Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 31(11): 6820–6832.
DOI: 10.3934/era.2023344
Received: 25 June 2023
Revised: 16 October 2023
Accepted: 17 October 2023
Published: 20 October 2023

Research article

Meet-in-the-middle differential fault analysis on Midori

Chunyan An[1,2,*], **Wei Bai**[1,2] **and Donglei Zhang**[1,2]

[1] State Grid Smart Grid Research Institute Co., Ltd., Beijing 102209, China

[2] Electric Power Intelligent Sensing Technology Laboratory of State Grid Corporation, Beijing 102209, China

* **Correspondence:** Email: anchunyan@geiri.sgcc.com.cn.

Abstract: Midori is a lightweight block cipher designed by Banik et al. and presented at the ASIACRYPT 2015 conference. According to the block size, it consists of two algorithms, denoted as Midori-64 and Midori-128. Midori generates 8-bit S-Boxes from 4-bit S-Boxes and applies almost MDS matrices instead of MDS matrices. In this paper, we introduce the meet-in-the-middle fault attack model in the 4-round cell-oriented fault propagation trail and reduce the key space in the last round by $2^{45.71}$ and $2^{39.86}$ for Midori-64 and Midori-128, respectively. For Midori-64, we reduce the time complexity from 2^{80} to 2^{28}, 2^{32} and 2^{56} for the different single fault injection approaches. For Midori-128, we provide a 4-round fault attack method, which slightly increases the complexity compared to previous attacks. Our results indicate that the first and last four rounds of Midori must be protected to achieve its security.

Keywords: meet-in-the-middle attack; differential fault analysis; Midori; fault attack

1. Introduction

Internet of Things (IoT) devices are often limited by storage, computing and energy consumption, and cannot run high-strength encryption algorithms and protocols. Lightweight algorithms and protocols have emerged to ensure device security while saving resource consumption. However, these devices always generate, process, transmit and store private information. The security of these devices is receiving increasing attention and the cryptographic technology is the key to ensuring these security requirements. Therefore, the widespread use of resource constrained devices has attracted attention to lightweight cryptographic ciphers, such as Ascon [1], CLEFIA [2] , HIGHT [3], KATAN [4], LED [5], Midori [6], Piccolo [7], PRESENT [8], PRINCE [9] and so on.

Lightweight block cipher Midori is designed by Banik et al. [6] and presented at the ASIACRYPT 2015 conference. It can be divided into two variants, Midori-64 and Midori-128, depending on the

block size. In order to reduce energy consumption during algorithm execution, the optimization part of Midori includes replacing 8-bit S-Boxes with 4-bit S-Boxes and replacing maximum separable distance (MDS) matrices with almost MDS binary matrices. With these optimizations, Midori seems to achieve lower latency while maintaining a smaller area. However, the security (including both theoretical and practical security) of lightweight ciphers is a key factor in protecting security sensitive data within the chip from attackers. The theoretical analysis results of Midori include differential analysis [6], linear analysis [6], truncated differential and related-key differential attacks [10], meet-in-middle attack [11] and impossible differential analysi [12–14]. In fact, the practical attacks are more important for lightweight ciphers, but for Midori, such cryptanalysis [15–18] are not adequate.

Differential fault analysis (DFA) is a typical cryptanalysis technique used to attack cryptographic implementations and devices. It was introduced by Biham and Shamir [19] against DES like cryptographic systems. Up to now, a large number of cryptographic algorithms are threatened by the differential fault analysis, such as AES [20, 21], DES [22] and SM4 [23] (also called SMS4). The countermeasures against fault attacks can be divided into two categories. The first type detects fault injection by adding hardware sensors. The main drawback of this attack is that the fault detection technique targets exact fault injection methods and cannot prevent all fault injection methods. The second category requires increasing hardware surfaces and number of operations to protect the hardware implementation from fault attacks. A more effective approach is to have a trade-off between efficiency and protection. This type of countermeasure typically protects the hardware implementation by resisting the state-of-the-art fault attacks. The same countermeasures have been applied to AES [24], DES [25], etc.

In [15], Cheng et al. introduced a 3-round fault propagation property of single fault. Based on the vulnerability caused by the almost MDS matrix, they analyzed distinct patterns of nonzero differentials in the ciphertexts and found that the fault injection position could be inferred only using correct and faulty ciphertext. By retrieving the related subkeys, the secret key search space is reduced from 2^{128} to 2^{80} for Midori-64 and from 2^{128} to 2^{32} for Midori-128, respectively. In [16], Wang et al. extended the differential propagation trail to 4 rounds and evaluated the remained key entropy of Midori. Based on the 4-round fault propagation path, the key entropy of Midori-64 and Midori-128 decreased to 99.72 and 71.98 bits, respectively. The key recovery process requires a 3-round manual differential analysis and/or algebraic fault analysis, and the entropy of Midori-128 was ultimately reduced to 68.47 bits. In [17], Nozaki et al. injected faults in the last 2 rounds and proposed a 2 stages statistical fault analysis to analyze the 128-bit key of Midori. For Midori-64, the whitening key WK is recovered by injecting 16 different faults in the last round. Subsequently, they used the whitening key to decrypt 1 round and obtained the output of the 15-th round. Then some new faults are injected into the 15-th round, so that they can guess 65,536 keys for one column at a time. In [18], Li et al. proposed a ciphertext-only fault analysis with 6 different distinguishers to break Midori. They encrypt the same plaintext under the same key and inject faults in the penultimate round to obtain enough ciphertexts. Then, a differential analysis process is executed column by column to recover the secret key used in the last 2 rounds. Their work also indicates that the hamming weight distinguisher works best, requiring 280 and 132 ciphertexts for Midori-64 and Midori-128, respectively. In Table 1, we compare our works with previous fault attacks on Midori.

Using the meet-in-the-middle fault attack [26], 7 conditions can be attached to 4-round cell-oriented fault propagation trail, which allows us to reduce key space of the last round. Comparison with the

work of [16], the key space is reduced from $2^{99.72}$ to $2^{54.01}$ for Midori-64 and from $2^{71.98}$ to $2^{32.12}$ for Midori-128. In addition, using the 4-round fault trail, the complexity of the secret key recovery for Midori-64 is reduced from the previous 2^{80} to 2^{28}, 2^{32} and 2^{56} for the different cases. Our results indicate that the first and last four rounds of Midori must be protected to maintain the security.

Table 1. Comparison of this work with previous fault attacks on Midori.

Algorithm	Attacks	Rounds	Number of faults	Complexity	Ref.
Midori-64	Statistical fault analysis	2	64	2^{19}	[17]
Midori-64	Ciphertext-only fault analysis	2	280	$2^{20.13}$	[18]
Midori-64	Differential fault analysis	3	2	2^{80}	[15]
Midori-64	Algebraic fault analysis	4	2	$2^{15.35}$	[16]
Midori-64	Differential fault analysis	4	6	2^{28}	Ours
Midori-128	Statistical fault analysis	2	64	2^{18}	[17]
Midori-128	Ciphertext-only fault analysis	2	132	$2^{31.05}$	[18]
Midori-128	Differential fault analysis	3	2	2^{32}	[15]
Midori-128	Algebraic fault analysis	4	2	$2^{24.69}$	[16]
Midori-128	Differential fault analysis	4	6	$2^{32.12}$	Ours

The structure of the remaining part of the paper is as follows. Section 2 gives a brief description of Midori and provides some notations. Section 3 studies the 4-round fault propagation trail. Section 4 introduces the key recovery approach and experimental results, and Section 5 summarizes the article.

2. Description of Midori and notations

Midori is an SPN structure block cipher with a key size of 128 bits. Its state is represented by a 4 by 4 matrix and is defined in Eq (2.1), where $s_i \in \{0, 1\}^m$ and the length m of each cell is 4 (for Midori-64) or 8 (for Midori-128).

$$S = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \tag{2.1}$$

The number of rounds to be performed during the execution of the algorithm is dependent on the block size n. The number of rounds is represented by R, where $R = 16$ and $R = 20$ when $n = 64$ and $n = 128$, respectively.

2.1. SBoxes and matrices

Midori contains 2 bijective 4-bit S-Boxes, Sb_0 and Sb_1. The 8-bit S-Boxes of Midori-128 are generated by Sb_1, as detailed in [6]. For Midori-64, Sb_0 is applied to each cell of state S: $s_i = \mathrm{Sb}_0(s_i), 0 \le i \le 15$. Similarly, for Midori-128, 4 8-bit S-Boxes SSb_0, SSb_1, SSb_2 and SSb_3 are utilized, please see the design document [6] for details. The matrix M of Midori is defined in Eq (2.2) and the 4 m-bit values (x_0, x_1, x_2, x_3) are updated by Eq (2.3).

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{2.2}$$

$$(x_0, x_1, x_2, x_3)^T = M \cdot (x_0, x_1, x_2, x_3)^T \tag{2.3}$$

2.2. Round function

Midori's round function consists of four steps, namely S-layer SubCell, P-layer ShuffleCell and MixColumn, and key-addition layer KeyAdd. The n-bit state is updated at each layer as follows.

SubCell(S): For Midori-64, Sb_0 is applied to every 4-bit cell, i.e., $s_i = \mathrm{Sb}_0(s_i)$, where $0 \le i \le 15$. For Midori-128, $s_i = \mathrm{SSb}_{i \bmod 4}(s_i)$, where $0 \le i \le 15$.

ShuffleCell(S): Each cell of the state is permuted as follows: $(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}) = (s_0, s_{10}, s_5, s_{15}, s_{14}, s_4, s_{11}, s_1, s_9, s_3, s_{12}, s_6, s_7, s_{13}, s_2, s_8)$.

MixColumn(S): The matrix M is used for updating each column of S, i.e., $(s_i, s_{i+1}, s_{i+2}, s_{i+3})^T = M \cdot (s_i, s_{i+1}, s_{i+2}, s_{i+3})^T$, where $i = 0, 4, 8, 12$.

KeyAdd(S, RK_i): The i-th n-bit round key RK_i is XORed to the state S.

2.3. Ciphers

For encryption, Midori uses a round function that is composed of four different cell-oriented transformations including SubCell, ShuffleCell, MixColumn and KeyAdd. The plaintext or ciphertext is loaded into the state. After an initial whitening key addition, the state is transformed by implementing a round function 16 or 20 times, with the final round differing slightly from the first $R - 1$ rounds. The round function is parameterized using a key schedule and the overall structure of encryption is depicted in Figure 1. For decryption, the inverse round function consists of SubCell, MixColumn, InvShuffleCell and KeyAdd. The permutation of InvShuffleCell is defined as follows: $(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}) = (s_0, s_7, s_{14}, s_9, s_5, s_2, s_{11}, s_{12}, s_{15}, s_8, s_1, s_6, s_{10}, s_{13}, s_3, s_4)$. The round key used by KeyAdd of the inverse round function is performs the following operations: $L^{-1}(RK_i) = \mathrm{InvShuffleCell} \cdot \mathrm{MixColumn}(RK_i)$. Then, $L^{-1}(RK_i)$ is XORed to the state S.

2.4. Key schedule

For Midori-64, the 128-bit master key is divided into 2 64-bit values K_0 and K_1. The whitening key $WK = K_0 \oplus K_1$ and the round key $RK_i = K_{i \bmod 2} \oplus \alpha_i$, where $0 \le i \le 14$. For Midori128, $WK = K$ and $RK_i = K \oplus \beta_i$, where $0 \le i \le 19$. The specific values of constants α_i and β_i can be found in [6]. In our attack process, Midori-128 relies on the last round to recover the secret key K, while Midori-64 needs to utilize the last two rounds to obtain the 128-bit K.

2.5. Notations

DFA applies key recovery attacks using the relationship between the secret key and behavior information under the faults in the intermediate state. It derives information about the secret key through the

difference between correct and faulty ciphertext (using the same plaintext). For the sake of description, some symbols are introduced in this part.

1) X_i represents the input of the i-th round, where $0 \leq i \leq R$, X_0 is the plaintext and X_R is the ciphertext.
2) $X_i[j]$ denotes the j-th cell of X_i , where $0 \leq j \leq 15$.
3) RK_i is the round key for the the i-th round, where $1 \leq i < R$ and $RK_0 = RK_R = WK$.
4) ΔX is the XOR difference of 2 states X and X', i.e., $\Delta X = X \oplus X'$.
5) δx is the XOR difference of 2 cells x and x', i.e., $\delta x = x \oplus x'$.

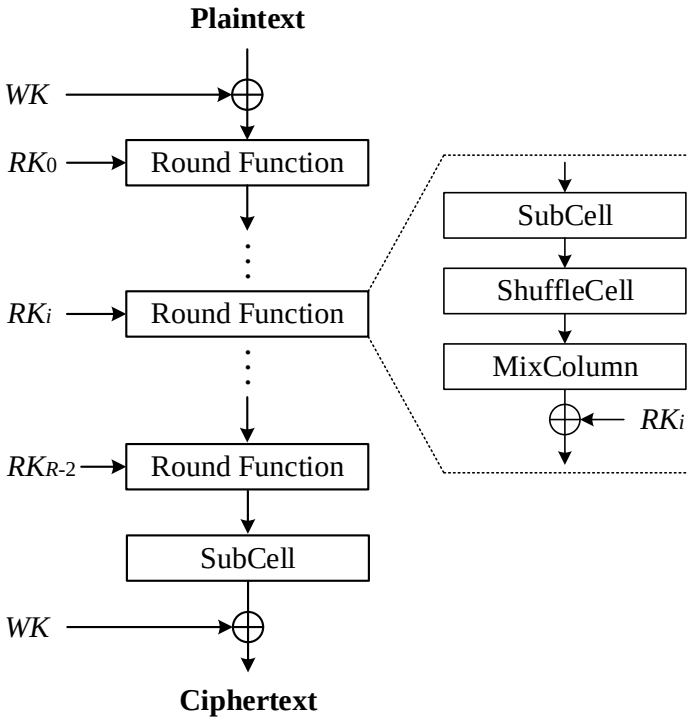

Figure 1. The overall encryption structure of Midori.

3. Fault Propagation and properties of Midori

3.1. Fault Propagation in the last 4 rounds

For the sake of simplicity, we assume that the fault occurred in the first cell. As shown in Figure 2, the falut F introduced in the (R-3)-th round is converted to F_1 after SubCell and ShuffleCell. Due to the influence of MixColumn layer, the difference of the other three cells belonging to the same column becomes F_1, and the KeyAdd step has no effect on the difference of S. The output difference of the (R-2)-th round is $\Delta S_{R-2} = \{\delta s_4 = \delta s_5 = \delta s_6 = F_2, \delta s_8 = \delta s_{10} = \delta s_{11} = F_3, \delta s_{12} = \delta s_{13} = \delta s_{15} = F_3\}$. The output differential values after the (R-1)-th round is $\Delta S_{R-1} = \{\delta s_0 = F_5, \delta s_1 = F_6, \delta s_2 = F_7, \delta s_3 =$

$F_8, \delta s_4 = \delta s_7 = F_9, \delta s_5 = F_{4c}, \delta s_6 = F_{2a}, \delta s_8 = \delta s_9 = F_{10}, \delta s_{10} = F_{2c}, \delta s_{11} = F_{3a}, \delta s_{12} = \delta s_{14} = F_{11}, \delta s_{13} = F_{4a}, \delta s_{15} = F_{3b}\}$. Here, $F_6 \oplus F_7 \oplus F_8 = 0$, $F_9 = F_{4c} \oplus F_{2a}$, $F_{10} = F_{2c} \oplus F_{3a}$, $F_{11} = F_{4a} \oplus F_{3b}$.

For any ciphertext C and C', by defining $OD(WK[i]) = \text{Subcell}(C[i] \oplus WK[i]) \oplus \text{Subcell}(C'[i] \oplus WK[i])$, the equation system (3.1) will be obtained. If the attacker knows which byte injected a fault, combining equation system (3.1), the $15m$ bits of WK will be recovered. There are two forms of equations in system (3.1), the first satisfying $OD(WK[i]) = OD(WK[j])$ and the second conforming to $OD(WK[i]) \oplus OD(WK[j]) = OD(WK[k])$. The method in [26] will be applied to recover the corresponding key with a complexity of 2^m and 2^{2m}, respectively.

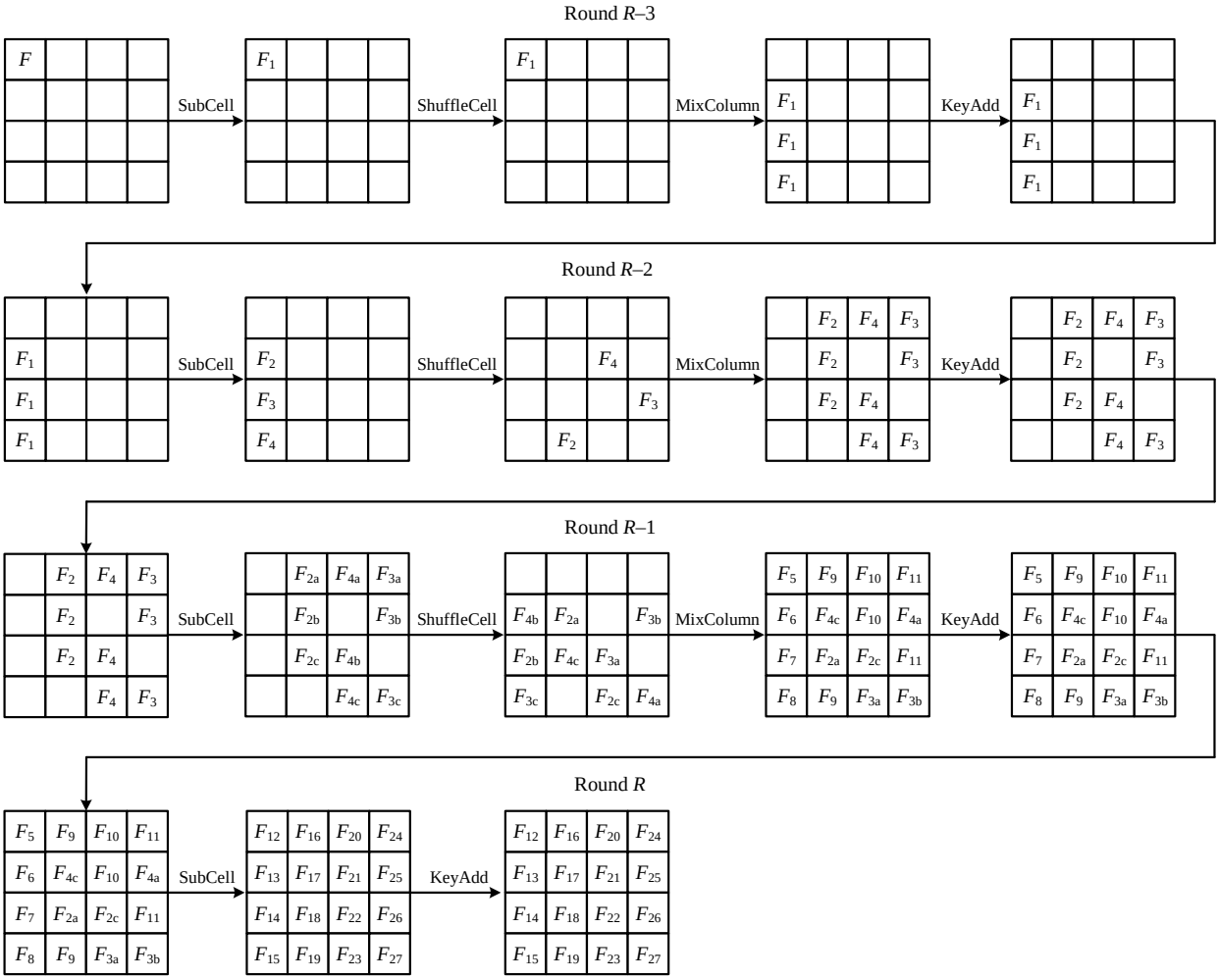

Figure 2. Fault Propagation in last four rounds.

After WK is recovered, we will obtain S_{R-1} and S'_{R-1} by decrypting C and C'. For state S_{R-1} and S'_{R-1}, assuming $SID(RK_{R-1}[i], RK_{R-1}[j], RK_{R-1}[k]) = \text{Subcell}(S_{R-1}[i] \oplus S_{R-1}[j] \oplus S_{R-1}[k] \oplus RK_{R-1}[i] \oplus RK_{R-1}[j] \oplus RK_{R-1}[k]) \oplus \text{Subcell}(S'_{R-1}[i] \oplus S'_{R-1}[j] \oplus S'_{R-1}[k] \oplus RK_{R-1}[i] \oplus RK_{R-1}[j] \oplus RK_{R-1}[k])$, the equation system (3.2) will be acquired. For Midori-64, the 64-bit RK_{14} will be obtain with complexity

of 2^{3m} and the details are provided in Section 4.1.

$$\begin{cases} OD(WK[4]) \oplus OD(WK[7]) = 0 \\ OD(WK[8]) \oplus OD(WK[9]) = 0 \\ OD(WK[12]) \oplus OD(WK[14]) = 0 \\ OD(WK[1]) \oplus OD(WK[2]) \oplus OD(WK[3]) = 0 \\ OD(WK[4]) \oplus OD(WK[5]) \oplus OD(WK[6]) = 0 \\ OD(WK[9]) \oplus OD(WK[10]) \oplus OD(WK[11]) = 0 \\ OD(WK[12]) \oplus OD(WK[13]) \oplus OD(WK[15]) = 0 \end{cases} \tag{3.1}$$

$$\begin{cases} SID(RK_{14}[4], RK_{14}[5], RK_{14}[6]) \oplus SID(RK_{14}[0], RK_{14}[1], RK_{14}[3]) = 0 \\ SID(RK_{14}[4], RK_{14}[5], RK_{14}[6]) \oplus SID(RK_{14}[8], RK_{14}[9], RK_{14}[10]) = 0 \\ SID(RK_{14}[12], RK_{14}[13], RK_{14}[14]) \oplus SID(RK_{14}[0], RK_{14}[2], RK_{14}[3]) = 0 \\ SID(RK_{14}[12], RK_{14}[13], RK_{14}[14]) \oplus SID(RK_{14}[4], RK_{14}[5], RK_{14}[7]) = 0 \\ SID(RK_{14}[8], RK_{14}[9], RK_{14}[11]) \oplus SID(RK_{14}[12], RK_{14}[14], RK_{14}[15]) = 0 \\ SID(RK_{14}[8], RK_{14}[9], RK_{14}[11]) \oplus SID(RK_{14}[0], RK_{14}[1], RK_{14}[2]) = 0 \end{cases} \tag{3.2}$$

From here until the end of the paper, we assume that t represents the number of pairs used for the attack. There are 16 different single-cell fault injection positions and each form corresponds to a set of equations similar to the equation systems (3.1) and (3.2). Meanwhile, we should consider 3 different fault injection approaches. In the first case, we know the exact location of the fault injection, but it may not necessarily be in the same location. The specific equation systems that match the fault ciphertext pairs will be used for the key recovery process. In the second scenario, we inject faults using the same method at the same time, i.e., fault occurs in the same cell. In the third case, which is the worst case, we do not impose any constraints on the falut injection location. Here, for all correct and faulty ciphertext pairs, we need to calculate each of the 16 intermediate states and this requires an additional $2^{4t}(= 16^t)$ time complexity.

3.2. Characteristics of SBoxes

For x, y, z and the differences δx, δy, δz, the relationship of input differential and output differential in the SubCell is defined as follows:

$$Nc2(\delta x, \delta y) = \#\{x, y \in F_{2^m} | OD(x, \delta x) = OD(y, \delta y)\} \tag{3.3}$$

$$Nc3(\delta x, \delta y, \delta z) = \#\{x, y, z \in F_{2^m} | OD(x, \delta x) = OD(y, \delta y) \oplus OD(z, \delta z)\} \tag{3.4}$$

When only 1 cell-oriented fault is injected and tested 2^{24} times for Midori-64, the positions of the faults satisfy the uniform distribution and the maximum probability of Eqs (3.3) and (3.4) are $2^{-2} = 64/256$. Equations (3.3) and (3.4) only involve 2 and 3 key calculations, and only 4 and 6 pairs of ciphertexts are needed to reduce the size of the candidate key set to 1. All equations in (3.2) require guessing the keys of 6 cells, and the size of the candidate key set obtained using 6 ciphertext pairs is $2^{12} = 2^{4\times6} \times 2^{-2\times6}$. We need an additional 6 pairs of ciphertext to filter 2^{12} candidate keys. Fortunately, the equations in (3.2) can provide an additional 3 filtering conditions, each of which can

filter 2^{12} keys. For example, the filtering condition corresponding to the first two equations in (3.2) is $SID(RK_{14}[0], RK_{14}[1], RK_{14}[3]) = SID(RK_{14}[8], RK_{14}[9], RK_{14}[10])$, which can reduce the size of the candidate key set for the first two equations to 1. Thus, we only need 6 pairs of ciphertext to reduce the number of candidate key sets to an acceptable level. In the following sections, we will provide the corresponding experimental results to recover the 128-bit key of Midori.

Algorithm 1 2-cell key recovery procedure

Input: a pair of correct and/or faulty ciphertexts (C, C'), cell number i and j.
Output: candidate key list $L_{i,j}$.

for $k = 0$ to $2^m - 1$ **do** {2^m time and memory}
  Set $WK[i] = k$.
  Calculate $OD(WK[i])$ and store $(OD(WK[i]), WK[i])$ in Hash Table L_1.
end for
for $k = 0$ to $2^m - 1$ **do** {2^m time and 1 memory}
  Set $WK[j] = k$.
  Calculate $OD(WK[j])$ and look in L_1 corresponding to $OD(WK[i])$.
  if $OD(WK[i]) = OD(WK[j])$ **then**
    Add $(WK[i], WK[j])$ to the candidate key list $L_{i,j}$.
  end if
end for

Algorithm 2 3-cell key recovery procedure

Input: a pair of correct and/or faulty ciphertexts (C, C'), cell number i, j and k.
Output: candidate key list $L_{i,j,k}$.

for $x = 0$ to $2^m - 1$ **do** {2^m time and memory}
  Set $WK[i] = x$.
  Calculate $OD(WK[i])$ and store $(OD(WK[i]), WK[i])$ in Hash Table L_1.
end for
for all $(WK[j], WK[k]) \in F_{2^m} \times F_{2^m}$ **do** {2^{2m} time and 1 memory}
  Calculate $Temp = OD(WK[j]) \oplus OD(WK[k])$ and look in L_1 corresponding to $OD(WK[i])$.
  if $Temp = OD(WK[i])$ **then**
    Add $(WK[i], WK[j], WK[k])$ to the candidate key list $L_{i,j,k}$.
  end if
end for

4. Key recovery and experiment

4.1. Key recovery of Midori

In our attack, we implement a 1-byte fault injection between the MixColumn of the (R-4)-th round and the MixColumn of the (R-3)-th round on Midori. Assuming our fault attack approach requires t correct and/or faulty ciphertext pairs, denoted as $(C_i, C_i')(1 \le i \le 6)$, we can use the same method as

in Section 3.1 to derive 16 different fault propagation patterns (the work of Cheng et al. also induced fault propagation patterns). Based on the fault injection way, we will propose 3 different key recovery attacks, i.e., 2-cell key recovery procedure, 3-cell key recovery procedure and 6-cell key recovery procedure, please see the Algorithms 1–3 for details.

If a fault is fixed in the first cell which corresponds to the first case of Section 3.1, the following technique allows us to recover $15m$ bits of WK in time 6×2^{2m} and memory 6×2^m for Midori, see also Algorithm 4. In this algorithm, the time complexity and memory usage of steps 2 to 7 are 2^m. The time complexity of steps 8 to 15 is 2^{2m}, and the memory consumption is 2^m. Thus, the key recovery process need 6×2^{2m} time and 6×2^m memory. In fact, the memory usage of Algorithm 4 can be reduced to 2^m. By guessing all the values of $WK[0]$, the 128 bits WK will be recovered for Midori-128. However, for Midori-64, we can use equation system (3.2) to recover 64-bit RK_{14}. With the help of the key generation algorithm, the 128 bits key of Midori-64 can be recovered with 6×2^{4m} time and 6×2^{3m} memory.

When the faults occur in an unknown specific cell, which corresponds to the second scenario, we need to traverse all the 16 patterns one by one, and the time complexity of the attack requires an additional 16, i.e., 6×2^{5m} for Midori-64 and 6×2^{3m} for Midori-128. For random single-cell faults, the time complexity of our attack is $6 \times 2^{5m} \times 2^{4t}$ for Midori-64 and $6 \times 2^{3m} \times 2^{4t}$ for Midori-128.

Algorithm 3 6-cell key recovery procedure

Input: a pair of correct and/or faulty ciphertexts (C, C'), cell number i, j, k, l, m and n.
Output: candidate key list $L_{i,j,k,l,m,n}$.

1: **for all** $(WK[i], WK[j], WK[k]) \in F_{2^m} \times F_{2^m} \times F_{2^m}$ **do** $\{2^{3m}$ time and memory$\}$
2: Calculate $\Delta = SID(RK_{14}[i], RK_{14}[j], RK_{14}[k])$ and store $(\Delta, RK_{14}[i], RK_{14}[j], RK_{14}[k])$ in Hash Table L_1.
3: **end for**
4: **for all** $(WK[l], WK[m], WK[n]) \in F_{2^m} \times F_{2^m} \times F_{2^m}$ **do** $\{2^{3m}$ time and 1 memory$\}$
5: Calculate $Temp = OD(WK[l]) \oplus OD(WK[m] \oplus OD(WK[n])$ and look in L_1 corresponding to Δ.
6: **if** $Temp = \Delta$ **then**
7: Add $(WK[i], WK[j], WK[k], WK[l], WK[m], WK[n])$ to the candidate key list $L_{i,j,k,l,m,n}$.
8: **end if**
9: **end for**

Table 2. Subkey recovery for $WK[4]$ and $WK[7]$.

Number of Ciphertexts	Number of Candidate Keys	Expected Value	Proportion
1	45.3	64	90.7%
2	13.5	16	87.1%
3	7.3	4	80.1%
4	5.1	1	64.5%
5	4.8	1	71.4%
6	4.6	1	74.8%

4.2. Experimental result

Our attacks implemented a PC using Visual Studio 2019 with Intel(R) Core(TM) i5-10210U CPU and 16 GB memory. We use software simulation for the fault injection. Six simulated faults are induced into the first cell of the (R-3)-th round. We tested the first and fourth equations of (3.1) using 6 ciphertext pairs, and the experimental results are shown in Table 2. Furthermore, we tested the first two equations of (3.2), and the results are shown in Table 3. Each test is measured 10,000 times, and the average is taken as the final result.

Table 3. Subkey recovery for $WK[1]$, $WK[2]$ and $WK[3]$.

Number of Ciphertexts	Number of Candidate Keys	Expected Value	Proportion
1	332.4	1024	98.9%
2	36.1	256	99.1%
3	9.2	64	99.0%
4	3.8	16	98.8%
5	2.9	4	96.7%
6	2.7	1	92.5%

From the above tables, it can be seen that the differential propagation characteristics of Midori are much worse than expected. As shown in Table 2, the fault propagation deviates from the expected value by $1/4$, which will increase the time complexity by $13.3 = (0.75^{-3})^3$. The complexity of 3.4 has increased by $13.3 = 0.75^{-9}$. The fault propagation in 3 deviates from the expected value by 0.075 and the time complexity increase by $3.49 = (0.925^{-4})^4$. For Midori-64, the key space of the last round is reduced to $2^{54.01} = 2^4 \times 45.3^3 \times 332.4^4$ using the single fault injection. The time complexity of the attack is reduced from 2^{80} to $2^{28} = 6 \times 2^{16} \times 13.3 \times 13.3 \times 3.49$, 2^{32} and 2^{56}. Tor Midori-128, the time complexity is slightly increased from 2^{32} to $2^{32.12} = 6 \times 2^{24} \times 13.3 \times 3.49$.

5. Conclusions

In this paper, we present a 4-round cell-oriented fault propagation trail and categorize the single fault injection approachs into 3 types, i.e., exactly known the fault injuction location, faults occurred in an unknown specific cell and injucted random single-cell faults. When the faults occur in the specific cells, we reduced the key space of the last round by $2^{45.71}$ and $2^{39.86}$ for Midori-64 and Midori-128. For Midori-64, the complexity of key recovery is reduced from 2^{80} to 2^{28}. In the second type of fault injection style, the time complexity is decreased to 2^{32} for Midori-64. For random single-cell faults, the time complexity of our attack is 2^{56} for Midori-64. Moreover, we have also demonstrated that 4 rounds of Midori must be protection to achieve its security.

Algorithm 4 Key Recovery of WK

Input: pairs of correct and/or faulty ciphertexts $(C_i, C'_i)((1 \leq i \leq 6))$.
Output: candidate key list L.

for $1 \leq i \leq 6$ **do**
  Call Algorithm 1 with Parameters (C_i, C'_i), 4, 7 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[4]$ and $WK[7]$.
  Call Algorithm 1 with Parameters (C_i, C'_i), 8, 9 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[8]$ and $WK[9]$.
  Call Algorithm 1 with Parameters (C_i, C'_i), 12, 14 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[12]$ and $WK[14]$.
  Call Algorithm 2 with Parameters (C_i, C'_i), 1, 2, 3 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[1]$, $WK[2]$ and $WK[3]$.
  Call Algorithm 2 with Parameters (C_i, C'_i), 4, 5, 6 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[4]$, $WK[5]$ and $WK[6]$.
  Call Algorithm 2 with Parameters (C_i, C'_i), 9, 10, 11 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[9]$, $WK[10]$ and $WK[11]$.
  Call Algorithm 2 with Parameters (C_i, C'_i), 12, 13, 15 and obtain temporary list L_i.
  Search the intersection of L_i and store them as candidate keys of $WK[12]$, $WK[13]$ and $WK[15]$.
end for

Use of AI tools declaration

The authors declare that they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by Science and Technology Projects of State Grid Corporation of China (Research on Lightweight Secure Connection Technologies for Electric Low Power Wireless Sensor Network with both Wide-band and Narrow-band Terminals, No. 5500-202158416A-0-0-00).

Conflict of interest

The authors declare that there are no conflicts of interest.

References

1. C. Dobraunig, M. Eichlseder, F. Mendel, M. Schläffer, Ascon v1.2: lightweight authenticated encryption and hashing, *J. Cryptology*, **34** (2021), 1–42. https://doi.org/10.1007/s00145-021-09398-9
2. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, T. Iwata, The 128-bit blockcipher CLEFIA (extended abstract), in *Fast Software Encryption* (eds. A. Biryukov), Springer, (2007), 181–195. https://doi.org/10.1007/978-3-540-74619-5_12

3. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, et al., HIGHT: a new block cipher suitable for low-resource device, in *Cryptographic Hardware and Embedded Systems* (eds. L. Goubin, M. Matsui), Springer, (2006), 46–59. https://doi.org/10.1007/11894063_4
4. C. Cannière, O. Dunkelman, M. Knežević, KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers, in *Cryptographic Hardware and Embedded Systems* (eds. C. Clavier, K. Gaj), Springer, (2009), 272–288. https://doi.org/10.1007/978-3-642-04138-9_20
5. J. Guo, T. Peyrin, A. Poschmann, M. Robshaw, The LED block cipher, in *Cryptographic Hardware and Embedded Systems* (eds. B. Preneel, T. Takagi), Springer, (2011), 326–341. https://doi.org/10.1007/978-3-642-23951-9_22
6. S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, et al., Midori: a block cipher for low energy, in *International Conference on the Theory and Application of Cryptology and Information Security* (eds. T. Iwata, J. H. Cheon), Springer, (2015), 411–436. https://doi.org/10.1007/978-3-662-48800-3_17
7. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, T. Shirai, Piccolo: an ultra-lightweight blockcipher, in *Cryptographic Hardware and Embedded Systems* (eds. B. Preneel, T. Takagi), Springer, (2011), 342–357. https://doi.org/10.1007/978-3-642-23951-9_23
8. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, et al., PRESENT: an ultra-lightweight block cipher, in *Cryptographic Hardware and Embedded Systems* (eds. P. Paillier, I. Verbauwhede), Springer, (2007), 450–466. https://doi.org/10.1007/978-3-540-74735-2_31
9. J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, et al., PRINCE - a low-latency block cipher for pervasive computing applications, in *International Conference on the Theory and Application of Cryptology and Information Security* (eds. X. Wang, K. Sako), Springer, (2012), 208–225. https://doi.org/10.1007/978-3-642-34961-4_14
10. X. Dong, Y. Shen, Cryptanalysis of reduced-round Midori64 block cipher, preprint. Available from: https://eprint.iacr.org/2016/676.
11. L. Lin, W. Wu, Meet-in-the-middle attacks on reduced-round Midori-64, *IACR Trans. Symmetric Cryptology*, **2017** (2017), 215–239. https://doi.org/10.13154/tosc.v2017.i1.215-239
12. Z. Chen, H. Chen, X. Wang, Cryptanalysis of Midori128 using impossible differential techniques, in *Information Security Practice and Experience* (eds. F. Bao, L. Chen, R. Deng, G. Wang), Springer, (2016), 1–12. https://doi.org/10.1007/978-3-319-49151-6_1
13. M. Tolba, A. Abdelkhalek, A. M. Youssef, Improved multiple impossible differential cryptanalysis of Midori128, in *IEICE Transactions on Fundamentals of Electronics, Communications and Computer*, **E100-A** (2017), 1733–1737. https://doi.org/10.1587/transfun.E100.A.1733
14. A. R. Shahmirzadi, S. A. Azimi, M. Salmasizadeh, J. Mohajeri, M. R. Aref, Impossible differential cryptanalysis of reduced-round Midori64 block cipher, *ISC Int. J. Inf. Secur.*, **10** (2018), 3–13. https://doi.org/10.22042/isecure.2018.110672.399
15. W. Cheng, Y. Zhou, L. Sauvage, Differential fault analysis on Midori, in *Information and Communications Security* (eds. K. Y. Lam, C. H. Chi, S. Qing), Springer, (2016), 307–317. https://doi.org/10.1007/978-3-319-50011-9_24

16. Y. Wang, X. Zhao, F. Zhang, S. Guo, L. Wu, W. Li, et al., Security evaluation for fault attacks on lightweight block cipher Midori, *J. Cryptologic Res.*, **4** (2017), 58–78. https://doi.org/10.13868/j.cnki.jcr.000163
17. Y. Nozaki, Y. Ikezaki, M. Yoshikawa, Two stages statistical fault analysis method for Midori and its evaluation, *Electron. Commun. Jpn.*, **101** (2018), 3–11. https://doi.org/ 10.1002/ecj.12057
18. W. Li, L. Liao, D. Gu, S. Cao, Y. Wu, J. Li, et al., Ciphertext-only fault analysis on the Midori lightweight cryptosystem, *Sci. China Inf. Sci.*, **63** (2020), 139112. https://doi.org/10.1007/s11432-018-9522-6
19. E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems, in *Conference on the Theory and Application of Cryptography* (eds. A. J. Menezes, S. A. Vanstone), Springer, (1990), 2–21. https://doi.org/10.1007/3-540-38424-3_1
20. C. Giraud, DFA on AES, in *International Conference on Advanced Encryption Standard* (eds. H. Dobbertin, V. Rijmen, A. Sowa), Springer, (2004), 27–41. https://doi.org/10.1007/11506447_4
21. M. Tunstall, D. Mukhopadhyay, S. Ali, Differential fault analysis of the advanced encryption standard using a single fault, in *Information Security Theory and Practice* (eds. C. A. Ardagna, J. Zhou), Springer, (2011), 224–233. https://doi.org/10.1007/978-3-642-21040-2_15
22. L. Hemme, A differential fault attack against early rounds of (triple-)DES, in *Cryptographic Hardware and Embedded Systems* (eds. M. Joye, J. J. Quisquater), Springer, (2004), 254–267. https://doi.org/10.1007/978-3-540-28632-5_19
23. R. Li, B. Sun, C. Li, J. You, Differential Fault Analysis on SMS4 using a single fault, *Inf. Process. Lett.*, **111** (2011), 156–163. https://doi.org/10.1016/j.ipl.2010.11.011
24. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, Passive and active combined attacks on AES combining fault attacks and side channel analysis, in *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, (2010), 10–19. https://doi.org/10.1109/FDTC.2010.17
25. M. Rivain, Differential fault analysis on DES middle rounds, in *Cryptographic Hardware and Embedded Systems* (eds. C. Clavier, K. Gaj), Springer, (2009), 457–469. https://doi.org/10.1007/978-3-642-04138-9_32
26. P. Derbez, P. A. Fouque, D. Leresteux, Meet-in-the-middle and impossible differential fault analysis on AES, in *Cryptographic Hardware and Embedded Systems* (eds. B. Preneel, T. Takagi), Springer, (2011), 274–291. https://doi.org/10.1007/978-3-642-23951-9_19