2^{32} . We have also presented a key schedule attack that is a variant of the single fault scheme, exploiting the same vulnerability and reduces the search space to 2^{40} . The paper concludes with the simulation results for the single word fault scheme followed by countermeasures.

1 Introduction

With the advent of the IoT era, there is of course an increasing need for lightweight ciphers to provide security in small embedded devices. Moreover, these devices have strong cost constraints like speed, area, power, memory and energy consumption. RECTANGLE [1] is a lightweight block cipher designed to meet such purposes. The design of this cipher meets all these constraints and is extremely hardware friendly. Also, due to its bit-slice implementation, it achieves a very competitive speed among the existing lightweight ciphers.

The designers of RECTANGLE have presented a security analysis of the cipher in [1], indicating its resistance against differential, linear, impossible differential and integral cryptanalysis along with attacks such as the statistical saturation and key schedule. They have reported that even though differential and linear trails exist, their clustering is limited and effective difference or linear propagation cannot be constructed with more than 14 rounds. For the statistical saturation attack, the longest distinguisher can reach 15 rounds and can be used to attack 18 rounds at most. Some impossible differential distinguishers were found for 8-rounds, and some integral distinguishers were found for 7 rounds. To prevent slide attacks on key schedule they have added different round constants for different rounds. Based on this security analysis, the cipher has been designed to run for 25 rounds to resist the above mentioned attacks, the extra 7 rounds after 18 serve as a safe security margin.

The designers have also reported a comparative analysis of RECTANGLE's implementations in different hardware environments against some popular ciphers in literature. The throughput of RECTANGLE is quite remarkable when compared against block ciphers like AES [2], PRESENT [3] or even against

stream ciphers like Grain [4] or Trivium [5]. On software implementations, the bit-slice implementation gives it an edge in speed over the other ciphers which do not have such implementations.

The RECTANGLE is indeed an “interesting” cipher and keeping in mind the above discussion on its security and speed, it can be used for encryption in various devices. However, modern-day ciphers not only need to be resistant against mathematical cryptanalysis, they have to also show robustness against implementation attacks also known as side-channel attacks. The device where the algorithm is implemented in hardware or software, generally, leak information such as power consumption, electromagnetic energy etc. Analysing such leakages may lead to getting information either full or partial about the secret key to an unwanted user. This kind of attacks are known as side-channel attacks. Fault attacks are sometimes classified as a type of side-channel attack. In this scenario faults into the device where the crypto-algorithm is implemented are used to retrieve secret information about the implementation [6]. Almost all the ciphers proposed till date are vulnerable to fault attacks. For example, AES [7, 8, 9, 10, 11], Grain-128 [12], Trivium [13] etc. have fault attacks reported against them. The solidity of RECTANGLE obviously poses the question whether it is resistant to side-channel attacks. This makes its fault analysis against RECTANGLE even more important. There exists a discussion of differential power attack on RECTANGLE in the literature [14]. But, to the best of our knowledge, there is no known differential fault attack on RECTANGLE. In differential fault attack, the faulty ciphertexts and the fault-free ciphertexts output from the implementation are analyzed to deduce partial or full knowledge of the secret key. We present differential fault schemes against RECTANGLE-80 in this paper. Furthermore, we show that it can be attacked with a single fault and minutes of computation. Also the paper proposes fault attacks that target both the state and the key schedule of the cipher, which implies that weakness of the cipher is not only limited to its state update, it is also prevalent to its key schedule algorithm.

Organization:

The paper is organized as follows:

Section 2 : Background

Section 3 : 80/64 bit fault scheme

Section 4 : Single word fault and associated schemes

Section 5 : Fault Scheme for the Key Scheduler

Section 6 : Conclusion

2 Background

2.1 The RECTANGLE-80 Cipher

The plaintext of 64 bits of RECTANGLE is represented as a matrix of 4×16 bits (called the state) whereas the key of 80 bits is represented as matrix of 5×16 bits. The structure of the RECTANGLE-80, as used to perform encryption, is illustrated in Algorithm 1.

Algorithm 1: The RECTANGLE-80 encryption function

Input: The 64-bit plaintext P, The 80-bit key K.

Output: The 64-bit ciphertext C.

```
round  $\leftarrow$  1
X  $\leftarrow$  P
while round  $\leq$  25 do
  X  $\leftarrow$  AddRoundKey(X,K)
  X  $\leftarrow$  SubColumn(X)
  X  $\leftarrow$  ShiftRow(X)
  K  $\leftarrow$  KeySchedule(K,round)
end
X  $\leftarrow$  AddRoundKey(X,K)
return X
```

2.2 Round Transformation

The RECTANGLE is a 25 round SP-network Cipher. Each of the 25 rounds consists of the following 3 steps: AddRoundkey, SubColumn, ShiftRow. After the last round, there is a final AddRoundKey.

AddRoundkey: performs a Bitwise XOR of the round subkey to the intermediate state.

SubColumn: applies S-boxes in a parallel manner to the 4 bits in the same column. The S-box is applied to each of the 16 columns. The S-box used in RECTANGLE is a 4-bit to 4-bit S-box $S : F_2^4 \rightarrow F_2^4$. The S-box values are shown in Table 1.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	6	5	C	A	1	E	7	9	B	0	3	D	8	F	4	2

Table 1. S-box of RECTANGLE

ShiftRow: A cyclic left rotation to each row over different offsets. Row 0 is not rotated, row 1 is left rotated over 1 bit, row 2 is left rotated over 12 bits, and row 3 is left rotated over 13 bits.

2.3 Key Schedule

The KeySchedule function generates the next Round key from the previous key, as shown in Algorithm 2.

Algorithm 2: The RECTANGLE-80 Key Schedule function

Input: r^{th} round key K_r , Round Number r .

Output: $(r + 1)^{th}$ round key K_{r+1} .

$Y \leftarrow K_r$

$Y \leftarrow \text{PartialSubColumn}(Y)$

$Y \leftarrow \text{KeyFeistel}(Y)$

$Y \leftarrow \text{RCXor}(Y,r)$

return Y

At round i ($i = 1, 2, \dots, 25$), the 64-bit round subkey K_i consists of the first 4 rows of the current contents of the key register. After extracting K_i , the key register is updated by follows transformations:

PartialSubColumn: S-box is applied vertically to the bits intersected at the 4 uppermost rows and the 4 rightmost columns.

KeyFeistel: Applying a 1-round generalized Feistel transformation,

$$Row'_0 := (Row_0 \lll 8) \oplus Row_1, \quad Row'_1 := Row_2, \quad Row'_2 := Row_3,$$

$$Row'_3 := (Row_3 \lll 12) \oplus Row_4, \quad Row'_4 := Row_0.$$

RCXor: A 5-bit round constant $RC[i]$ is XORed with the 5-bit key state ($k_4 \parallel k_3 \parallel k_2 \parallel k_1 \parallel k_0$) Finally, K_{25} is extracted from the updated key state. The round constants for each round are fixed and given in [1].

3 Fault Attack: 80/64 bit Fault Scheme

3.1 Specifications :

Fault Timing : After round 1's AddRoundKey operation (for first 64 bits) and after round 2's AddRoundKey operation (for last 16 bits)

Fault Location : A given bit of the Intermediate state

No. of Bits affected : 1

Nature of Fault : Sets the affected bit to 0 (hard fault)

No. of Faults required : 80 or 64

3.2 Details of the Attack:

Let there be a plaintext P such that for all a_i (a single bit) belonging to P , $a_i = 0$. The attacker encrypts P and stores the result in $enc0$.

Recovery of the first 64 bits:

After the initial AddRoundKey operation, the value of a_i becomes k_i . This is the result of $\mathbf{a}_i \leftarrow \mathbf{0} \oplus \mathbf{k}_i$ being performed. The attacker then sets the i^{th} bit of the intermediate state to 0. The rest of the operations proceed normally and a faulty ciphertext c_f is obtained.

Now, if $c_f = enc0$, then $k_i := 0$ otherwise $k_i := 1$ This is because if k_i was

0, then setting it to 0 results in no change and we get a fault-free encryption. Otherwise, if k_i was 1, then setting it to 0 results in a different ciphertext. The attacker repeats the attack for different values of i to recover the first 64 bits.

Recovery of the last 16 bits: The attacker has two options :

a) Brute force the last 16 key-bits in 2^{16} trials. This a trivial task with modern processing powers. It is better than the second option as lesser faults are required. Thus, **the 80 fault scheme reduces to a 64 fault scheme.**

b) Using the 64- bits recovered, the attacker constructs a plaintext such that after the first round transformations the intermediate state becomes 0. We observe that, from the RECTANGLEs S-box Nature that, $\mathbf{S(9)} \rightarrow \mathbf{(0)}$.

Hence, to obtain the special plaintext, we perform a column wise XOR of (1,0,0,1) to the columns of the recovered key. Now, after AddRound Key of Round 2, we fault the i^{th} bit of the 3rd Row in the intermediate state. Similar observations between faulty ciphertext and enc0 follow, and the attack is repeated 16 times for each 3rd Row bit and we deduce the value of the 3rd row of key state produced after the first key schedule. With the relation between 3rd, 4th rows of key and the new 3rd row of key state after key schedule i.e ($Row'_3 = Row_3 \lll 12 \oplus Row_4$) and knowing the values of Row'_3 and Row_3 , we can find out the value of Row_4 and the entire key is now recovered.

3.3 Practicality of the attack :

An attack of a similar nature was proposed in [7] for the Advanced Encryption Standard. The authors mention spike, glitch, optical and electromagnetic perturbations as mechanisms for injecting the fault (setting the bit to 0). However, this fault model poses the following challenges :

- a) The timing and precision should be impeccable for it to succeed.
- b) At least 64 faults are required, which is too high and chances of permanent damage to the device increases. This is because every injected fault will stress the device to some extent and there will be some probability that it will produce a permanent, rather than a transient fault.

4 Fault Attack: Single Word fault scheme

4.1 Specifications :

Fault Timing : At the beginning of the 25th Round
Fault Location : 1st Row of the Intermediate State
No. of Bits affected : 16
Nature of Fault : Flips the affected bit
No. of Faults required : 1

4.2 Details :

Let's analyze the nature of the RECTANGLE's S-box. Let M be a particular column in the intermediate state. Thus, M is a 4-bit integer. The following table shows values of

$$\delta z = S(M) \oplus S(M \oplus f) \quad (1)$$

Here, $f = (8,4,2,1)$ This is analogous to faulting M at first, second, third and fourth bit respectively.

M ⇒	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
f = 8	13	5	15	7	9	1	3	11	13	5	15	7	9	1	3	11
f = 4	7	11	11	3	7	11	11	3	3	15	7	15	3	15	7	15
f = 2	10	15	10	15	6	7	6	7	8	13	8	13	12	13	12	13
f = 1	3	3	6	6	15	15	14	14	11	11	14	14	7	7	6	6

Table 2. δz values for all possible M for different f

Using the above table, we can find the values of δz for a given M and faulty bit f of M. The following table now indicates the frequency of δz values.

δz	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Freq.	0	2	0	8	0	2	6	10	2	2	2	8	2	6	4	10

Table 3. Frequency of all possible δz values

Clearly, a differential analysis results in smaller set of possible values of M (cardinality of the possible values set varies from 2 to 10) as compared to the number of values in brute force case which is 16. An attack scheme based on this info. reduces the key space from 2^{80} to 2^{70} in worst case scenario. Hence, we investigate further the nature of the S-box. Note that value for $\delta z = 0$, frequency = 0. This is obvious as δz is 0 only if there is no fault occurring in M i.e. $f = 0$. The next table uses the above two tables to infer the number of possible values of M, given a δz and fault location(f).

From the table, we can clearly see that faulting the first bit of M, possible values of M in worst case is 2 as compared to second/third/fourth bit where this value is 4. Clearly, an attack scheme based on first bit fault reduces the key space to 2^{32} and based on second/third/fourth bit fault reduces the key space to 2^{48} . Hence, we use the first scheme and present an algorithm that breaks the cipher in a single word fault.

$\delta z \Rightarrow$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f = 8$	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2
$f = 4$	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4
$f = 2$	0	0	0	0	0	0	2	2	0	2	0	2	0	2	4	0
$f = 1$	0	0	0	2	0	0	4	2	0	0	0	2	0	0	4	2

Table 4. Given f and δz , no. of possible values of M

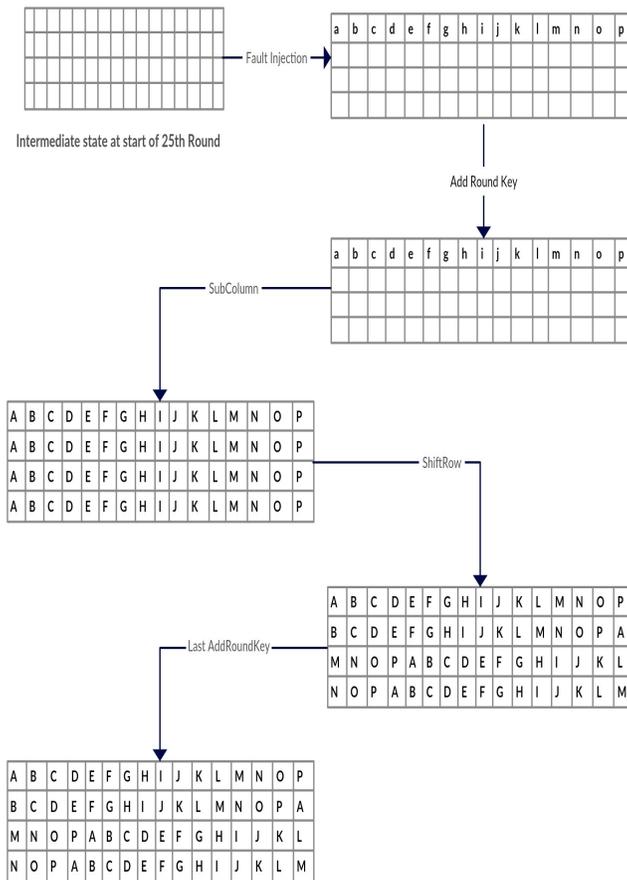


Fig. 1. Propagation of Fault Induced in the input to 25th round of RECTANGLE-80.

Description of Algorithm :

`dev_encrypt()` is the fault-free encryption performed by the device.
`dev_faultyencrypt()` is the encryption with fault injection (as per specifications)

Algorithm 3: DFA on RECTANGLE-80 using a single Word fault

Input: A 64-bit plaintext, P

Output: The 80 bit key K

$C \leftarrow \text{dev_encrypt}(P)$

$C' \leftarrow \text{dev_faultyencrypt}(P)$

DECLARE Δz : [4][16] array of bits

for $i = 0$ **to** 3 **do**

for $j = 0$ **to** 15 **do**

$\Delta z[i][j] = C[i][j] \oplus C'[i][j]$

end

end

$\Delta z \leftarrow \text{ShiftRow}^{-1}(\Delta z)$

/* We now guess the columns of intermediate state after Round 25 by looking at each column value of Δz by using Table 1. There exist two possible values for each column of Δz i.e 2^{16} possible intermediate states. */

$M \leftarrow$ All possible combinations from 2 choices for each 16 columns

$R \leftarrow$ All possible combinations of 2^{16} bits

for each $m \in M$ **do**

$m \leftarrow \text{SubColumn}(m)$

$m \leftarrow \text{ShiftRow}(m)$

$k \leftarrow m \oplus C$

for each $r \in R$ **do**

$K \leftarrow$ append r to k as the 5th Row

$K \leftarrow \text{CompleteInvertKeySchedule}(K)$

if $\text{encrypt}(P, K) = C$ **then**

 return K

end

end

end

performed by the device.

Now, if I_{25} is the intermediate state as input to the Round 25,

If $C \leftarrow dev_encrypt(P)$, then

$$\Rightarrow C = ShiftRow(SubColumn(I_{25} \oplus K_{25})) \oplus K_{last}$$

Also, if $C' \leftarrow dev_faultyencrypt(P)$

$$\Rightarrow C' = ShiftRow(SubColumn(I_{25} \oplus K_{25} \oplus f)) \oplus K_{last}$$

Now, if $\Delta z \leftarrow C \oplus C'$, then

$$\begin{aligned} \Rightarrow \Delta z &= ShiftRow(SubColumn(I_{25} \oplus K_{25})) \\ &\oplus ShiftRow(SubColumn(I_{25} \oplus K_{25} \oplus f)) \end{aligned}$$

$$\Rightarrow ShiftRow^{-1}(\Delta z) = SubColumn(I_{25} \oplus K_{25}) \oplus SubColumn(I_{25} \oplus K_{25} \oplus f)$$

Compare above equation with eqn (1). Now, we can calculate the column value for each column of Δz , which gives us the value of M. We already know the value of f to be 8(as first row is faulted). Using Table 1, we can guess the value of $I_{25} \oplus K_{25}$. Two such possible values exist. Hence, we have 2^{16} possible combinations of the intermediate state. Xoring each combination with C, we get 2^{16} possible combinations of KeyState after 25 rounds. It should be noted that C, C' and Δz are all [4][16] matrices.

However, as the key is of 80 bits, we have no information about the fifth row of the key state. Now, there are 2^{16} possible combinations for the fifth row (16 bits). Hence, we have **2^{32} possible key states** after 25th round. We can now perform an exhaustive search by inverting the key states and find out the value of the correct key by checking if it is giving the correct ciphertext C on encrypting P.

4.3 Practicality of the attack :

Flipping a particular bit at a given instant was considered probabilistic two decades ago [15] but has been demonstrated perfectly in [16]. Using the same technique with particular parameters of the laser beam, it is indeed possible to flip an entire row in the chip leading to a successful fault injection.

Even if the 16 consecutive bits are not flipped at once, the bit that is not flipped can easily be detected with δz value being 0 for the corresponding column. Instead of simultaneous flipping we can flip the bits one by one, an approach similar to the one mentioned in [10] for the AES.

It is also very interesting to note that if we Repeat Steps 1 to 6 but this time with fault injection at a different row than first, we can know the value of K exactly and our exhaustive search reduces to 2^{16} . This obviously **requires two word faults, but with a reduced exhaustive search of 2^{16} .**

5 Fault Attack: Fault scheme for the Key Scheduler

The fault scheme in the key scheduler is a variant of the attack described in section-4. The obvious scheme is to fault the first 16 bits of the Key State at

the end of 24th round. The AddRoundKey operation in the subsequent round then faults the 16 bits of the first row and the key scheduler scheme reduces to the attack in section-4. However, this scheme has a major flaw. The Key Scheduler function (Algorithm-2) of RECTANGLE-80 has a PartialSubColumn round. Faulting the key state in such a manner causes the SubColumn function to change the contents of the topmost and leftmost four columns as well. This change then increases the overall “guess” complexity of our scheme. To avoid this we split our scheme in two phases.

5.1 Phase I

Specifications :

Fault Timing : At the end of the 24th Round
Fault Location : 1st Row of the Key
No. of Bits affected : 12 leftmost bits
Nature of Fault : Flips the affected bit
No. of Faults required : 1

Details :

The flipping of 12 leftmost bits ensure that there is no effect of the PartialSubColumn on the Key State during the fault propagation. This is shown in Fig 2. It is interesting to note that a,b,...,k,l all denote bit flip faults and are analogous to a xor with 1. Hence, we now present an algorithm that first remove the presence of a,b,...,k,l from the cipher text and then we use a modified Algorithm 3 to reduce the possible search space for 12 columns from 2^{48} to 2^{12} .

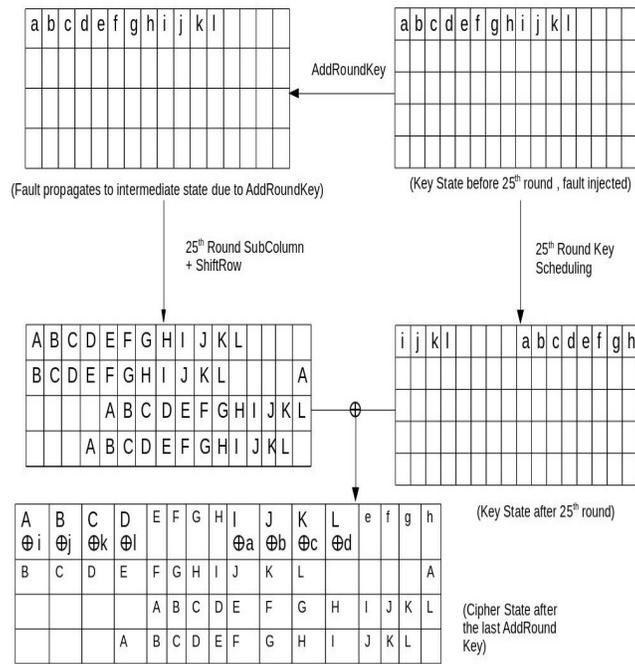


Fig. 2. Propagation of Fault in Phase I of Key Scheduler Scheme

5.2 Phase II

Specifications :

Fault Timing : After 25th Round's AddRoundkey

Fault Location : 1st Row of the KeyState

No. of Bits affected : 4 right most bits

Nature of Fault : Flips the affected bit

No. of Faults required : 1

Details :

We now flip the four rightmost bits of the first row of the key state. Due to the PartialSubColumn function of the key schedule, the same differential vulnerability of the S-box will be exploited to reduce the possible search space for four columns from 2^{16} to 2^4 . The fault propagation in this case is shown in Fig 3. The AddRoundConstant function will not contribute to the propagation of the fault in the keystate.

Algorithm 4: Phase-I of Key Schedule attack on RECTANGLE-80

Input: A 64-bit plaintext, P
Output: 2^{12} choices for 48 bits of the 25th Round intermediate state

$C \leftarrow \text{dev_encrypt}(P)$
 $C' \leftarrow \text{dev_faultyencrypt}(P)$
 DECLARE $\Delta z : [4][16]$ array of bits
for $i = 0$ **to** 3 **do**
 | **for** $j = 0$ **to** 15 **do**
 | | $\Delta z[i][j] = C[i][j] \oplus C'[i][j]$
 | **end**
end
 $\Delta z[0] \leftarrow \Delta z[0] \oplus \{1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1\}$
 $\Delta z \leftarrow \text{ShiftRow}^{-1}(\Delta z)$
 /* We now guess the columns of intermediate state after Round 25 by
 looking at each column value of Δz by using Table 1. There
 exist two possible values for each of 12 leftmost columns of Δz
 i.e 2^{12} possible intermediate states. */
 M_left \leftarrow All possible combinations from 2 choices for each of the leftmost 12
 columns of the intermediate state
 return M_left

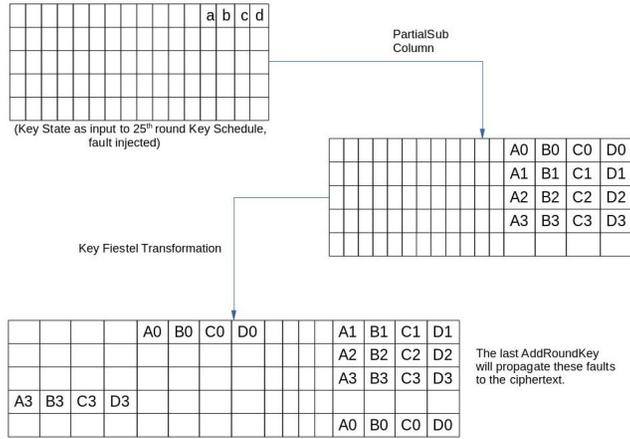


Fig. 3. Propagation of Fault in Phase II of Key Scheduler Scheme

Using this knowledge of the keyspace, coupled with the information of phase-I and Key-Scheduler Function, attacker can easily calculate the following bits, with '1' representing bit calculated using information by Phase 1 and '2' using information from both P1 and P2. This is shown in Table 5. The rest of the bits can be found using a brute-force approach. If only phase-I is implemented the

Algorithm 5: Phase-II of Key Schedule attack on RECTANGLE-80

Input: A 64-bit plaintext, P
Output: 2^4 choices for 16 bits of the 25th Round Key state
 $C \leftarrow \text{dev_encrypt}(P)$
 $C' \leftarrow \text{dev_faultyencrypt}(P)$
 DECLARE Δz : [4][16] array of bits
for $i = 0$ **to** 3 **do**
 for $j = 0$ **to** 15 **do**
 $\Delta z[i][j] = C[i][j] \oplus C'[i][j]$
 end
end
 $\Delta z \leftarrow \text{KeyFiestel}^{-1}(\Delta z)$
 K_right \leftarrow All possible combinations from 2 choices for each of the rightmost 4
 columns of the key state
 return K_right

key space reduces to 2^{44} and if both Phase-I and Phase-II are implemented then the key-space reduces to 2^{40} .

1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2
				1	1	1	1	1	1	1	1	2	2	2	2
			2	1	1	1	1	1	1	1	1	2	2	2	2
												2	2	2	2

Table 5. Bits recovered with the attack on the key schedule

6 Conclusion

We have proposed different fault schemes for RECTANGLE 80 . A brief summary of each scheme is given in Table 6. For the single word fault scheme, when simulated (non-bit slice implementation) in C on a single processor machine, we observed that the InvertKeySchedule() takes $2.6 \mu s$ whereas the encrypt() takes $2.9 \mu s$ on average for a single execution (based on 1,000,000 encryptions). Hence, it can be calculated that the best case timing of $5.5 \mu s$ (instant match, when the last row of K_{25} is 0x0000) and the worst case to be 6.5 hours (traversing the entire 2^{32} search space, when the last row of K_{25} is 0xFFFF). This can further be improved using GPUs and multicore processors along with bit slice-implementation. As most RECTANGLE implementations use the bit-slice technique (12 instructions required for SubColumn round[1]), our word fault can be applied more precisely in such cases by other mechanisms like clock-glitching,

the equipments of which are cheaper than that of the laser-based bit-flipping. The word fault is better than the key schedule attack as the key schedule attack requires greater accuracy in fault injection. As a countermeasure, to prevent the single fault attack described in this paper the last two rounds would need to be repeated to check that no fault was injected.

Fault Scheme	Reduced key space
80-bit fault	2^0
64-bit fault	2^{16}
Two word faults	2^{16}
Single word fault	2^{32}
Key-Schedule :Phase-I only	2^{44}
Key-Schedule :Both phases	2^{40}

Table 6. Fault Schemes for RECTANGLE-80 and search complexity

References

1. Zhang W T, Bao Z Z, Lin D D, et al. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *Sci China Inf Sci*, 2015, 58: 122103(15), doi: 10.1007/s11432-015-5459-7
2. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). FIPS Publication 197, available for download at <http://www.itl.nist.gov/fipspubs/>, 2001.
3. Bogdanov A. et al. (2007) PRESENT: An Ultra-Lightweight Block Cipher. In: Pailier P., Verbauwhede I. (eds) *Cryptographic Hardware and Embedded Systems - CHES 2007*. CHES 2007. Lecture Notes in Computer Science, vol 4727. Springer, Berlin, Heidelberg
4. M. Hell, T. Johansson, A. Maximov, and W. Meier. A Stream Cipher Proposal: Grain-128. *IT, IEEE International Symposium on*, pages 16141618, 2006.
5. C. D. Canniere and B. Preneel. Trivium specifications. eSTREAM, ECRYPT Stream Cipher Project, 2006.
6. E. Biham, A. Shamir, Differential fault analysis of secret key cryptosystems, *Proc. of CRYPTO 97*, Springer LNCS vol. 1294, pp. 513525, 1997.
7. J. Blömer and J.-P. Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In R. N. Wright, editor, *Financial Cryptography FC 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 162181. Springer-Verlag, 2003.
8. P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on A.E.S. In J. Zhou, M. Yung, and Y. Han, editors, *Applied Cryptography and Network Security ACNS 2003*, volume 2846 of *Lecture Notes in Computer Science*, pages 293306. Springer-Verlag, 2003.
9. G. Piret and J.-J. Quisquater. A differential fault attack technique against SPN structure, with application to the AES and KHAZAD. In C. D. Walter, . K. Ko, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems CHES 2003*,

- volume 2779 of Lecture Notes in Computer Science, pages 7788. Springer-Verlag, 2003.
10. C. Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, International Conference Advanced Encryption Standard AES 2004, volume 3373 of Lecture Notes in Computer Science, pages 2741. Springer-Verlag, 2004.
 11. Michael Tunstall, Debdeep Mukhopadhyay, and Sk Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In WISTP, pages 224233, 2011.
 12. Alexandre Berzati, Cecile Canovas, Guilhem Castagnos, Blandine Debraize, Louis Goubin, Aline Gouget, Pascal Paillier, and Stephanie Salgado. Fault Analysis of Grain-128. Hardware-Oriented Security and Trust, IEEE International Workshop on, 0:714, 2009.
 13. Hojsk, M., Rudolf, B.: Differential fault analysis of Trivium. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 158172. Springer, Heidelberg (2008)
 14. Selvam, R., Shanmugam, D., Annadurai, S.: Side channel attacks: Vulnerability analysis of prince and rectangle using dpa
 15. S. Skorobogatov, R. Anderson, Optical Fault Induction Attacks, Proc. of CHES 02, Springer LNCS, pp. 212, 2002.
 16. Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, et al.. How to Flip a Bit?. On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International, Jul 2010, Corfu, Greece.