Decomposition of higher-order nonlinear S-boxes in lightweight block ciphers for algebraic fault analysis

Xing Fang (fancy_t@bupt.edu.cn)  
Beijing University of Posts and Telecommunications  
https://orcid.org/0000-0002-1801-6831

Hongxin Zhang  
Beijing University of Posts and Telecommunications

Xiaotong Cui  
Beijing University of Posts and Telecommunications

Fan Fan  
Institute of Microelectronics

Research Article

Keywords: S box, Nonlinear, Decomposition, Skinny, Algebraic equation

Posted Date: June 30th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1787786/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
Decomposition of higher-order nonlinear S-boxes in lightweight block ciphers for algebraic fault analysis

Xing Fang · Hongxin Zhang · Xiaotong Cui · Fan Fan

Received: date / Accepted: date

Abstract Nowadays, with the development of Internet of Things and information security technologies, lightweight block ciphers are gradually being widely used. As a side-channel attack method, algebraic fault analysis has received attention from experts and scholars since its introduction. The only nonlinear operation in lightweight block ciphers is S-box substitution, and the performance index of S-box directly determines the security strength of the cipher. In order to further improve the efficiency of algebraic fault analysis, this paper proposes a method to rewrite the algebraic equations of S-box substitution by decomposing the original cubic S-boxes into two quadratic S-boxes. The results show that this method is significantly effective compared to the original method in GIFT 64 and SKINNY 64, especially in SKINNY 64 block cipher, where the average solving time is reduced by several hundred times in the best case with the same samples. At the same time, the need for the number of faults is reduced, and at least 2 faulty ciphertexts can be used to recover the master key. In addition, the PRESENT 64 block cipher is also studied in this paper, and the results show that the method can also improve the efficiency significantly when the number of fault injection rounds is deep.

Keywords S-box · Nonlinear · Decomposition · Skinny · Algebraic equation

1 Introduction

Along with the booming Internet of Things industry, the design, manufacture and use of chips are on a large scale. At the same time, with the popularity of smart cards, sensors and other low power devices, the demand for lightweight cryptography and related security technology is becoming increasingly urgent. Therefore, various lightweight ciphers have been proposed in recent years. Examples of lightweight ciphers include PRESENT[1], LED[2], SKINNY[3], GIFT[4] and SIMON[5] family. These lightweight cryptographic ciphers are gradually known, recognized and used because of their obvious advantages.

Even though an important goal of lightweight block cipher is to achieve effective encryption under the condition of limited computational resources[6, 7], its most important and core objective is the security of the cryptography. At present, there are many kinds of attacks[8, 9, 10] against block ciphers, including differential analysis, integral analysis, linear analysis, algebraic analysis, differential fault analysis, algebraic fault attack. As an efficient key-recovery method, fault attack[11, 12] technology has been widely used since it was proposed. It is of great practical significance to analyze the security performance of lightweight block cipher under different attacks.

The existing literatures on GIFT block cipher’s security research mainly includes differential analysis, rectangle analysis, differential fault attack, Biclique analysis, chosen plaintext attack, persistent fault attack et al. The existing

X. Fang
Beijing University of Post and Telecommunications, Beijing 100876, China
E-mail: fancy_t@bupt.edu.cn

X. Cui
E-mail: cuixiaotong@bupt.edu.cn

H. Zhang(✉)
Beijing University of Post and Telecommunications, Beijing 100876, China
Beijing Key Laboratory of Work Safety Intelligent Monitoring (Beijing University of Post and Telecommunications), Beijing 100876, China
E-mail: hongxinzhang@bupt.edu.cn

F. Fan
Beijing Microelectronics Technology Institute, Beijing 100076, China
E-mail: 18911650922@163.com
literals on SKINNY block cipher's mainly includes Bi-clique analysis, impossible differential fault attack, differential attack, persistent fault attack, related-key attacks et al. The method of algebraic fault analysis was proposed by Courtois et al. in 2010[13], they attacked DES, and the results show that it is very effective in recovering keys. Zhang Fan et al. proposed a framework for algebraic fault analysis of lightweight block ciphers in 2016[14], which provides some general methods for establishing algebraic fault analysis equations. By comparing other methods, algebraic fault analysis can achieve key-recovery using a smaller number of samples, which is an efficient means of attack. Therefore, it is relevant to perform algebraic fault analysis for these ciphers.

As the only nonlinear component of the block ciphers, the performance of the S-box is one of the important indicators to evaluate the security of this cryptographic algorithm. The S-boxes of these two lightweight block ciphers (GIFT-64 and SKINNY-64) are cubic, it can be troublesome to build algebraic equations. Due to the characteristics of CryptoMiniSAT Solver, it is unable to deal with quadratic or more than quadratic variables, so it is necessary to introduce some new intermediate variables to replace higher-order variables in the construction of each round of S-box equations. In order to reduce the total number of variables and the number of CNF equations in the whole algebraic equation, we propose an S-box decomposition method. This approach is inspired by a technical solution based on a threshold implementation[16, 17, 18] proposed in the literature[15] by Poschmann et al. in 2011, who successfully implemented a decomposition of a cubic S-box equation into two quadratic equations. By taking the decomposition of the S-boxes, we succeeded in getting the three intermediate variables out of the way, and the number of intermediate variables used in total by the changed S-boxes was reduced.

In this paper, we combine S-box decomposition with the classical algebraic fault analysis method to effectively investigate some lightweight block ciphers.

Firstly, we use the algebraic fault analysis framework proposed by Zhang et al. to analyze GIFT-64 block cipher and SKINNY-64 block cipher. According to the different structure of these ciphers, the diffusion of a single fault in the operation of these ciphers are illustrated in the form of schematic diagrams, and the optimal round of fault injection for the two ciphers is given through experiments.

Then, we decomposed the S-boxes of the GIFT-64 and SKINNY-64 block ciphers, respectively. Using the method proposed in the literature[15], we successfully decomposed the cubic S-box into two quadratic S-boxes and transformed them into the form of algebraic equations. A new S-box algebraic equation is used to replace the original S-box algebraic equation and re-establish the algebraic fault analysis framework.

To verify the effect after decomposition, we did a large number of experiments with the same experimental sample premise to analyze the relationship between the average solution time and the success rate of the solution in a specific time. The experimental results show that by decomposition, there is some improvement for the GIFT-64 block cipher, and the effect is significant for the SKINNY-64 block cipher. The optimal fault injection scenario of SKINNY-64 is explored under the new method.

To illustrate the applicability of this approach in lightweight block ciphers, the PRESENT cipher is also investigated in this paper. Unlike the other two ciphers, the number of intermediate variables about the S-box part is not reduced after the decomposition, but only the three cubic variables and five quadratic variables are turned into eight quadratic variables. The experimental results show that this method improve the solution efficiency when fault injected to deeper round.

2 Algebraic fault analysis for GIFT-64

In this section, the GIFT-64 block cipher research process is explained in detail in order to better illustrate the algebraic fault analysis method. Firstly, the structural characteristics of the GIFT-64 are introduced, and the process of establishing the encryption equation and the fault injection equation are given. According to the characteristics of the cipher, the optimal scheme of fault injection is analyzed.

2.1 The structure of GIFT-64 cipher

GIFT is a new lightweight block cipher designed by Banik et al. to commemorate the 10th anniversary of Present block cipher in 2017. It uses the classical SPN (substitution permutation network) structure and absorbs the advantages of PRESENT block cipher. At the same time, the optimization of SBoxes and P-Permutation makes the cipher remove the constraint that the branch number of SBoxes must be 3 in the Present cipher and avoid finding the high-probability difference feature that the number of active SBoxes in each round is 1. Therefore, it is more difficult to find the iterative differential route, which makes harder for finding non-random features to recover the key. GIFT has two versions GIFT-128: 28 rounds with a block size of 64 bit, and GIFT-128-128: 40 rounds with a block size of 128-bit. Both the versions have 128-bit keys. For this work, we focus only on GIFT-128. Figure 1 gives the encryption flowchart of the GIFT-64 block cipher.

(1) Initialization

The cipher state S is first initialized from the 64-bit plaintext represented as 16 nibbles of 4-bit represented as
Algorithm 1: The Encryption of GIFT\textsubscript{64}

\begin{verbatim}
Input: P.K
Output: C
1 \textbf{i} := 0, i \leq 28;
2 \textbf{i} \textbf{while} i \leq r - 1 \textbf{do}
3 \quad C := F(\text{ARK(\textbf{PB}(SC(P))) \oplus K});
4 \textbf{end}
5 \textbf{return} C
\end{verbatim}

Algorithm 2: The Key Expansion of GIFT\textsubscript{64}

\begin{verbatim}
Input: K
Output: subkey\textsubscript{0}, subkey\textsubscript{1}, \ldots, subkey\textsubscript{r - 1}
1 \textbf{i} := 0, i \leq 28;
2 \textbf{while} i \leq r - 1 \textbf{do}
3 \quad subkey\textsubscript{i} := U \parallel V, U \leftarrow k\textsubscript{i}, V \leftarrow k_{0};
4 \quad k_{7} \parallel k_{6} \parallel \cdots \parallel k_{0} \leftarrow k_{1} \gg 2 \parallel k_{0} \gg 12 \parallel \cdots \parallel k_{3} \parallel k_{2};
5 \textbf{end}
6 \textbf{return} subkey\textsubscript{i}
\end{verbatim}

Table 1 S-box of GIFT\textsubscript{64}

<table>
<thead>
<tr>
<th>(X)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>S[X]</td>
<td>1</td>
<td>4</td>
<td>a</td>
<td>c</td>
<td>6</td>
<td>f</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>d</td>
<td>b</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>e</td>
</tr>
</tbody>
</table>

\(w_{15} \parallel w_{14} \parallel \cdots \parallel w_{0}\). The cipher also receives a 128-bit key \(K = k_{7} \parallel k_{6} \parallel \cdots \parallel k_{0}\) as the key state, where \(k_{i}\) is a 16-bit word. Algorithm 1 shows the process of the encryption of GIFT\textsubscript{64}.

(2) The round function

Each round of the cipher comprises of a SubCells followed by PermBits and AddRoundKey which is a XOR with the round-key and predefined constants.

- Subcells

The S-box is applied to every nibble of the cipher state. Table 1 gives the action of this S-box in hexadecimal notation.

- PermBits

The bit permutation used in GIFT\textsubscript{64} is given in [4]. The purpose of this operation is to map bits from bit position \(i\) of the cipher state to bit position \(P(i)\).

---

AddRoundKey

This operation is a combination of two parts: 32-bit round key and 7-bit round constants. The round key is extracted from the key state, it is further partitioned into 2 words \(RK = U \parallel V = u_{15}u_{14}\cdots u_{0} \parallel v_{15}v_{14}\cdots v_{0}\). Round key addition is an XOR operation between \(U, V\) and \(b_{4i+1}\), \(b_{4i}\) of the cipher state after PermBits. \(b_{4i+1} \leftarrow b_{4i+1} \oplus u_{i}, b_{4i} \leftarrow b_{4i} \oplus v_{i}, \forall i \in \{0, \cdots, 15\}\). The 6-bit round constant C and a single-bit ‘1’ is XORed to the cipher state by following definition: \(b_{63} \leftarrow b_{63} + 1, b_{23} \leftarrow b_{23} + c_{5}, b_{19} \leftarrow b_{19} + c_{4}, b_{15} \leftarrow b_{15} + c_{1}, b_{11} \leftarrow b_{11} + c_{2}, b_{7} \leftarrow b_{7} + c_{1}, b_{3} \leftarrow b_{3} + c_{0}\). The round constants are generated by 6-bit affine LFSR, whose update function is defined as: \((c_{5}, c_{4}, c_{3}, c_{2}, c_{1}, c_{0}) \leftarrow (c_{4}, c_{3}, c_{2}, c_{1}, c_{5} \oplus c_{4} \oplus 1)\).

---

KeyExpansion

After AddRoundKey, the key register is updated by following definitions: \(k_{2} \parallel k_{3} \parallel \cdots \parallel k_{0} \leftarrow k_{1} \gg 2 \parallel k_{0} \gg 12 \parallel \cdots \parallel k_{3} \parallel k_{2};\) where \(\gg i\) is an \(i\) bits right rotation within a 16-bit word. Algorithm 2 shows the process of the key expansion of GIFT\textsubscript{64}.

2.2 Building the algebraic equation set for GIFT\textsubscript{64}

- Represent SubCells

In GIFT\textsubscript{64}, 16 S-boxes \(s_{0}, s_{1}, \ldots, s_{15}\) are used in encryption in one round. In order to construct the equation of the all S-boxes, let’s suppose the input of the S-box is \((x_{3} \parallel x_{2} \parallel x_{1} \parallel x_{0})\) and the output of S-box is \((y_{3} \parallel y_{2} \parallel y_{1} \parallel y_{0})\), where \(x_{3}\) and \(y_{3}\) denote the most significant bit, \(x_{0}\) and \(y_{0}\) denote the least significant bit. The Boolean equations corresponding to the individual bits of any non-linear function(in our case S-box) are typically represented using Algebraic Normal Form(ANF). We use four equations with the method of literature[19] to express the relationship between eight variables, the equations are shown as:

\begin{equation}
y_{0} = x_{0} + x_{1} + x_{2} + x_{3} + x_{0}x_{1}
y_{1} = x_{0} + x_{2} + x_{3} + x_{0}x_{1} + x_{0}x_{2}
y_{2} = x_{1} + x_{2} + x_{0}x_{3} + x_{1}x_{3} + x_{1}x_{2}x_{3}
y_{3} = x_{0} + x_{1} + x_{3} + x_{0}x_{2}x_{3}
\end{equation}

Generalized to all S-boxes of the \(r\) rounds, the equations are expressed as follows, where \(0 \leq r \leq 27, 0 \leq i \leq 15\):

\begin{equation}
y'_{0} = 1 + x_{4i} + x_{4i+1} + x_{4i+2} + x_{4i+3} + x_{4i+4}x_{4i+1}
y'_{4i+1} = x_{4i} + x_{4i+2} + x_{4i+3} + x_{4i+4}x_{4i+1} + x_{4i+5}x_{4i+2}
y'_{4i+2} = x_{4i} + x_{4i+2} + x_{4i+3} + x_{4i+4}x_{4i+3} + x_{4i+5}x_{4i+4} + x_{4i+6}x_{4i+3} + x_{4i+7}x_{4i+2}x_{4i+3} + x_{4i+8}x_{4i+3}
y'_{4i+3} = x_{4i} + x_{4i+3} + x_{4i+4}x_{4i+3} + x_{4i+5}x_{4i+4} + x_{4i+6}x_{4i+3} + x_{4i+7}x_{4i+2}x_{4i+3} + x_{4i+8}x_{4i+3}
\end{equation}
When the CryptoMinisat solver is used for algebraic analysis, due to the nature of the solver itself, quadratic and quadratic+ variables cannot be processed in the process of linearization of the equations. In order to achieve the purpose of normal operation, six unknown variables need to be introduced to establish the algebraic equation of a single S-box. Let \( q_{di}, q_{di+1}, q_{di+2}, q_{di+3}, q_{di+4}, q_{di+5} \) represent \( x_{4i}x_{4i+1}, x_{4i}x_{4i+2}, x_{4i}x_{4i+3}, x_{4i+1}x_{4i+2}, x_{4i+1}x_{4i+3}, x_{4i+2}x_{4i+3}, x_{4i+3}x_{4i+4} \), respectively. According to the equation of \( q_{di+5} = x_{4i+1}x_{4i+2}x_{4i+3}, \) four extra equations can be introduced to represent this relationship:

\[
\begin{align*}
    x_{4i+1} & \lor q_{di+5} = 1 \\
    x_{4i+2} & \lor q_{di+5} = 1 \\
    x_{4i+3} & \lor q_{di+5} = 1 \\
    x_{4i+1} \lor x_{4i+2} \lor x_{4i+3} \lor q_{di+5} & = 1
\end{align*}
\]

- **Represent PermBits:**

Suppose the input and output of this operation are \( x_{63} \parallel x_{62} \parallel \cdots \parallel x_0 \) and \( y_{63} \parallel y_{62} \parallel \cdots \parallel y_0 \) respectively. According to the Table 2, the \( i \)-th bit of the input is mapped to the \( -i \)-th bit of the output. The PermBits operation can be expressed by follows, where \( 0 \leq i \leq 63 \):

\[
y_{p(i)} + x_i = 0 \tag{4}
\]

Generalizing to all rounds, the equations are expressed as follows, where \( 0 \leq r \leq 27, 0 \leq i \leq 15 \):

\[
y_{p(i)} + x'_i = 0 \tag{5}
\]

- **Represent AddRoundKey:**

Let the \( x_{63} \parallel x_{62} \parallel \cdots \parallel x_0 \) and \( y_{63} \parallel y_{62} \parallel \cdots \parallel y_0 \) be the input and output of this operation respectively. Suppose the sub-key is \( (u_{15} \parallel u_{14} \parallel \cdots \parallel u_0) \parallel (v_{15} \parallel v_{14} \parallel \cdots \parallel v_0) \), and it can be illustrated as the following equations, where \( 0 \leq i \leq 15 \):

\[
x_{4i+1} + u_i + y_{4i+1} = 0 \\
x_{4i} + v_i + y_{4i} = 0 \tag{6}
\]

Consider the addition of round constant, let the \( (c_5 \parallel c_4 \parallel \cdots \parallel c_0) \) be the round constant, the equations can be illustrated as the follows, where \( 0 \leq i \leq 15 \):

\[
x_{63} + y_{63} + 1 = 0 \\
x_{4i+3} + y_{4i+3} + c_i = 0 \tag{7}
\]

- **Represent KeyExpansion:**

The equations can be established according to the relationship between the sub-key and the master key. So far, all the algebraic equations of the correct encryption have been constructed.

According to the algebraic equation given above, each round of SubCells can be represented with 160 variables and 384 CNF equations, each round of PermBit can be represented with 64 variables and 64 CNF equations, each round of AddRoundKey can be represented with 64 variables and 64 CNF equations, so each round can be represented with 326 variables and 544 CNF equations.

### 2.3 Building equation set for fault injection

In our study, we do not consider fault injection into the Key-Expansion part. Assume that the correct intermediate value is \( X \) in the encryption, \( X = x_{63} \parallel x_{62} \parallel \cdots \parallel x_0 \). Let \( Y \) denote the faulty value of \( X \), \( Y = y_{63} \parallel y_{62} \parallel \cdots \parallel y_0 \). Let \( w \) be the width of a single fault. There are \( m \) possible locations for the injected faults where \( m = 64 / w \). According to the location of the fault we establish the fault equations. (Key recovery in the case of unknown fault location is not discussed in this article). \( Z = z_{63} \parallel z_{62} \parallel \cdots \parallel z_0 \) is assumed to be the difference of the \( X \) and \( Y \).

\[
Z = X \oplus Y; \quad z_i = x_i \oplus y_i, \quad 0 \leq i \leq 63 \tag{8}
\]

It is worth noting that in this paper we inject faults of the single-bit type. Assuming that the fault location is known and the fault is injected at the \( j \)-th position, the algebraic equation can be expressed as follows.

\[
z_i = 0; \quad 0 \leq i \leq 63 \text{ and } i \neq j \tag{9}
z_i \neq 0; \quad i = j
\]

### 2.4 Fault injection to the GIFT_64

With the appearance of laser fault injection, precise fault injection has become a reality. In this scenario, the fault can be injected to the encryption precisely with single-bit. The fault \( f \) is injected into the input of the SubCells.

**Algorithm 3:** Fault Injection to GIFT_64

```plaintext
Input: P, K
Output: C'
1: i = 0; r = 28;
2: while i <= r - 1 do
3:     if i = r_{\text{fault}} then
4:         P_i = P_i^f;
5:     end
6: C' = F(ARK(PB(SC(P)) \oplus K));
7: end
```

Algorithm 3 gives the process of fault injection in the GIFT_64 block cipher. At the time the cipher encrypts to the
specific round of fault-injection, the fault is injected into some registers, which causes the value in that register to change (P<sub>i</sub> represents the input of the i-th round). In fact, the type and location of the fault can be unknown, but what is discussed in our study is done with the premise that the type of the fault is known and the location is known.

The pseudo code for algebraic fault analysis for the GIFT 64 block cipher is shown in Algorithm 4. The inputs to the algorithm are P, num, R<sub>fault</sub> and N, which represent the plaintext, number of faults, the round of fault injection and number of experiments, respectively. The outputs are T<sub>j</sub> and T<sub>ave</sub>, representing the j-th solving time and the average solving time, respectively. In one instance, we perform a single correct encryption against a fixed plaintext and create an algebraic equation. Then we build fault injection equations according to the different locations of fault injection, and combine num fault-injection equations with the correct encryption equations to form a total set of CNF equations for algebraic fault analysis. It is worth stating that all experiments in this paper are conducted under this fault-injection model.

Algorithm 4: The AFA of GIFT 64

Input: P, num, R<sub>fault</sub>, N
Output: T<sub>j</sub>, T<sub>ave</sub>
1 for i = 0; i < r;i + + do 
2 Generate the equation set of Key Expansion; 
3 Generate the equation set of correct encryption 
4 end 
5 C = Enc(P, K); 
6 for j = 1; j < num; j += do 
7 C<sub>j</sub> = Enc(P<sub>j</sub>, K) 
8 end 
9 Generate the equation of P, C, C<sub>j</sub>; 
10 for i = 1; i < num;i += do 
11 for R = R<sub>fault</sub>; l < 32; l += do 
12 Generate the equation sets of fault-injection encryption 
13 end 
14 end 
15 T = RunAFA(); 
16 Set the number(N) of samples; 
17 T<sub>ave</sub> = Sum(T<sub>j</sub>)/N;

Figure 2 shows the spread of single-bit or nibble fault injected into one 25th round’s S-box. It can be found that by the time the fault spread to 28th’s output, a single fault injected in one 25th round’s S-box has affected all the bits. It can be concluded that when the fault injection is too deep, the information redundancy increases and the algebraic equation is difficult to solve. In previous experiments, fault locations were randomly generated at the 64 bits input, which resulted in multiple faults in the same S-box. Based on the diffusion analysis in Figure 2, this is an inefficient way for the experiment, and faults should be injected into as many different S-boxes as possible. To take an extreme example, according to the rule of P permutation, if all the 16 randomly generated faults are exactly 16 bits lower, then fault diffusion only affects the 25th round’s 8-bit key and does not provide more algebraic information for the other 24-bit sub-key. Meanwhile, because these 16 faults only cause the failures of the 0th, 4th, 8th and 12th S-boxes in the 26-round input, when the fault diffusion occurs, only 8 bits of sub-key are affected in 26th round, so the 64-bit random injection fault approach is extremely ineffective. In our experiments, all faults are randomly injected into different S-boxes so that key-recovery can be done more efficiently.

3 S-box decomposition for lightweight block cipher

In the process of building the algebraic equations of lightweight block ciphers, the S-box, as a nonlinear component, is usually a bit more complicated than other operations in the algebraic equations building process. Because of the characteristics of the Solver, we add some intermediate variables to represent the algebraic relationship of the S-box. In order to reduce the influence of higher order equations on the solution, we adopt a new method to rewrite the algebraic equations of S-box without affecting the function of S-box. Figure 3 gives the flow chart of the AFA of lightweight block ciphers based on S-box decomposition. Unlike traditional algebraic fault analysis, we need to decompose the original S-box before performing algebraic fault analysis. The goal is
to reduce the number of intermediate variables used and the number of CNF equations by turning the original S-box from a cubic S-box into two quadratic S-boxes. Then, the original algebraic equations are replaced by the set of algebraic equations of the two new S-boxes for the experiment. First, we need to decompose the cubic S-box. An effective decomposition principle is to eliminate the cubic variables and make the number of quadratic types as small as possible. In 2011, Poschmann et al. proposed a technique to decompose a cubic S-box function into two quadratic functions. This relation can be expressed by the following equations $S(X) = H(F(X))$, where $S, F, H : GF(2^4) \rightarrow GF(2^3)$. Considering the input and output of $G(X)$ as 4-bits vectors $X = (x, y, z, w)$ and $F(X) = (f_0(X), f_1(X), f_2(X), f_3(X))$. Each $f_i$, as a quadratic Boolean function, can be represented in ANF as following equation, where $a_i, a_j$ are the binary coefficients of the Boolean function:

$$ f_i(x, y, z, w) = a_0 + a_1 x + a_2 y + a_3 z + a_4 w + a_{12} x y + a_{13} x z + a_{14} x w + a_{23} y z + a_{24} y w + a_{34} z w $$

(10)

As discussed in [20], we choose one decomposition showed by Table 2. Let $F(w, z, y, x) = (f_3, f_2, f_1, f_0)$, $H(w, z, y, x) = (h_3, h_2, h_1, h_0)$, where $w$ denotes the most significant bit and $x$ donotes the least significant bit of input, meanwhile, $f_3$, $h_3$ denote the most significant bit of output. The ANFs for both the quadratic functions are as below:

$$ F(w, z, y, x) = (f_3, f_2, f_1, f_0) $$

$$ f_0 = x + y + xy + z + w $$

$$ f_1 = y + xz $$

$$ f_2 = 1 + z $$

$$ f_3 = x + y + z $$

$$ H(w, z, y, x) = (h_3, h_2, h_1, h_0) $$

$$ h_0 = 1 + x $$

$$ h_1 = x + y $$

$$ h_2 = 1 + y + z + w + xw $$

$$ h_3 = xy + w $$

After rewriting the expression of $S_{\text{box}}$, we need to update the algebraic equation. We define 64 new variables in each round to represent the middle value of the S-box. Let $x_{63} \parallel x_{62} \parallel \cdots \parallel x_1 \parallel x_0$ be the input of $S_{\text{box}}$, $y_{63} \parallel y_{62} \parallel \cdots \parallel y_1 \parallel y_0$ be the output of the first quadratic $S_{\text{box}}$, $z_{63} \parallel z_{62} \parallel \cdots \parallel z_1 \parallel z_0$ be the output of the second quadratic $S_{\text{box}}$ while $z_{63} \parallel z_{62} \parallel \cdots \parallel z_1 \parallel z_0$ is also the output of the original cubic S-box. The relationship between the S-box input and the output of the first quadratic S-box is shown below, where $0 \leq r \leq 27, 0 \leq i \leq 15$.

$$ y_{4i+1} = x_{4i} + x_{4i+1} + x_{4i+2} + x_{4i+3} + x_{4i} x_{4i+1} $$

$$ y_{4i+2} = 1 + x_{4i+2} $$

$$ y_{4i+3} = x_{4i} + x_{4i+1} + x_{4i+2} x_{4i+1} $$

The relationship between the output of the first quadratic $S_{\text{box}}$ and the output of the second quadratic $S_{\text{box}}$ is shown below, where $0 \leq r \leq 27, 0 \leq i \leq 15$.

$$ z_{4i+1} = 1 + y_{4i} $$

$$ z_{4i+2} = 1 + y_{4i+1} + y_{4i+2} + y_{4i+3} + y_{4i} y_{4i+3} $$

$$ z_{4i+3} = y_{4i+3} + y_{4i+1} $$

(14)

Convert all equations into CNF statements suitable for the Solver. According to the algebraic representation, each round of new SubCells can be represented with 208 variables and 368 CNF equations, other operations can be represented as before, so each round can be represented with 374 variables and 528 CNF equations.

By comparison, it can be found that rewriting $S_{\text{box}}$ adds intermediate variables to the whole encryption process. Meanwhile, since the number of newly added variables after rewriting $S_{\text{box}}$ is smaller than that before rewriting, the number of CNF equations decreases.

Table 3 gives the comparison of the number of cubic and quadratic variables in the sets of algebraic equations before and after the $S_{\text{box}}$ decomposition. By using the method
of S_box decomposition, we transform the unique nonlinear operation S_box in the GIFT_64 block cipher from one cubic S_box to two quadratic S_boxes, two cubic variables are subtracted, and the total number of quadratic and cubic variables introduced by a single S_box is reduced from 6 to 5.

In order to compare the influence of the two methods on the solving speed, 14, 15 and 16 faults were randomly injected into different S_boxes in 25-round with 50 fault samples. (In the full text, method1 represents before decomposition and method2 represents after decomposition.) The solving time was set as 1 hour, if the solution is not completed within a specified time, the attack is considered to have failed. It is worth saying that the two methods adopt the same batch of fault samples. Figure 4 gives the distribution of the solution time for different scenarios. When the number of faults is 16, the average solution time decreases from 14.4s to 3.1s. When the number of faults is 15, the average solution time decreases from 14.6 to 11.7s. When the number of faults is 14, the average solution time decreases from 27.2s to 21.8s. The decrease in solution time slows down with the decrease of fault samples. Figure 14, the average solution time decreases from 27.2s to 21.8s. when the number of faults is within a specified time, the attack is considered to have failed. To explore the impact of S_box decomposition on other block ciphers, we next analyze and experiment with the skinny_64.

4 Application to Skinny_64

4.1 The structure of Skinny_64 cipher

Skinny is a new lightweight block cipher designed by Beierle et al. at CRYPTO 2016. SKINNY is based on the Substitution-Permutation-Network approach, which is similar with GIFT block cipher. The lightweight block ciphers of the SKINNY family have 64-bit and 128-bit block versions, in this work, we focus only on the 64-bit version. The structure of the SKINNY_64 block cipher is given in Figure 5.

- Initialization
  The cipher state S is first initialized from the 64-bit plaintext represented as 16 nibbles of 4-bits represented as $w_0 \parallel w_1 \parallel \cdots \parallel w_{15}$. This is the initial value of the cipher internal state and note that the state is loaded row-wise.

- The round function Each round of the cipher comprises of SubCells, AddConstants, AddRoundTweakey, ShiftRows, and MixColumns.
- SubCells The S_box is applied to every nibble of the cipher state. Table 4 gives the action of this S_box in hexadecimal notation.

### Table 3 Numbers of cubic and quadratic variables under different methods for GIFT_64

<table>
<thead>
<tr>
<th>method</th>
<th>$N_1$(cubic variable)</th>
<th>$N_2$(quadratic variable)</th>
<th>N(total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>3+2</td>
<td>5</td>
</tr>
</tbody>
</table>

### Table 4 The S_box of Skinny_64

| $s$ | c | 6 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| S[x]|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

In the 64-bit version, the cipher receives a tweakey input $K = k_0 \parallel k_1 \parallel \cdots \parallel k_{63}$.
$$Y(x_3, x_2, x_1, x_0) = (y_3, y_2, y_1, y_0)$$

$$y_0 = x_1 + x_2 + x_3 + x_0 x_1 + x_0 x_2 + x_0 x_3 + x_1 x_3 + x_0 x_1 x_2 + x_1 x_2 x_3$$

$$y_1 = x_0 + x_3 + x_0 x_1 + x_1 x_2 + x_2 x_3 + x_0 x_2 x_3$$

$$y_2 = 1 + x_1 + x_2 + x_3 + x_1 x_2$$

$$y_3 = 1 + x_0 + x_2 + x_3 + x_2 x_3$$

(15)

**AddContants**

The round constants are generated by 6-bit affine LFSR, whose update function is defined as:

$$(c_5, c_4, c_3, c_2, c_1, c_0) \leftarrow (c_4, c_3, c_2, c_1, c_5 \oplus c_4 \oplus 1)$$

The six bits are initialized to zero, and updated before use in given round. The bits from the LFSR are arranged into a $4 \times 4$ array:

$$\begin{bmatrix}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}$$

(16)

with $c_2 = 0 \times 2$, $(c_0, c_1) = (rc_0 || rc_1 || rc_1 || rc_0, 0 || 0 || rc_5 || rc_5)$.

**AddRoundTweakey**

The first and second rows of all tweakey arrays are extracted and bitwise exclusive-ored to the cipher internal state, respecting the array positioning. The specific subkey generation method can be found in [3].

**ShiftRows**

This operation can be represented as a permutation. A permutation $P$ is applied on the cells positions of the cipher internal state cell array: for all $0 \leq i \leq 15$, the operation can be shown as:

$$P = [0, 1, 2, 3, 4, 5, 6, 10, 11, 8, 9, 13, 14, 15, 12]$$

**MixColumns**

This operation can mix each column by multiplication. The matrix M of the multiplication is showed as follow:

$$\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}$$

(17)

The method of building the algebraic equation set for SKINNY-64 is similar with GIFT-64. Figure 4 shows the spread of single-bit or nibble fault injected into $R$-th round. The figure 6 shows the spread of a single fault in the operation of four rounds. It can be concluded that when the fault injection is too shallow, the information of the subkey is too small to recover, but when the fault injection is too deep, the information redundancy increases and the algebraic equation is difficult to solve. So, in this study, we inject faults in different rounds and compare the average solution time and solution success rate.

4.2 The $S$-box decomposition of Skinny-64 cipher

The cubic $S$-box can be decomposed into the following two quadratic $S$-boxes using the same method as above. Table 5 gives the lookup table of the SKINNY-64 after $S$-box decomposition. The translation of the two look-up tables into the form of algebraic equations can be expressed by equation (18) and (19).

$$F(w, z, y, x) = (f_3, f_2, f_1, f_0)$$

$$f_0 = x + z + zw + w$$

$$f_1 = y$$

$$f_2 = y + yz + z + w$$

$$f_3 = z$$

(18)

$$H(w, z, y, x) = (h_3, h_2, h_1, h_0)$$

$$h_0 = y + xz$$

$$h_1 = x + xy + w$$

$$h_2 = 1 + z$$

$$h_3 = 1 + x$$

(19)

As shown in Table 6, by decomposing the origin $S$-box,
the number of the variable can be reduced from 8 to 4. We can use smaller variables to express the relationship between S-box input and output, so that the number of variables in the whole CNF equation decreases and the total number of equations decreases.

In order to compare the influence of the two different methods on the solving speed, we randomly injected the specified number of faults at a given location. In the same way, the two methods adopt the same batch of samples. To avoid contingency, the number of samples was set to 100. And it’s worth saying that the solving time was set as 1 hour, if the solution is not completed within the specified time, the attack is considered as failure.

By comparing the results of the above experiments, we can find that in the case of the same batch of samples, using different algebraic equations to represent S-box operations has a great impact on the solution time. By decomposing the S-box, the number of the faults required for key-recovery is reduced. The experimental comparison chart is shown in Table 7. Under the same condition of fault-injection samples, the solving speed of method 2 is much faster than method 1. The best experimental result is that when the round of fault injection is 30 and the number of faults is 12, the average solution time is reduced from 322.3s to 0.4s, which is about 806 times more efficient. Meanwhile, in the specific time, the solving success rate of method 2 is always 100%, while the solving success rate of method 1 decreases with the decrease of the number of faults. The best experimental result is that when the round of fault injection is 28 and the number of faults is 12, the success rate increases from 27% to 100%. The experimental results have all illustrated the effectiveness of this method of S-box decomposing for skinny block cipher’s algebraic fault analysis, and it has good superiority compared with the traditional algebraic fault analysis methods.

In order to obtain the optimal round of fault injection and the minimum number of faults for algebraic fault analysis of skinny block cipher, the following experiments are conducted in this paper. With the same setting background as before, this experiment injected a single fault multiple times on the premise of a fixed plaintext. According to the diffusion diagram, we can see that the fault will affect all bytes after four rounds of diffusion. However, with the increase of fault depth, the redundancy of fault information will increase, and the time to solve the fault will increase. However, when the fault depth is shallow, less effective fault information is available and more faults are required. The correct choice of the round of fault-injection will reduce the need for the number of faults. Extensive experiments were conducted from two perspectives: the number of fault injection rounds and the number of faults. In this experiment, we chose to perform fault injection in rounds 30, 29, 28 and 27. The specific solution results are shown in Table 8.

From the above experiments, we can obtain the minimum number of faults required to complete the key-recovery for all samples within one hour of the specified time under different fault injection rounds. When the fault is injected to the 30th round, the required number of faults is 10. When the fault is injected into the 29th round, the required number of faults is 4. When faults are injected into round 28, the required number of faults is 2. When the fault is injected into the 27th round, the required number of faults is 6. The specific time distribution of these is shown in Figure 7. Therefore, the optimal round of fault injection the 28th round, and the solution of all keys can be completed in an average time of 27.6s using two random single-bit faults (injected into

---

**Table 6 Numbers of cubic and quadratic variables under different methods for SKINNY**

<table>
<thead>
<tr>
<th>Method</th>
<th>( N_1 ) (cubic variable)</th>
<th>( N_2 ) (quadratic variable)</th>
<th>( N ) (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>2+2</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 7 Comparison of the average solution time and success rate of the two methods in different scenarios for SKINNY**

<table>
<thead>
<tr>
<th>Round</th>
<th>( N )</th>
<th>( T_{ave1} ) (s)</th>
<th>( T_{ave2} ) (s)</th>
<th>Success Rate1</th>
<th>Success Rate2</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>16</td>
<td>182.7</td>
<td>0.5</td>
<td>98%</td>
<td>100%</td>
</tr>
<tr>
<td>30</td>
<td>14</td>
<td>251.0</td>
<td>0.4</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>30</td>
<td>12</td>
<td>322.3</td>
<td>0.4</td>
<td>89%</td>
<td>100%</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>404.5</td>
<td>1.2</td>
<td>74%</td>
<td>100%</td>
</tr>
<tr>
<td>29</td>
<td>12</td>
<td>286.0</td>
<td>3.2</td>
<td>93%</td>
<td>100%</td>
</tr>
<tr>
<td>29</td>
<td>8</td>
<td>687.9</td>
<td>2.9</td>
<td>43%</td>
<td>100%</td>
</tr>
<tr>
<td>28</td>
<td>12</td>
<td>1068.2</td>
<td>39.7</td>
<td>27%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Fig. 7 Distribution of solving time under different fault injection scenarios for SKINNY.**

<table>
<thead>
<tr>
<th>Scene</th>
<th>Time Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td></td>
</tr>
</tbody>
</table>
different $S$ boxes). For the SKINNY block cipher, the use of $S$ box decomposition substantially reduces the number of faults required.

### 5 Application to PRESENT 64 block cipher

In order to make a comparative experiment, this paper also studies the classical block cipher of PRESENT. The structure of PRESENT is not the focus of this paper, so it will not be described here. Please refer to the literature for details. The PRESENT 64 block cipher is similar to the GIFT 64, which contains unique nonlinear component Subcells in all encryption operations. Its $S$ box lookup table is shown in Table 9. The literature also gives an algebraic expression for the $S$ box, as shown in Equation 20.

$$F(x_3,x_2,x_1,x_0) = (y_3,y_2,y_1,y_0)$$

\[
\begin{align*}
y_0 &= x_0 + x_2 + x_3 + x_1 x_2 \\
y_1 &= x_1 + x_3 + x_1 x_3 + x_2 x_3 + x_0 x_1 x_2 + x_0 x_1 x_3 \\
y_2 &= 1 + x_2 + x_3 + x_0 x_1 + x_0 x_3 + x_1 x_3 + x_0 x_1 x_3 \\
y_3 &= 1 + x_0 + x_1 + x_3 + x_1 x_2 + x_0 x_1 x_2 + x_0 x_1 x_3
\end{align*}
\]  

The cubic $S$ box can be decomposed into the following two quadratic $S$ boxes by the same method mentioned above. The lookup table of the decomposed $S$ boxes is shown in Table 10. Next we perform an algebraic equation repre-

<table>
<thead>
<tr>
<th>Round</th>
<th>Number</th>
<th>$T_{ave2}(s)$</th>
<th>Success Rate of Method2</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>8</td>
<td>29.6</td>
<td>98%</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>322.6</td>
<td>84%</td>
</tr>
<tr>
<td>29</td>
<td>6</td>
<td>2.2</td>
<td>100%</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>5.2</td>
<td>100%</td>
</tr>
<tr>
<td>29</td>
<td>3</td>
<td>155.7</td>
<td>86%</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>421.6</td>
<td>47%</td>
</tr>
<tr>
<td>28</td>
<td>8</td>
<td>13.2</td>
<td>100%</td>
</tr>
<tr>
<td>28</td>
<td>6</td>
<td>16.2</td>
<td>100%</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>12.6</td>
<td>100%</td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>14.0</td>
<td>100%</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>27.6</td>
<td>100%</td>
</tr>
<tr>
<td>27</td>
<td>6</td>
<td>237.8</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Table 8** Comparison of the average solution time and success rate of the two methods in different scenarios for SKINNY 64

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S[x]$</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td>a</td>
<td>d</td>
<td>3</td>
<td>e</td>
<td>f</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Table 9** The $S$ box of PRESENT 64

For the SKINNY block cipher, the use of $S$ box decomposition substantially reduces the number of faults required.

In order to make a comparative experiment, this paper also studies the classical block cipher of PRESENT. The structure of PRESENT is not the focus of this paper, so it will not be described here. Please refer to the literature for details. The PRESENT 64 block cipher is similar to the GIFT 64, which contains unique nonlinear component Subcells in all encryption operations. Its $S$ box lookup table is shown in Table 9. The literature also gives an algebraic expression for the $S$ box, as shown in Equation 20.

\[
F(w,z,y,x) = (f_3,f_2,f_1,f_0) \\
f_0 = y + z + w \\
f_1 = 1 + y + z \\
f_2 = 1 + x + z + yw + zw \\
f_3 = 1 + w + xy + xz + yz
\]  

\[
H(w,z,y,x) = (h_3,h_2,h_1,h_0) \\
h_0 = y + z + w + zw \\
h_1 = x + zw \\
h_2 = y + z + xw \\
h_3 = z + yw
\]  

It can be seen from Table 11 that the number of variables adopted has not changed by decomposing one cubic $S$ box to two quadratic $S$ boxes. Meanwhile, the number of variables introduced into the total algebraic fault equation increases due to the addition of 64-bit intermediate variables as outputs of the first quadratic $S$ box in each round.

The two methods used in this paper are compared and the experimental results are shown as Table 12. Four sets of comparison experiments were done in this study. The experiments were set with a maximum solution time of 1 hour, and failure was adjudged if the time exceeded 1 hour. 50 samples were randomly generated for each group of experiments. When the number of fault injection rounds is 28 and the number of faults is 5, the average solving time is reduced from 563.7s to 4.1s by $S$ box decomposition, which is 137 times faster, and the success rate of solution within the specified time is increased from 90% to 100%. When the number of fault injection rounds is 28 and the number of faults is 5, the average solving time is reduced from 563.7s to 4.1s by $S$ box decomposition, which is 137 times faster, and the success rate of solution within the specified time is increased from 90% to 100%.
Decomposition of higher-order nonlinear S-boxes in lightweight block ciphers for algebraic fault analysis

6 Conclusion and future work

This paper proposes a new S-box algebraic representation for algebraic fault analysis on lightweight block ciphers. By using the proposed S-box decomposition method, the efficiency and success rate of some ciphers are greatly improved. The master key can be recovered with fewer faults and shorter solution time, so that the algebraic fault attack experiment can be completed under the premise of limited resources.

In this paper, three lightweight block ciphers are studied based on algebraic fault analysis of S-box decomposition. The resolution results of these three ciphers are different, which better illustrates the applicable scenarios of this method. First, we conducted an algebraic fault analysis on GIFT\textsubscript{64} proposed in 2017. By comparing the two methods of original S-box and decomposed S-box, we found that under the same batch of samples, the solution time of decomposed S-box was improved. To find other suitable environments for this method, we analyzed the SKINNY and PRESENT ciphers. The results show that when the S-box decomposition method is used, if the total number of variables introduced after the S-box decomposition is less than before the S-box decomposition, the fault solving speed will be improved. For SKINNY cipher, the number of intermediate variables introduced for S-box representation decreases significantly after S-box decomposition, so the number of faults to key-recovery and the solving time will also decrease significantly. On the contrary, for PRESENT cipher, this method can’t reduce the number of intermediate variables required to express S-box algebraic equations. The results show that when the number of fault rounds is deep, decomposing the S-box can also serve to improve the solving efficiency.

Future work can be derived in algebraic persistent fault analysis. In the latest study, Zhang Fan et al. proposed an algebraic persistent fault analysis method. They analyzed the skinny block cipher in the literature. If the method of S-box decomposition is adopted, I believe that the solving speed will be improved to a certain extent. Research on this content is in progress, and experimental results will be given in the future.

References


Statements and Declarations

Funding

This work was supported by the National Natural Science Foundation of China(62071057),the Fundamental Research Funds for the Central Universities(2019XD17),Aeronautical Science Foundation of China(2019ZG073001)

Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

Author Contributions

All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by [Xing Fang]. The first draft of the manuscript was written by Xing Fang and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Declarations

Data Availability

It is available from the corresponding author on reasonable request.