Hindawi
Security and Communication Networks
Volume 2017, Article ID 8051728, 15 pages
https://doi.org/10.1155/2017/8051728

Research Article

Stealthy Hardware Trojan Based Algebraic Fault Analysis of HIGHT Block Cipher

Hao Chen,[1,2] Tao Wang,[1] Fan Zhang,[2,3,4] Xinjie Zhao,[5] Wei He,[6] Lumin Xu,[2] and Yunfei Ma[1]

[1] *Department of Information Engineering, Ordnance Engineering College, Shijiazhuang 050003, China*
[2] *Science and Technology on Communication Security Laboratory, Chengdu 610041, China*
[3] *Zhejiang University, Yuquan Campus, Hangzhou 310027, China*
[4] *School of Computing, National University of Singapore, Singapore 117417*
[5] *Institute of North Electronic Equipment, Beijing 100191, China*
[6] *Central Research Institute, Huawei Pte Ltd., Singapore 117674*

Correspondence should be addressed to Fan Zhang; fanzhang@zju.edu.cn

Received 17 May 2017; Accepted 16 October 2017; Published 20 December 2017

Academic Editor: Namje Park

HIGHT is a lightweight block cipher which has been adopted as a standard block cipher. In this paper, we present a bit-level algebraic fault analysis (AFA) of HIGHT, where the faults are perturbed by a stealthy HT. The fault model in our attack assumes that the adversary is able to insert a HT that flips a specific bit of a certain intermediate word of the cipher once the HT is activated. The HT is realized by merely 4 registers and with an extremely low activation rate of about 0.000025. We show that the optimal location for inserting the designed HT can be efficiently determined by AFA in advance. Finally, a method is proposed to represent the cipher and the injected faults with a merged set of algebraic equations and the master key can be recovered by solving the merged equation system with an SAT solver. Our attack, which fully recovers the secret master key of the cipher in 12572.26 seconds, requires three times of activation on the designed HT. To the best of our knowledge, this is the first Trojan attack on HIGHT.

1. Introduction

The resource-constrained devices such as RFID tags and smart cards have been pervasively used in the daily activities of human society, such as intelligent transportation, modern logistics, and food safety [1, 2]. As these devices have inherent constrains in storage space, computation ability, and power supply, modern cryptographic primitives like DES, AES, or RSA are difficult to be deployed on them. Hence, the research of lightweight cryptography, which aims at designing and implementing security primitives fitting the needs of low-resource devices, has been focused on a large scale [3]. Particularly, the lightweight block cipher is one of the most studied metrics, which has been extensively explored in numerous prior papers. There have existed a lot of lightweight block ciphers, such as PRESENT [4], LED [5], SIMON [6], mCrypton [7], and HIGHT [8, 9].

Hardware Trojan is a circuit maliciously inserted into integrated circuit (IC) that typically functions to deactivate the host circuit, change its functionality, or provide covert channels through which sensitive information can be leaked [10, 11]. They can be implemented as hardware modifications to ASICs, commercial-off-the-shelf (COTS) parts, microprocessors, microcontrollers, network processors, or digital-signal processors (DSPs) and can also be implemented as firmware modifications to, for example, FPGA bitstreams [12]. An adversary is expected to make a Trojan stealthy in nature, that is, to evade detection by methods such as postmanufacturing test, optical inspection, or side-channel analysis [13–15]. Due to outsourcing trend of the semiconductor design and fabrication, hardware Trojan attacks have emerged as a major security concern for integrated circuits (ICs) [13].

Differential Fault Analysis (DFA) [16] was one of the earliest techniques invented to attack block ciphers by provoking a computational error. DFA retrieves the secret key based on information of the characteristics of the injected faults and the difference of the ciphertexts and faulty ciphertexts.

However, since DFA relies on manual analysis, it often has inherit inherent limitations in scenarios that have very high complexity, for example, when faults are located in deeper rounds of the cipher or when the exact location of the injected faults in a deep round is unknown.

In eSmart 2010, Courtois and Pieprzyk combine algebraic cryptanalysis [17] with fault analysis to propose a more powerful fault analysis technique called algebraic fault analysis (AFA) [18]. The basic idea of AFA is to convert both the cipher and the injected faults into algebraic equations and recover the secret key with automated solvers such as SAT instead of the manual analysis on fault propagations in DFA, hence making it easier to extend AFA to deep rounds and different ciphers and fault models. AFA has been successfully used to improve DFA on the stream ciphers such as Trivium [19] and Grain [20] and block ciphers such as AES [21], LED [22, 23], KASUMI [24], and Piccolo [25].

1.1. Motivation. HIGHT is a lightweight block cipher that has attracted a lot of attention because it is constructed by only ARX operations (modular addition, bitwise rotation, bitwise shift, and XOR), which exhibits high performance in terms of hardware compared to other block ciphers. HIGHT has been selected as a standardized block cipher by Telecommunications Technology Association (TTA) of Korea and ISO/IEC 18033-3 [9].

It is noted that both the DFA and AFA require high precision in the fault injection in terms of location and timing. In practice, low-cost fault injection techniques like reduction of the feeding voltage or clock manipulation do not achieve the required accuracy, while highly precise methods such as pinpointed irradiation of desired fault sites by intensive laser light are difficult to perform and require costly equipment [26]. However, if the adversary is able to insert hardware Trojan (HT) to the underlying cryptographic hardware [10], AFA can be easily achieved. A well designed HT can precisely inject any type of faults to enable AFA and evade detections, by having low cost and with low activation rate.

In addition, since the design of lightweight block ciphers is compact, especially for HIGHT whose construction only based on ARX operations, it is simple to represent the cipher as a set of algebraic equations. It is also easier to implant hardware Trojans into devices that adopt such lightweight algorithms because these devices are normally used in RFID system and composed of sorts of IPs, and they are typically designed and manufactured by offshore design houses or foundries. In theory, any parties involving into the design or manufacturing stages can make alterations in the circuits for malicious purpose [15], and thus these circuits are more vulnerable to algebraic fault attacks which inject faults by triggering HT.

1.2. Contribution. In this paper, we show that the lightweight block cipher HIGHT is prone to algebraic fault analysis, which can be feasible with a stealthy HT. The proposed analysis of HIGHT is implemented on SASEBO-GII board soldering a 65 nm Virtex-5 FPGA [27] and recovers the 128-bit secret master key with only 3 faults. The main contributions of the paper are summarized as follows:

(1) We design a stealthy FSM-based HT by using 4 flip-flops overhead which is a 1.63% additional cost in flip-flops for HIGHT implemented on SASEBO-GII board and with an extremely low activation rate of about 0.000025. The HT enables the adversary to induce a single-bit fault precisely in both location and time when it is activated and thus make the bit-level AFA efficiently.

(2) Some properties of faults are given to maximize the utilization of the fault leakages and show that the adversary can predetermine the optimal location for the HT by AFA to maximize the attack efficiency.

(3) A very simple and efficient method is proposed to describe HIGHT and the injected faults as a merged set of algebraic equations and transform the problem of searching for the secret master key into solving the merged equation system with an SAT solver.

(4) It is proven that the lower bound for the number of the required faults is 3 and an efficient distinguisher is proposed to uniquely determine the secret master key.

1.3. Organization. The rest of this paper is organized as follows. Section 2 introduces the related works. Section 3 lists the notations used in the paper and briefly describes the HIGHT algorithm and the overview of the attack. Section 4 presents some important properties of the faults and the details of the HT are given in Section 5. Then, Section 6 describes our attack on HIGHT and the experimental results are shown in Section 7. Finally, Section 8 concludes the paper.

2. Related Work

Since the proposal of HIGHT, there have been many studies on the security of HIGHT. The preliminary security analysis [8], conducted during the HIGHT design process, includes the assessment of the cipher with respect to different cryptanalytic attacks such as differential cryptanalysis, related-key attack, saturation attack, and algebraic attack and the designers claim that at least 20 rounds of HIGHT are secure against these attacks. But in 2007, Lu [28] presents the first public cryptanalysis of reduced versions of HIGHT which indicates the reduced versions of HIGHT are less secure than the designers claimed. Then in 2009, Lu's attack results were improved by Özen et al. [29] by presenting an impossible differential attack on 26-round HIGHT and a related-key impossible differential attack on 31 round HIGHT. At CANS 2009, Zhang et al. [30] present a 22-round saturation attack on HIGHT including full whitening keys with $2^{62.04}$ chosen plaintext and $2^{118.71}$ 22-round encryptions. The first attack on full HIGHT was proposed by Koo et al. at ICISC 2010 [31] using related-key rectangle attack based on a 24-round related-key distinguisher with the data complexity of $2^{57.84}$ chosen plaintext and the time complexity of $2^{123.17}$ encryptions. The second attack on full HIGHT was proposed by Hong et al. at ICISC 2011 [32] with a Biclique cryptanalysis of the full HIGHT which recovers the 128-bit secret master key with the computational complexity of $2^{126.4}$, faster than exhaustive search. In [33], Lee et al. present the first DFA against HIGHT. In this attack, authors claimed that the full secret master key of HIGHT can be recovered in a few

Table 1: Summary of the attacks on HIGHT.

Attack	#rounds	Complexities		References
		Data	Time	
Imp. Diff.	18	$2^{46.8}$ CP	$2^{109.2}$ EN	[8]
Saturation	22	$2^{62.04}$ CP	$2^{118.71}$ EN	[30]
Imp. Diff.	25	2^{60} CP	$2^{126.78}$ EN	[29]
Imp. Diff.	26	2^{61} CP	$2^{119.53}$ EN	[22]
Rel.-Key Rec.	26	$2^{51.2}$ CP	$2^{120.41}$ EN	[28]
Rel.-Key Imp.	28	2^{60} CP	$2^{125.54}$ EN	[28]
Rel.-Key Imp.	31	2^{63} CP	$2^{127.28}$ EN	[29]
Rel.-Key Rec. for Weak.	*32 (full)*	$2^{57.84}$ *CP*	$2^{123.17}$ *EN*	[31]
Biclique	*32 (full)*	2^{48} *CP*	$2^{126.4}$ *EN*	*[32]*
DFA	*32 (full)*	*12 faults*	$O(2^{32})$ *Com.* + $O(2^{32})$ *Mem.* +22 *seconds*	*[33]*
AFA with HT	*32 (full)*	*3 faults*	*12572.26 seconds (≈3.49 Hours)*	*This paper*

Imp.: impossible, Diff.: differential, Rel.: related, Rec.: rectangle, Weak.: weak key, CP: chosen plaintext, EN: encryptions, Com.: computational, and Mem.: memory.

minutes or seconds with a success rate of 96%, computational complexity of $O(2^{32})$, and memory complexity of $O(2^{12})$ by injecting 12 faults based on a random byte fault model.

The main idea of this attack is to collect pairs of correct and faulty ciphertexts by injecting adequate faults and use them to distinguish where the faults are injected. Once the fault locations are determined, a number of equations can be built based on manual analysis of the fault propagations to filter out the wrong subkey candidates and thus to recover the secret master key. However, since the adversary analyzes fault propagations and filters out wrong subkey candidates manually, the fault leakages are not maximally utilized and the attack can be further improved.

In this paper, we elaborate an algebraic fault analysis of HIGHT with a stealthy HT. The fault model we choose in this attack is the one in which the adversary is assumed to inject a single-bit fault precisely in both location and the time of the disturbance by a HT which is activated just by choosing certain plaintexts. The attack converts both the cipher and the injected faults into algebraic equations automatically and recovers the secret master key with an SAT solver. The attack recovers the secret master key with a success rate of 96% within 12,572.26 seconds and requires only 3 faults. We summarize our results as well as the major previous results in Table 1.

3. Preliminaries

In this section, the notations used in the paper are listed in Section 3.1. Then, we briefly describe the HIGHT algorithm in Section 3.2 and the overview of the attack is given in Section 3.3.

3.1. Notations. In the rest of the paper, the following notations are used:

1. $\boxplus$: $x \boxplus y$ means $x + y \bmod 2^8$, where $0 \le x, y < 2^8$.
2. $\oplus$, $\|$: bitwise XOR and concatenation operations.
3. $A^{\lll s}$: s-bit left rotation of an 8-bit value A.
4. $'$: sign for denoting faulty ciphertext or intermediate values.
5. P, C, C': the 64-bit plaintext, ciphertext, and faulty ciphertext.
6. $\mathrm{MK} = \mathrm{MK}_{15} \| \cdots \mathrm{MK}_1 \| \mathrm{MK}_0$: the 16 bytes master key.
7. WK_i: the whitening keys, $0 \le i \le 7$.
8. SK_k: the round keys, $0 \le k \le 127$.
9. $X_r = X_{r,7} \| \cdots X_{r,1} \| X_{r,0}$: the 64-bit input of the $(r+1)$th round, $0 \le r \le 32$.
10. $X_{r,i}^k$: the kth bit of $X_{r,i}^k$, $0 \le k \le 7$.

3.2. Brief Description of HIGHT Cipher. HIGHT is a lightweight block cipher with 64-bit block length and 128-bit key length. The encryption process of HIGHT is as follows.

(1) The *KeySchedule* is performed to generate 8 bytes whitening keys WK_i $(0 \le i \le 7)$ and 128 bytes SK_i $(0 \le i \le 127)$:

$$\mathrm{WK}_i = \mathrm{MK}_{i+12} \quad (i = 0, 1, 2, 3); \tag{1}$$

$$\mathrm{WK}_i = \mathrm{MK}_{i-4} \quad (i = 4, 5, 6, 7); \tag{2}$$

$$\mathrm{SK}_{16i+j} = \mathrm{MK}_{j-i \bmod 8} \boxplus \delta_{16i+j} \quad (0 \le i, j \le 7), \tag{3}$$

$$\mathrm{SK}_{16i+j+8} = \mathrm{MK}_{(j-i \bmod 8)+8} \boxplus \delta_{16i+j+8} \quad (0 \le i, j \le 7). \tag{4}$$

(2) The *InitialTransformation* is performed to transform the 64-bit plaintext P to the input of the first round X_0 by using four bytes whitening keys WK_0, WK_1, WK_2, and WK_3.

$$X_{0,0} = P_0 \boxplus \mathrm{WK}_0;$$

$$X_{0,1} = P_1;$$

$$X_{0,2} = P_2 \oplus \mathrm{WK}_1;$$

$$X_{0,3} = P_3;$$

$$X_{0,4} = P_4 \boxplus \mathrm{WK}_2;$$

$$
\begin{aligned}
X_{0,5} &= P_5;\\
X_{0,6} &= P_6 \oplus \mathrm{WK}_3;\\
X_{0,7} &= P_7.
\end{aligned}
\tag{5}
$$

(3) For $i = 1, 2, \ldots, 31$, *RoundFunction* is performed to transform X_i into X_{i+1} as follows:

$$
\begin{aligned}
X_{i,0} &= X_{i-1,7} \oplus \left(F_0\left(X_{i,6}\right) \boxplus \mathrm{SK}_{4i-1}\right);\\
X_{i,2} &= X_{i-1,1} \boxplus \left(F_0\left(X_{i,0}\right) \oplus \mathrm{SK}_{4i-4}\right);\\
X_{i,4} &= X_{i-1,3} \oplus \left(F_0\left(X_{i,2}\right) \boxplus \mathrm{SK}_{4i-3}\right);\\
X_{i,6} &= X_{i-1,5} \boxplus \left(F_0\left(X_{i,4}\right) \oplus \mathrm{SK}_{4i-2}\right);\\
X_{i,1} &= X_{i-1,0};\\
X_{i,3} &= X_{i-1,2};\\
X_{i,5} &= X_{i-1,4};\\
X_{i,7} &= X_{i-1,6}.
\end{aligned}
\tag{6}
$$

For $i = 32$,

$$
\begin{aligned}
X_{i,1} &= X_{i-1,1} \boxplus \left(F_1\left(X_{i-1,0}\right) \oplus \mathrm{SK}_{4i-4}\right);\\
X_{i,3} &= X_{i-1,3} \oplus \left(F_0\left(X_{i-1,2}\right) \boxplus \mathrm{SK}_{4i-3}\right);\\
X_{i,5} &= X_{i-1,5} \boxplus \left(F_1\left(X_{i-1,4}\right) \boxplus \mathrm{SK}_{4i-2}\right);\\
X_{i,7} &= X_{i-1,7} \oplus \left(F_0\left(X_{i-1,6}\right) \boxplus \mathrm{SK}_{4i-1}\right);\\
X_{i,0} &= X_{i-1,0};\\
X_{i,2} &= X_{i-1,2};\\
X_{i,4} &= X_{i-1,4};\\
X_{i,6} &= X_{i-1,6}.
\end{aligned}
\tag{7}
$$

The two auxiliary functions F_0 and F_1 are defined as follows:

$$
\begin{aligned}
F_0(x) &= \left(x^{\lll 1}\right) \oplus \left(x^{\lll 2}\right) \oplus \left(x^{\lll 7}\right),\\
F_1(x) &= \left(x^{\lll 3}\right) \oplus \left(x^{\lll 4}\right) \oplus \left(x^{\lll 6}\right).
\end{aligned}
\tag{8}
$$

(4) The *FinalTransformation* transforms X_{32} into the ciphertext C:

$$
\begin{aligned}
C_0 &= X_{32,1} \boxplus \mathrm{WK}_4;\\
C_1 &= X_{32,2};\\
C_2 &= X_{32,3} \oplus \mathrm{WK}_5;\\
C_5 &= X_{32,6};\\
C_4 &= X_{32,5} \boxplus \mathrm{WK}_6;\\
C_3 &= X_{32,4};\\
C_6 &= X_{32,7} \oplus \mathrm{WK}_7;\\
C_7 &= X_{32,0}.
\end{aligned}
\tag{9}
$$

For complete description of HIGHT, the reader is referred to [8, 9].

3.3. Overview of the Attack. As illustrated in Figure 1, our attack consists of four steps.

(1) Inducing the Designed HT in a Selected Location. The task of this step is to design a HT and insert it in the cipher chip. The optimal location of inserting a HT should be in a deeper round to enable the fault to involve the whole master key bytes during its propagation. It also should ensure that the injected HT escapes detections by having low cost and with extremely low activation rate.

(2) Constructing Boolean Equations for the Cipher. In this step, the target cipher and its key schedule are described by a set of Boolean equations $\mathscr{C}$, which contain unknowns (master key bits, whitening key bits, subkey bits, and intermediate variables) and constants (plaintext and ciphertext bits). The most important and difficult part in this step for HIGHT is to describe nonlinear operations like addition mod 2^n and complicated linear functions like $F_0(\cdot)$ and $F_1(\cdot)$.

(3) Constructing Boolean Equations for the Faults. After the fault injections, the faults are also represented with a set of Boolean equations $\mathscr{D}$. It is obvious that the more secret variables $\mathscr{D}$ contains, the more master key bits that can be recovered. Therefore, the key point of this step is how to make $\mathscr{D}$ contain secret variables that were involved during the fault propagation as many as possible in an efficient and simple way.

(4) Solving the Algebraic Equation System. The problem of searching for the secret master key is now transformed into solving the merged equation system $\mathscr{C}$ and $\mathscr{D}$. Many automatic tools [25, 34–37] can be leveraged.

4. Some Properties of the Faults

This section is devoted to presenting the fault properties, which are helpful to our attack. For the sake of simplicity, we denote the deduction of $\mathscr{B}$ from $\mathscr{A}$ by equation $(*)$ by

$$
\mathscr{A} \xrightarrow{(*)} \mathscr{B}. \tag{10}
$$

Property 1. Assume that a fault was induced to $X_{r,i}$, then define

$$
\Omega_{r,i} = \left\{\mathrm{WK}_\omega, \mathrm{SK}_\xi \mid X_{r,i} \longrightarrow \mathrm{WK}_\omega, \mathrm{SK}_\xi\right\} \tag{11}
$$

as the set of subkey bytes and whitening key bytes that were involved by the fault during its propagation form round r to *FinalTransformation*, where $1 \le r \le 32$, $0 \le \omega \le 15$, $0 \le \xi \le 127$, $0 \le i \le 7$, and $0 \le m \le 3$. Then we have

$$
\begin{aligned}
&\Omega_{r,i}\\
&= \begin{cases}
\Omega_{r+1,i+1} + \Omega_{r+1,(i+2)\bmod 8} + \left\{\mathrm{SK}_{4r+m}\right\}, & r < 31,\ i = 2m\\
\Omega_{r+1,(i+1)\bmod 8} + \left\{\mathrm{SK}_{4r+m}\right\}, & r < 31,\ i = 2m+1\\
\Omega_{r+1,i} + \Omega_{r+1,i} + \left\{\mathrm{SK}_{4r+m}\right\}, & r = 31,\ i = 2m\\
\Omega_{r+1,i} + \left\{\mathrm{SK}_{4r+m}\right\}, & r = 31,\ i = 2m+1.
\end{cases}
\end{aligned}
\tag{12}
$$

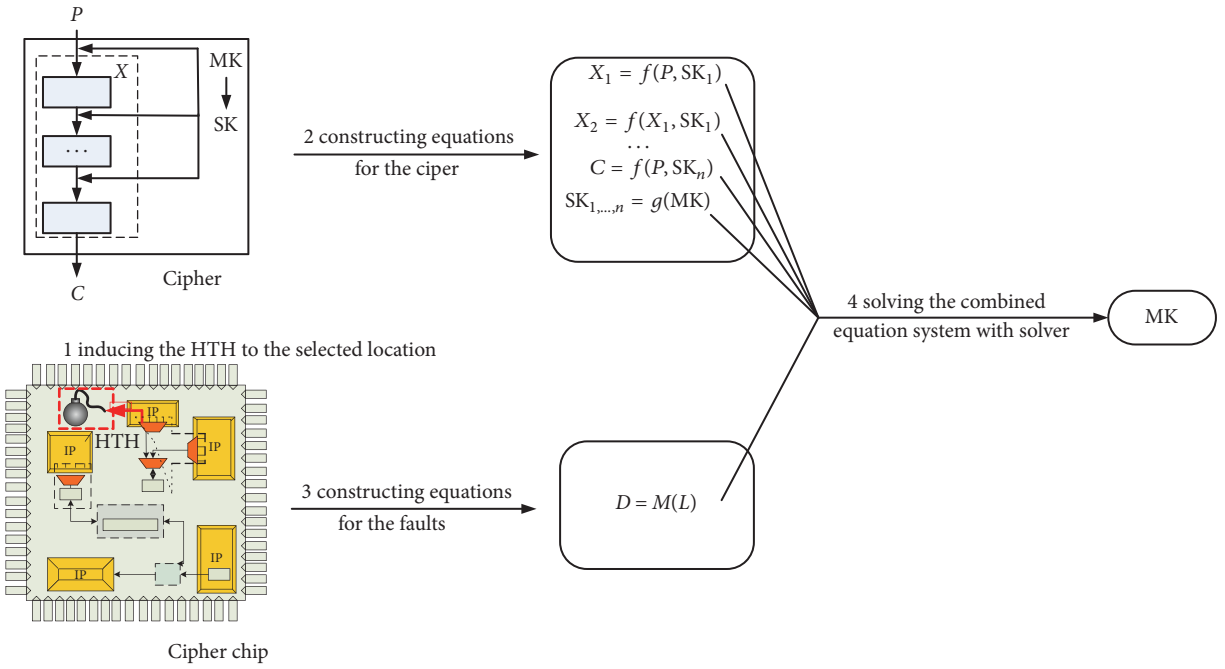

Figure 1: Overview of the attack on HIGHT.

Proof. Without loss of generality, we assume that the fault was induced to $X_{r,i}$.

(1) For $i = 2m$, $1 \le r < 31$, and $0 \le m \le 3$, the fault will propagate to $X_{r+1,2(m+1)\text{mod}8}$ and $X_{r+1,2m+1}$ in the next round as shown in Figure 2; thus we have $\Omega_{r,i} = \Omega_{r+1,i+1} + \Omega_{r+1,(i+2)\text{mod}8} + \{SK_{4r+m}\}$.

When the fault was injected in $r = 31$, the fault will propagate to $X_{r+1,2m\text{mod}8}$ and $X_{r+1,2m+1\text{mod}8}$ in the final round. Then, we have $\Omega_{r+1,i} + \Omega_{r+1,i} + \{SK_{4r+m}\}$.

(2) For $i = 2m+1$, $1 \le r < 31$, and $0 \le m \le 3$, the fault will only propagate to $X_{r+1,2(m+1)\text{mod}8}$ in the next round as shown in Figure 3; thus we have $\Omega_{r,i} = \Omega_{r+1,(i+1)\text{mod}8} + \{SK_{4r+m}\}$.

In the similar way, the fault will propagate to $X_{r+1,2m\text{mod}8}$ and $X_{r+1,2m+1\text{mod}8}$ in the next round for $r = 31$. Then, we have $\Omega_{r,i} = \Omega_{r+1,i} + \{SK_{4r+m}\}$. □

Property 2. Assume that a fault was induced to $X_{r,i}$, then define

$$\Sigma_{r,i} = \{MK_\tau \mid X_{r,i} \longrightarrow MK_\tau, 1 \le r \le 32,\ 0 \le \tau \le 15,\ 0 \le i \le 7\} \tag{13}$$

as the set of the master key bytes that were involved during the propagation of the fault. Then we have the following conclusion.

Proof. From Section 3.2, for the *FinalTransformation* of HIGHT, we have the following formula:

$$X_{32,2m} \xrightarrow{(7)} WK_{m+4}, \quad 0 \le m \le 3. \tag{14}$$

For the *KeySchedule* of HIGHT, we have the following formula:

$$WK_n \xrightarrow{(2)} MK_{n-4}, \quad 4 \le n \le 7. \tag{15}$$

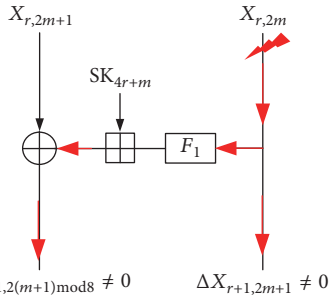

Figure 2: The fault was injected in $X_{r,2m}$ for $1 \le r < 31$.

For (14)~(15), we have

$$X_{32,2m} \xrightarrow{(14)\ (15)} MK_m, \quad 0 \le m \le 3, \tag{16}$$

$$X_{32,2m+1} \xrightarrow{(14)\ (15)} \varnothing, \quad 0 \le m \le 3. \tag{17}$$

Thus, we have $\Omega_{32,2m+1} = \varnothing$ and $\Omega_{32,2m} = \{WK_m\}$. Moreover, according to Property 2, then we have the desired conclusions which are shown in Table 2.

Note that, to fully recover the master key, the entire master key bytes must be included in the merged equation system. That is, the master key bytes can possibly be recovered only for the case that $\Sigma_{r,i} = MK$. □

Property 3. Given that a single-bit fault is inserted in $X_{t,2m}$, the fault propagation paths are shown in Figure 4. The intermediate words $X_{r,2m+1}$, $X_{r,2m+2}$, $X_{r+1,2m+2}$, $X_{r+1,2m+3}$, $X_{r+2,2m+3}$, and $X_{32,n}$ are all corrupted that $\Delta X_{r,2m+1} = \Delta_1$, $\Delta X_{r,2m+2} = \Delta X_{r+1,2m+3} = \Delta_2$, $\Delta X_{r+1,2m+2} = \Delta_3$, $\Delta X_{r+2,2m+3} =$

Table 2: The location of fault for $\#\Sigma_{r,i} = 16$ or $\#(\Sigma_{r,i} \cup \Sigma_{r,j}) = 16$.

r	I	$\#(\Sigma_{r,i}) = 16$ or $\#(\Sigma_{r,i} \cup \Sigma_{r,j}) = 16$ (i, j)
25	0, 1, 2, 3, 4	$(0,1), \ldots, (0,7), (1,2), \ldots, (1,7), (2,3), \ldots, (2,7), (3,4), \ldots, (3,7), (4,5), (4,6), (4,7)$
26	–	$(0,2), (0,3), (0,4), (0,5), (1,4), (1,5), (2,4), (2,5), (2,6), (2,7), (3,6), (3,7), (4,6), (4,7), (5,6), (5,7)$
27	–	$(0,4), (0,5), (0,6), (1,4), (1,5), (1,6), (2,6)(3,6)$
28	–	$(2,6), (2,7)$

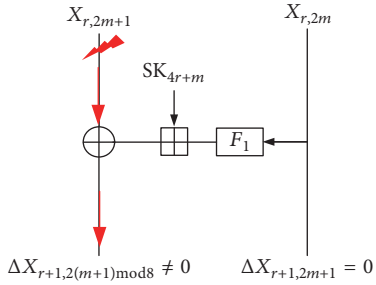

Figure 3: The fault was injected in $X_{r,2m+1}$ for $1 \le r < 31$.

Δ_5, and $\Delta X_{32,n} = \Delta_6$. Then the intermediate words are included in

$$\begin{aligned} &(X_{r+1,2m+3} \boxplus (\mathrm{SK}_{4(m+1)+1} \oplus F_1(X_{r+1,2m+2}))) \\ &\quad \oplus ((X_{r+1,2m+3} \oplus \Delta_2) \\ &\quad \boxplus (\mathrm{SK}_{4(m+1)+1} \oplus F_1(X_{r+1,2m+2} \oplus \Delta_2))) = \Delta_4, \\ &(\mathrm{WK}_t \boxplus X_{32,n}) \oplus (\mathrm{WK}_t \boxplus (X_{32,n} \oplus \Delta_5)) = \Delta C_n. \end{aligned} \tag{18}$$

More generally, if we use 8-bit words x and y to denote the inputs of modular addition, α and β to denote the difference of the inputs, and γ to denote the corresponding output difference in (18), then the above two equations can be simplified as

$$(x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma, \tag{19}$$

$$(x \boxplus y) \oplus (x \boxplus (y \oplus \alpha)) = \gamma. \tag{20}$$

That is, the intermediate words, whitening keys and subkeys can be recovered by solving the two equations.

5. The Proposed Trojan Circuit

In this section, we give the details of the HT. In general, a hardware Trojan consists of two parts: *trigger logic* (TL) and *payload logic* (PL). The TL is used to judge whether the values of signal lines and states meet the activation condition which is referred to the values of signal lines and states set by the adversary in advance. Once the activation condition is satisfied, the PL executes attacks. Attacks of Trojan circuit may deactivate the circuit (denial-of-service), change its functionality, or provide covert channels through which the protected secret information can be leaked.

5.1. Assumption of Trojan Circuits. In this paper, we make three assumptions about the design of hardware Trojan circuits.

(1) HIGHT is implemented in a cryptographic intellectual property (IP) with advanced protections like sensors from an untrusted IP vendor or system integrator. The prototype is on a Xilinx FPGA device implementing a cryptographic IP. In fact, it is a common practice to deploy physical sensors alongside cryptographic IP in industrial designs.

(2) The adversary is assumed to be able to assign the plaintext to be encrypted. And he is also assumed to be able to insert a smart but functional hardware Trojan in Register Transfer Level (RTL) by either modifying the RTL or the corresponding logic elements in the postplace or route netlist. But he only has the access to the Xilinx Design Language (XDL) file and no access to the design stage.

(3) The hardware Trojan is designed to introduce a fault by flipping only one bit of a certain intermediate word of the cipher when it was activated.

5.2. Trigger Design. The FSM-based Trojans [38] have two prominent advantages over many other Trojans: one is that they can be designed to be arbitrarily complicated with the same amount of resources and can reuse both combinational logic and flip-flops of the original circuit, and the other is that the FSM-based Trojans are bidirectional which means they can have state transitions leading back to the previous or initial state, thus causing the final Trojan state to be reached only if the entire state sequence is satisfied in consecutive clock cycles. The above two advantages both make the FSM-based Trojans harder-to-detect than other Trojans.

As shown in Figure 6, to design a hard-to-detect Trojan circuit, the TL of the proposed HT is designed based on a finite state machine (FSM). In this FSM, a 3-bit register is used to store the current state. The Trojan circuit undergoes state transition under the certain state transition diagram which is defined by the adversary in advance and shown in Figure 5. Moreover, only the adversary knows the predefined state transition diagram. The 3-bit *input* is derived from any three of the four different 8-bit intermediate words $X_{0,1}$, $X_{0,3}$, $X_{0,5}$, and $X_{0,7}$, randomly. And it is assigned as the transition condition of the FSM that causes the state transition. If the *input* agrees with the current state, the FSM will transition to the next state; otherwise the FSM will go back to the previous state. When the FSM reaches the final state S_{16}, the Trojan output is activated (the single act is "1") and the PL will cause a single-bit fault in the original circuit. In the next clock cycle,

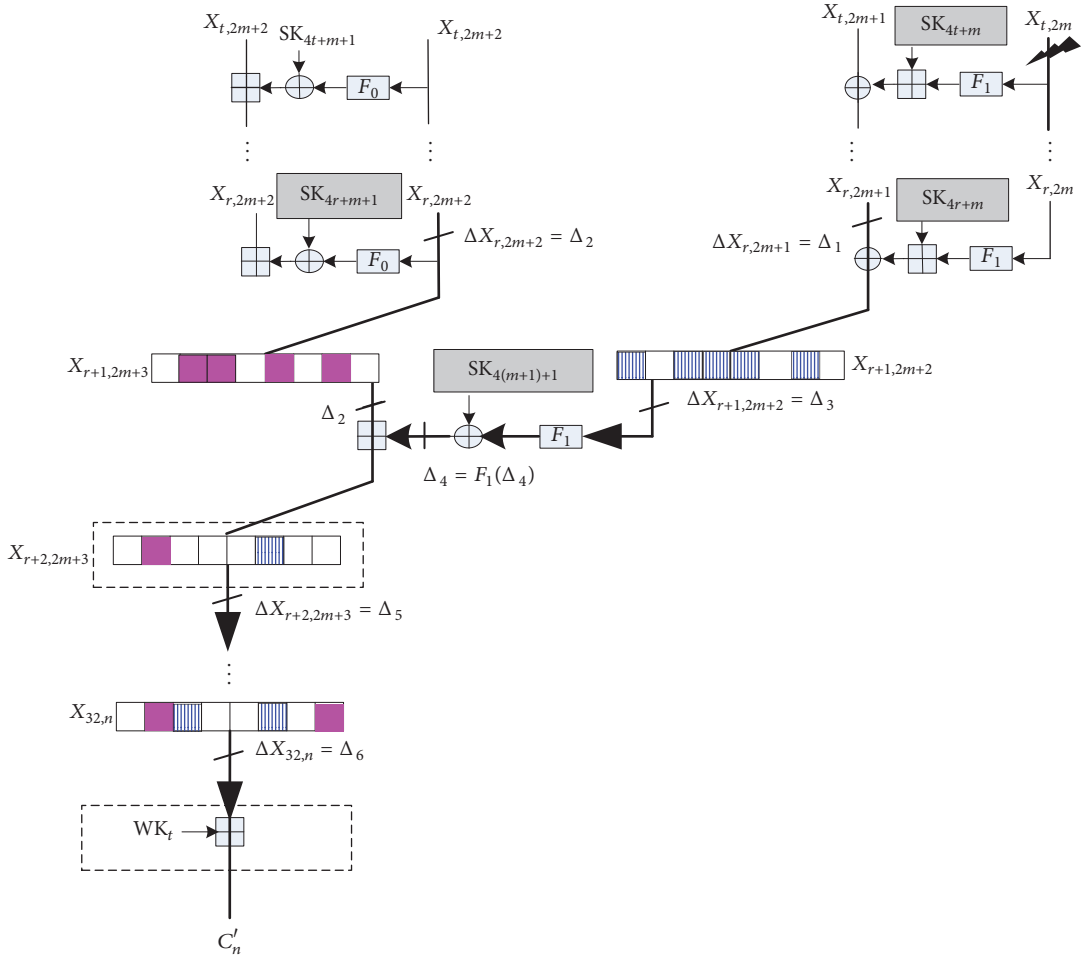

FIGURE 4: Fault propagation paths for the case where the fault is induced to $X_{t,2m}$.

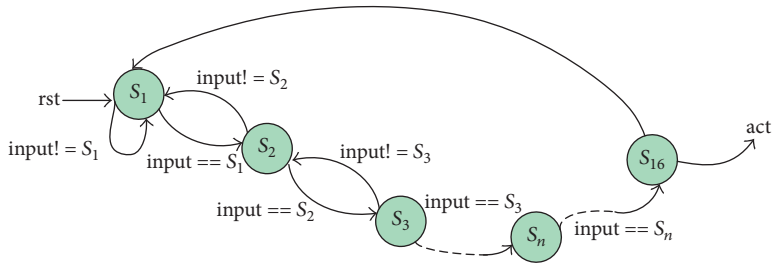

FIGURE 5: State transition graph of the FSM.

the Trojan will automatically go back to the initial state S_0; thus the Trojan can be disguised as a random fault.

Since a 3-bit register is able to store 8 different states, the test space that is to activate the trigger logic is 8! ($>2^{15}$); that is, the probability of activating the HT is Pr ≈ 0.000024 which is an extremely low probability. However, since according to *InitialTransformation* (see (5)), the required four plaintext bytes P_1, P_3, P_5, and P_7 can be directly deduced by $X_{0,1}$, $X_{0,3}$, $X_{0,5}$, and $X_{0,7}$. Hence, the adversary can trigger the HT by carefully choosing P_1, P_3, P_5, and P_7. The total logic overhead of the implemented trigger logic is three flip-flops and four 3-input LUTs.

5.3. Payload Design. For clarity, the mth encryption and plaintext are denoted as E_m and P_m, respectively. A pair of correct and faulty ciphertexts (C_m, C'_m) is required to be collected for the same plaintext P_m. The payload component PL(A) is designed to inject a single-bit fault in round r during E_m. When the HT is triggered by carefully choosing some certain plaintexts, a "1" is stored in the flip-flop M which waits for the target round $r + 1$. A signal *Rflag*, derived from state machine, indicates whether the current round is the target round $r + 1$ or not. The value of (r, i, k) is determined by AFA which will be described in detail in Section 7.2.1. Once the Trojan is triggered, the kth bit of $X_{r,i}$, that is, $X^k_{r,i}$, is flipped

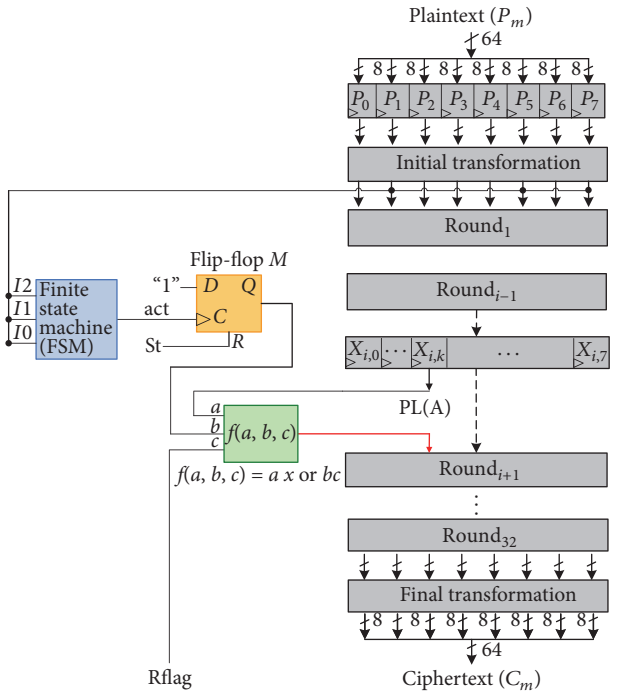

FIGURE 6: The structure of the Trojan.

due to PL(A) in E_m. This is realized by function f as shown in Figure 6. The total costs of implementing the payload logic are a flip-flop and a 3-input payload gate $f(a, b, c)$ that can be implemented by 1 LUT in both 4-input and 6-input FPGA series.

6. The AFA with a HT of HIGHT

6.1. The Optimal Location Selection. Let $X_{r,i}^k$ be the location where the HT is inserted, $0 \le r \le 32$, $0 \le i$, and $k \le 7$. In order to search the optimal location, four properties are desired:

(1) Note that the secret master key can be recovered only for the case that they are involved during the fault propagation; thus the number of elements in $\Sigma_{r,i}$ should be equal to 16.

(2) The required number of faults to recover the secret master key and the reduced key search space $\varphi(K)$ after the injection to $X_{r,i}^k$ should be both minimized to make the attack more practical.

(3) The average time of the solver to solve the merged equation system should be minimized to increase the effectiveness of the attack.

(4) $X_{r,i}^k$ should be in a deeper round to maximize utilize the fault leakages and to evade the detection.

In order to search the optimal bit location for the HT, AFA is used to enumerate every possible (r, i, k). The attempts are conducted in advance, which can guide the logic designs of the HT and reduce costs. Since AFA is executed as machine-based automation, all possible key candidates will be eventually checked along the fault propagation paths. The utilization of fault leakages is maximized. The automation shows its advantage over traditional manual analysis, such as DFA, especially when the analysis goes into the deeper round.

6.2. Constructing Algebraic Equations for Encryption of HIGHT. The task of this stage is to represent HIGHT cipher with a large system of low degree Boolean equations. Suppose $X_r = X_{r,7} \parallel X_{r,6} \cdots \parallel X_{r,1} \parallel X_{r,0}$ and $C = C_7 \parallel C_6 \parallel \cdots \parallel C_1 \parallel C_0$ are the 64-bit input of round $r + 1$ and ciphertext, respectively. Since the key schedule of HIGHT is very simple, we mainly focus on the encryption of HIGHT which is shown in Algorithm 1. From Algorithm 1, the most important yet difficult problem is to construct the equations for ARX operations.

It is stressed that in general the adversary will not choose a very deep round as the target round. That is, the rounds between the target round and *FinalTransformation* are not very large. Therefore, instead of constructing equations for the full rounds of the cipher, we only construct equations for the rounds from the target round to the *FinalTransformation* which will result in a smaller equation script and thus will accelerate the solving procedure.

According to Algorithm 1, for every fault that is injected in $X_{r,i}$, there are $64 \times (32 - r + 1)$ variables and $8 \times (8 \times (32 - r + 1) + 8)$ ANF equations were introduced to the equation system $\mathscr{C}$. In addition, $32 \times (32 - r + 1)$ variables and ANF equations are required for round keys, 64 variables and ANF equations are for the whitening keys, and 128 variables and ANF equations are for the master keys.

6.2.1. The Equations for Addition mod2^n. Assume $X, Y, Z \in \mathrm{GF}(2^n)$ are the two inputs and output of addition modulo 2^n, where $X = (x_{n-1}, x_{n-2}, \ldots, x_0)$, $Y = (y_{n-1}, y_{n-2}, \ldots, y_0)$, and $Z = (z_{n-1}, z_{n-2}, \ldots, z_0)$ with x_0, y_0, and z_0 being the least significant bit, respectively. Then addition modulo 2^n can be described as Boolean equations as follows:

$$
\begin{aligned}
z_0 &= x_0 \oplus y_0 \\
z_1 &= x_1 \oplus y_1 \oplus x_0 y_0 \\
z_2 &= x_2 \oplus y_2 \oplus x_1 y_1 \oplus (x_1 \oplus y_1)(x_1 \oplus y_1 \oplus z_1) \\
&\vdots \\
z_i &= x_i \oplus y_i \oplus x_{i-1} y_{i-1} \\
&\quad \oplus (x_{i-1} \oplus y_{i-1})(x_{i-1} \oplus y_{i-1} \oplus z_{i-1}) \\
&\vdots \\
z_{n-1} &= x_{n-1} \oplus y_{n-1} \oplus x_{n-2} y_{n-2} \\
&\quad \oplus (x_{n-2} \oplus y_{n-2})(x_{n-2} \oplus y_{n-2} \oplus z_{n-2}).
\end{aligned}
\tag{21}
$$

6.2.2. The Equations for $F_0(\cdot)$ and $F_1(\cdot)$. Given that the input and output of $F_0(\cdot)$ and $F_1(\cdot)$ are $X = (x_7, x_6, \ldots, x_0)$ and $Y = (y_7, y_6, \ldots, y_0)$, respectively, then $F_0(\cdot)$ and $F_1(\cdot)$ can be described as the following Boolean equations:

for $m = r$ **to** 31 **do**
$X_{m,0} \oplus X_{m-1,7} \oplus (F_0(X_{m-1,6}) \boxplus \mathrm{SK}_{4m-1}) = 0,\ X_{m,1} \oplus X_{m-1,0} = 0$
$X_{m,2} \oplus (X_{m-1,1} \boxplus (F_1(X_{m-1,0}) \oplus \mathrm{SK}_{4m-2})) = 0,\ X_{m,3} \oplus X_{m-1,2} = 0$
$X_{m,4} \oplus X_{m-1,3} \oplus (F_0(X_{m-1,2}) \boxplus \mathrm{SK}_{4m-3}) = 0,\ X_{m,5} \oplus X_{m-1,4} = 0$
$X_{m,6} \oplus (X_{m-1,5} \boxplus (F_1(X_{m-1,4}) \oplus \mathrm{SK}_{4m-4})) = 0,\ X_{m,7} \oplus X_{m-1,6} = 0$
end for
for $m = 32$ **do**
$X_{m,0} \oplus X_{m-1,0} = 0,\ X_{m,1} \oplus (X_{m-1,1} \boxplus (F_1(X_{m-1,0}) \oplus \mathrm{SK}_{4m-4})) = 0$
$X_{m,2} \oplus X_{m-1,2} = 0,\ X_{m,3} \oplus X_{m-1,3} \oplus (F_0(X_{m-1,2}) \boxplus \mathrm{SK}_{4m-3}) = 0$
$X_{m,4} \oplus X_{m-1,4} = 0,\ X_{m,5} \oplus (X_{m-1,5} \boxplus (F_1(X_{m-1,4}) \boxplus \mathrm{SK}_{4m-2})) = 0$
$X_{m,6} \oplus X_{m-1,6} = 0,\ X_{m,7} \oplus X_{m-1,7} \oplus (F_0(X_{m-1,6}) \boxplus \mathrm{SK}_{4m-1}) = 0$
end for
$C_0 \oplus (X_{32,0} \boxplus \mathrm{WK}_4) = 0,\ C_1 \oplus X_{32,1} = 0,\ C_2 \oplus X_{32,2} \oplus \mathrm{WK}_5 = 0,\ C_3 \oplus X_{32,3} = 0$
$C_4 \oplus (X_{32,4} \boxplus \mathrm{WK}_6) = 0,\ C_5 \oplus X_{32,5} = 0,\ C_6 \oplus X_{32,6} \oplus \mathrm{WK}_7 = 0,\ C_7 \oplus X_{32,7} = 0$
$C = C_7 \parallel C_6 \parallel C_5 \parallel C_4 \parallel C_3 \parallel C_2 \parallel C_1 \parallel C_0$

Algorithm 1: ConstructEquEncryption($X_{r,i}$).

$$F_0(\cdot) \Longleftrightarrow \begin{cases} y_7 = x_6 \oplus x_5 \oplus x_0 \\ y_6 = x_7 \oplus x_5 \oplus x_4 \\ y_5 = x_6 \oplus x_4 \oplus x_3 \\ y_4 = x_5 \oplus x_3 \oplus x_2 \\ y_3 = x_4 \oplus x_2 \oplus x_1 \\ y_2 = x_3 \oplus x_1 \oplus x_0 \\ y_1 = x_7 \oplus x_2 \oplus x_0 \\ y_0 = x_7 \oplus x_6 \oplus x_1, \end{cases} \tag{22}$$

$$F_1(\cdot) \Longleftrightarrow \begin{cases} y_7 = x_4 \oplus x_3 \oplus x_1 \\ y_6 = x_3 \oplus x_2 \oplus x_0 \\ y_5 = x_7 \oplus x_2 \oplus x_1 \\ y_4 = x_6 \oplus x_1 \oplus x_0 \\ y_3 = x_7 \oplus x_5 \oplus x_0 \\ y_2 = x_7 \oplus x_6 \oplus x_4 \\ y_1 = x_6 \oplus x_5 \oplus x_3 \\ y_0 = x_5 \oplus x_4 \oplus x_2. \end{cases}$$

6.3. Constructing Equations for the Injected Faults. This stage illustrates the method of constructing equations for the injected faults. To clarify the method, the example is shown in Figure 7.

Given that every time the HT was activated, a single-bit fault β was introduced to flip the most significant bit of $X_{25,3}$. The fault propagation paths are shown by bold line in Figure 7. The correct and faulty 64-bit inputs to the rth round are denoted by $X_r = X_{r,7} \parallel \cdots X_{r,1} \parallel X_{r,0}$ and $X'_r = X'_{r,7} \parallel \cdots X'_{r,1} \parallel X'_{r,0}$, respectively. Then, the complex fault propagation paths can be described as a set of algebraic equations with the variables that were involved. Since the fault flips the most significant bit of $X_{25,0}$, we have

$$X_{25,3} \oplus X'_{25,3} \oplus \beta = 0, \tag{23}$$

where $\beta = (1, 0, \ldots, 0)$. For the fault propagation paths that from round 25 to the *FinalTransformation*, they can be described by equations as Algorithm 2.

Algorithm 2 constructs the equations for the injected faults. The main idea is that every time a fault was induced, the intermediate variables from round r to round 32 were viewed as new variables $X'_{i,\varphi}$. Then, we reconstruct the equations for the encryption by replacing $X_{i,\varphi}$ with $X'_{i,\varphi}$. Furthermore, for variables that were not involved along the fault propagation paths Θ which can be deduced by the function *SearchFaultyInterVal*($X_{r,i}$), we have $X'_{i,\varphi} = X_{i,\varphi}$. Thus, there are $64 \times (32 - r + 1)$ variables and $8 \times (\#(\Theta) + 8 \times (32 - r + 1) + 8)$ ANF equations were introduced to the equation system $\mathscr{D}$ for every fault that was injected in $X_{r,i}$.

The function, *SearchFaultyInterVal*($X_{r,i}$) searches the faulty intermediate variables automatically according to the fault location $X_{r,i}$ and finally returns them. The main idea is explained earlier in Section 3. Algorithm 3 describes the procedure.

6.4. Solving the Equations System. After activating the inserted HT to introduce single-bit faults and constructing the merged equation system for both the cipher and faults, the whole secret master key can be fully recovered by solving the merged equation system with an automatic solver. Since the SAT-based solvers [39, 40] have prominent advantage of the memory usage when solving large equations systems over many other automatic tools, such as mutantXL algorithm [35, 37] and Gröbner basis-based [41] solvers and recently further significant improvements have been made to SAT-based solvers, we have chosen the CryptoMiniSAT v4.4 which is a DPLL-based SAT solver developed from MiniSAT to solve the equation system. The readers can refer to [25, 35, 40, 42] for details of how to generate equations and how to feed them to the solvers.

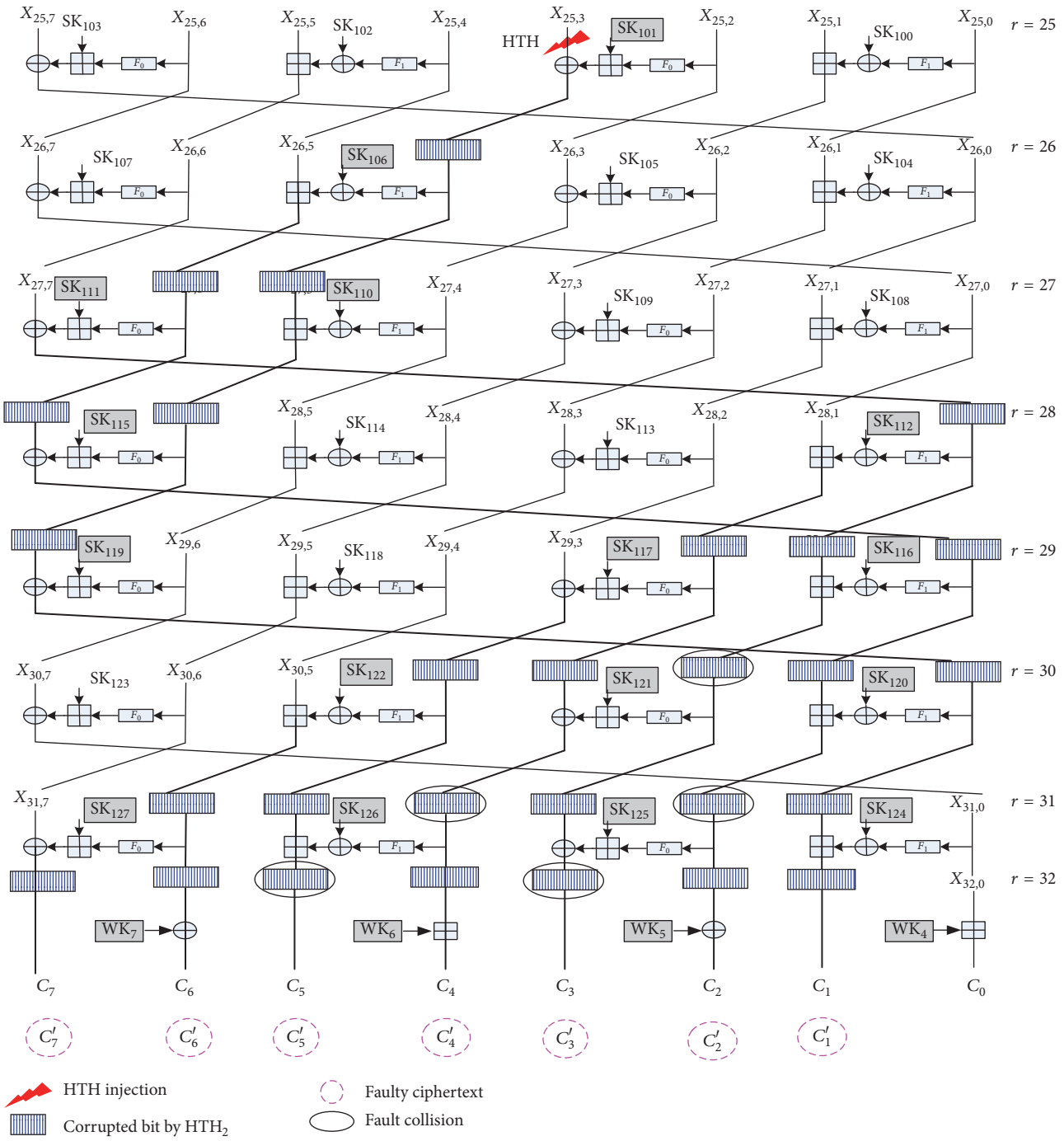

Figure 7: The fault propagation corresponding to the case where $X_{25,0}$ is faulted.

7. Theoretical and Simulation Results

In order to verify the effectiveness of the proposed attack on HIGHT and optimize the implementation of HT, we conduct many experiments and report the results in this section.

In the phase of searching the optimal location for the HT, we conduct the fault injection with software level simulations. The HIGHT software implementation was written in C and the CryptoMiniSAT 4.4 solver is running on a PC with Intel Core i7-4790, 3.60 GHZ, 12 G memory, and Windows 7 64-bit OS. An *instance* refers to one run of our attack on a set of (P, MK, C). The instance fails if the solver does not give an output within *48* hours (*172800* seconds). In the online phase, the HIGHT hardware implementation and HT are both running in SASEBO-GII board soldering a 65 nm Virtex-5 FPGA.

7.1. Data Complexity Analysis. Our aim is to fully recover the entire master key, which is mainly depending on solving the two equations (19) and (20) in Section 4. Our task is to investigate the number of queries (α, β) and $(0, \beta)$ to solve

$\Theta = I - \text{SearchFaultInterVal}\left(X_{r,i}\right)$;
while $X_{i,\varphi} \in \Theta$ **do**
$X'_{i,\varphi} \oplus X_{i,\varphi} = 0$
end while
for $m = r$ **to** 31 **do**
$X'_{m,0} \oplus X'_{m-1,7} \oplus \left(F_0\left(X'_{m-1,6}\right) \boxplus \text{SK}_{4m-1}\right) = 0,\ X'_{m,1} \oplus X'_{m-1,0} = 0$
$X'_{m,2} \oplus \left(X'_{m-1,1} \boxplus \left(F_1\left(X'_{m-1,0}\right) \oplus \text{SK}_{4m-4}\right)\right) = 0,\ X'_{m,3} \oplus X'_{m-1,2} = 0$
$X'_{m,4} \oplus X'_{m-1,3} \oplus \left(F_0\left(X'_{m-1,2}\right) \boxplus \text{SK}_{4m-3}\right) = 0,\ X'_{m,5} \oplus X'_{m-1,4} = 0$
$X'_{m,6} \oplus \left(X'_{m-1,5} \boxplus \left(F_1\left(X'_{m-1,4}\right) \oplus \text{SK}_{4m-2}\right)\right) = 0,\ X'_{m,7} \oplus X'_{m-1,6} = 0$
end for
for $m = 32$ **do**
$X'_{m,0} \oplus X'_{m-1,0} = 0,\ X'_{m,1} \oplus \left(X'_{m-1,1} \boxplus \left(F_1\left(X'_{m-1,0}\right) \oplus \text{SK}_{4m-4}\right)\right) = 0$
$X'_{m,2} \oplus X'_{m-1,2} = 0,\ X'_{m,3} \oplus X'_{m-1,3} \oplus \left(F_0\left(X'_{m-1,2}\right) \boxplus \text{SK}_{4m-3}\right) = 0$
$X'_{m,4} \oplus X'_{m-1,4} = 0,\ X'_{m,5} \oplus \left(X'_{m-1,5} \boxplus \left(F_1\left(X'_{m-1,4}\right) \boxplus \text{SK}_{4m-2}\right)\right) = 0$
$X'_{m,6} \oplus X'_{m-1,6} = 0,\ X'_{m,7} \oplus X'_{m-1,7} \oplus \left(F_0\left(X'_{m-1,6}\right) \boxplus \text{SK}_{4m-1}\right) = 0$
end for
$C'_0 \oplus \left(X'_{32,0} \boxplus \text{WK}_4\right) = 0,\ C'_1 \oplus X'_{32,1} = 0,\ C'_2 \oplus X'_{32,2} \oplus \text{WK}_5 = 0,\ C'_3 \oplus X'_{32,3} = 0$
$C'_4 \oplus \left(X'_{32,4} \boxplus \text{WK}_6\right) = 0,\ C'_5 \oplus X'_{32,5} = 0,\ C'_6 \oplus X'_{32,6} \oplus \text{WK}_7 = 0,\ C'_7 \oplus X'_{32,7} = 0$
$C' = C'_7 \parallel C'_6 \parallel C'_5 \parallel C'_4 \parallel C'_3 \parallel C'_2 \parallel C'_1 \parallel C_0$

Algorithm 2: ConstructgEquFault($X_{r,i}$).

FaultyInterVal $\leftarrow \{X_{r,i}\}$
TempArray [] [] $\leftarrow X_{r,i}$
for $m = r$ **to** 32 **do**
while $X_{m,\varphi} \in$ *TempArray*$[m]$ **do**
if $m < 32$ **then**
if φ %2 == 0 **then**
TempArray$[m+1] \leftarrow X_{m,\varphi+1}, X_{m,(\varphi+2)\%8}$
end if
if φ %2 == 1 **then**
TempArray$[m+1] \leftarrow X_{m,(\varphi+1)\%8}$
end if
end if
if m == 32 **then**
if φ %2 == 0 **then**
TempArray$[m+1] \leftarrow X_{m,\varphi}, X_{m,\varphi+1}$
end if
if φ %2 == 1 **then**
TempArray$[m+1] \leftarrow X_{m,\varphi}$
end if
end if
end while
FaultyInterVal $\leftarrow$ *FaultyInterVal* $\cup$ *TempArray* $[m]$
end for
Return *FaultyInterVal*

Algorithm 3: SearchFaultyInterVal($X_{r,i}$).

the equations. We notice that this issue was already explored from a theoretical point of view in [41]. And a worst case lower bound on the number of queries (α, β) to solve (16) is a constant 3, and the corresponding number of queries $(0, \beta)$ to solve (20) in the worst case is $(8 - t)$, where t is the position of the least significant "1" of x and $0 \le t \le 7$.

Additionally, we use N to denote the amount of faults required to recover the entire master key bits. In our case, every encryption the master key bits are assumed to be fixed while the plaintext was chosen randomly by the adversary. And since queries (α, β) and $(0, \beta)$ are introduced by activating the HT to flip one fixed bit of a certain intermediate word, the lower bound on the number of HT activated in the worst case is $N \ge 3$.

7.2. Experimental Results

7.2.1. Cost-Optimization Implementation of the HT. 6-input LUT is the mainstream look-up-table (LUT) architecture widely used from the 65 nm Virtex-5 FPGAs to the 20 nm Ultrascale FPGAs. In these devices, *slice* is the fundamental logic unit and each *slice* contains four 6-input LUTs. A single 6-input LUT is able to implement either one Boolean equation up to 6 inputs or two Boolean equations with no more than 5 different input signals in total. The structure of the 6-input LUT is shown in Figure 8.

According to Section 5, both the payload gate $f(a, b, c)$ which is illustrated in Figure 6 and the LUTs required to implement the trigger logic have 3 inputs. Moreover, occupied LUTs with 3 or less used inputs can be found by searching the XDL. In this stage, the payload gate $f(a, b, c)$ and the required four 3-input LUTs can be implemented by five arbitrarily occupied LUTs with no more than 3 used inputs, by just modifying the corresponding slice instances on XDL. Since the Trojan LUTs are implemented with existing logic, the eventual cost is *4 extra flip-flops*. The experiment result is shown in Table 3, which reports a 1.63% additional cost in flip-flops for the HT implemented on a 65 nm Virtex-5 FPGA.

Table 3: Overhead report of inserted HT.

	Slice	LUT	Flip-flop
HIGHT	404	750	245
Trojan HIGHT	404	750	249
Overhead	0%	0%	1.63%

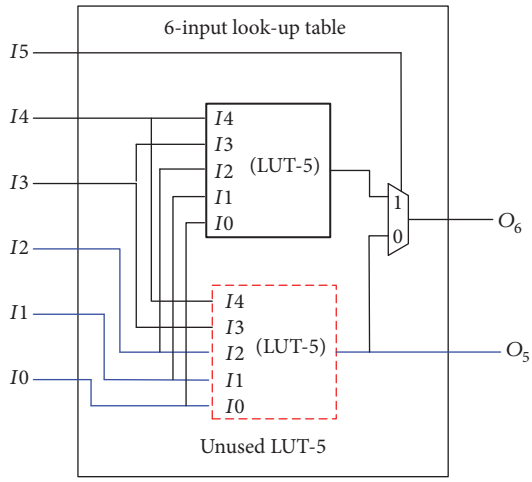

Figure 8: The structure of 6-input LUT.

7.2.2. The Optimal Location Selection for Inserting the HT

(1) Determining k. According to Section 5.1, to make the designed HT stealthy and reduce the costs, the HT is designed to flip only one bit of the 8-bit intermediate word when it is activated. Since the lower bound on the number of queries (α, β) to solve (16) in the worst case is inversely proportional to the size of t, we choose the most significant bit of the 8-bit intermediate word to be flipped. That is, $k = 7$. And in that case, the resulting N is minimum.

(2) Determining (r, i). According to Property 3 of the faults (Section 4), only when $r \leq 25$ the entire master key bytes can be involved during the fault propagation. Thus, the HT should be inserted in $X_{r,i}$ $(r \leq 25)$ to ensure the entire secret master key variables are included in the algebraic equations of the injected faults (Section 6.3). And there are 5 candidate locations $(r, i) = \{(25, 0), (25, 1), (25, 2), (25, 3), (25, 4)\}$ and each of them is tested by AFA to get the optimal location for the HT.

According to the equations for addition mod2^n in Section 6.2.1, the most significant bit of x and y for (19) and (20) can be never recovered for no observation which can be made about the most significant bit of γ. Thus, there are multiple solutions for (19) and (20); that is, multiple candidates for MK will be collected by solving the merged equation system with an SAT solver. To determine MK uniquely, a distinguisher is required to further filter the MK candidates.

The CryptoMiniSAT solver searches for the given amount of solutions, which means all candidates for MK will be checked if the given amount of solutions is set large enough. With this property, we can build a distinguisher. Note the fact that the unknown intermediate words X_r and the known ciphertext C are both depending on (P, MK); we can filter the MK candidates against C by constructing equations for the full rounds of HIGHT. For every MK candidate, C^* will be automatically deduced by the solver based on the MK candidate and the known P. If C^* does not match C, the candidate will be eliminated. Thus, with this property of the solver, an efficient distinguisher can be built just by constructing equations for the full rounds of HIGHT. Since the equations for round r to *FinalTransformation* has been constructed in Algorithm 1, we only need to construct equations for *InitialTransformation* to round $r - 1$ of HIGHT to build the distinguisher which is shown in Algorithm 4. Hence, there are additional $64 \times r$ variables and $8 \times (8 \times r + 8)$ ANF equations are required for the intermediate words, $32 \times r$ variables and ANF equations are required for round keys, and 64 variables and ANF equations are required for the whitening keys.

In order to verify the effectiveness of the distinguisher, that is, whether the entire master key bits can be uniquely determined, we set the given amount of solutions to 2^{128} for the solver. And the simulations are conducted under two different modes: one is denoted by mode A which is with the distinguisher and the other is denoted by mode B which is without the distinguisher. We use the method in Section 6 to build the emerged algebraic equation system for the cipher and the injected faults. The results in Table 4, which are derived statistically from 100 instances, show the statistics of solutions corresponding to different (r, i) under mode A and mode B with the number of faults N which varies from 3 to 9. We can see that the cases, which are conducted under mode A, have a unique solution for $N \geq 3$; that is, the entire 128-bit master key can be uniquely determined for these cases. However, when these attacks are conducted under mode B, the CryptoMiniSAT solver always outputs multiple solutions and the number of solutions seem to be inversely proportional to N. Thus, the experimental results indicate that the distinguisher is feasible and effectiveness and also proving the lower bound in the worst case for N is 3.

Serving the purpose of accelerating the experiments, five PCs with the same configuration are employed to run the CryptoMiniSAT solver in parallel so as to finish these attacks. Each PC runs 20 instances. Figure 9 shows the average solving time of the CryptoMiniSAT solver corresponding to the cases where the HT is inserted in $X_{25,i}$ $(i = 0, 1, 2, 3, 4)$ under mode A. The figure shows that when $N < 3$, the secret master key is failed to be recovered in 48 hours. And for the cases where the HT is inserted in $(25, 0)$ and $(25, 4)$, the minimum value for N is 4, while for the cases $(25, 1)$, $(25, 2)$, and $(25, 3)$ the minimum value for N is 3. The figure also clearly shows the distributions corresponding to the case $(25, 3)$ having a lower average compared to the other. Thus the optimal location for inserting the HT is $(25, 3)$ and the corresponding average solving time when $N = 3$ is $t_0 = 12572.26$ seconds (≈3.49 hours).

7.2.3. Success Rate of the Attack. To evaluate the success rate of the attack under mode A where the HT is inserted in the

$P = P_7 \parallel P_6 \parallel P_5 \parallel P_4 \parallel P_3 \parallel P_2 \parallel P_1 \parallel P_0$
$X_{0,0} \oplus (P_0 \boxplus \mathrm{WK}_0) = 0,\ X_{0,1} \oplus P_1 = 0,\ X_{0,2} \oplus P_2 \oplus \mathrm{WK}_1 = 0,\ X_{0,3} \oplus P_3 = 0$
$X_{0,4} \oplus (P_4 \boxplus \mathrm{WK}_2) = 0,\ X_{0,5} \oplus P_5 = 0,\ X_{0,6} \oplus P_6 \oplus \mathrm{WK}_2 = 0,\ X_{0,7} \oplus P_7 = 0$
for $m = 0$ **to** $r - 1$ **do**
$X_{m,0} \oplus X_{m-1,7} \oplus (F_0(X_{m-1,6}) \boxplus \mathrm{SK}_{4m-1}) = 0,\ X_{m+1,1} \oplus X_{m,0} = 0$
$X_{m,2} \oplus (X_{m-1,1} \boxplus (F_1(X_{m-1,0}) \oplus \mathrm{SK}_{4m-4})) = 0,\ X_{m+1,3} \oplus X_{m,2} = 0$
$X_{m,4} \oplus X_{m-1,3} \oplus (F_0(X_{m-1,2}) \boxplus \mathrm{SK}_{4m-3}) = 0,\ X_{m,5} \oplus X_{m-1,4} = 0$
$X_{m,6} \oplus (X_{m-1,5} \boxplus (F_1(X_{m-1,4}) \oplus \mathrm{SK}_{4m-2})) = 0,\ X_{m,7} \oplus X_{m-1,6} = 0$
end for

Algorithm 4: ConstructDistinguisher($X_{r,i}$).

Table 4: Statistics of solutions corresponds to different (r, i) under mode A and mode B.

(r, i)	Mode A (with a distinguisher)							Mode B (without a distinguisher)						
	$N = 9$	$N = 8$	$N = 7$	$N = 6$	$N = 5$	$N = 4$	$N = 3$	$N = 9$	$N = 8$	$N = 7$	$N = 6$	$N = 5$	$N = 4$	$N = 3$
(25, 0)	1	1	1	1	1	1	–	8.24	8.48	8.64	8.96	9.44	11.74	–
(25, 1)	1	1	1	1	1	1	1	16.76	17.55	18.58	21.33	26.40	264.47	9660.49
(25, 2)	1	1	1	1	1	1	1	2.12	2.40	2.66	2.78	3.20	3.93	2654.37
(25, 3)	1	1	1	1	1	1	1	34.74	34.82	35.66	36.57	39.31	55.77	5782.57
(25, 4)	1	1	1	1	1	1	–	8.32	8.48	8.96	8.96	10.88	21.69	–

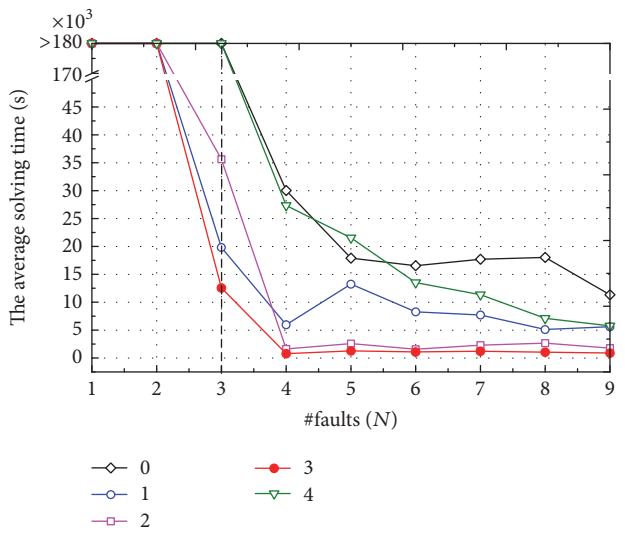

Figure 9: The results of the cases where the HT is inserted in $X_{25,i}$ under mode A.

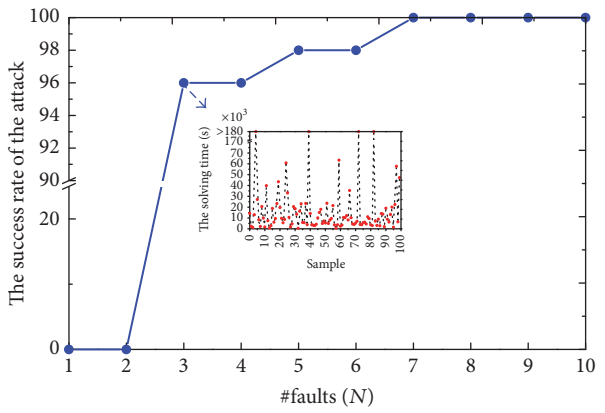

Figure 10: Success rate of the attack.

optimal location determined in Section 7.2.2, 100 instances are tested with different (P, MK). Figure 10 shows the success rate of the attack. It can be seen that when N is lower than 3, the success rate of the attack remains 0%. Once N is greater than or equal to 3, as N taken grows, the success rate of the attack increases. And the success rate of the attack can reach 100% by increasing N to 7. It can be also seen in the lower part of Figure 10 that when N is equal to 3, only 4 instances fail to recover the secret master key in 48 hours; thus the success rate of the attack is 96%.

8. Conclusions

In this paper, an *algebraic fault analysis* (AFA), relying on the stealthy hardware Trojan, against HIGHT cipher has been proposed. To facilitate a bit-level AFA of HIGHT, a FSM-based stealthy HT is designed with an extremely low activation rate of around 0.000025. The optimal location for inserting the HT is determined by AFA in advance. Experiments report a 1.63% additional cost in flip-flops for the HT implemented on a 65 nm Virtex-5 FPGA. As for HIGHT implementation, a single-bit flip on the most significant bit of $X_{25,3}$ when the HT is activated requires only 3 injections to recover the secret master key with a success rate of 96%. In this paper, we showed that even with very limited number of faults from a lightweight Trojan, modern

cryptographies are still vulnerable against algebraic attacks. This work certified the severity of the lightweight HT for the security-critical ciphers in ICs, and hence extensive security investigations must be devoted throughout the entire design and manufacture process of the security chips.

In the future work, we aim at explore effective solutions to detect the stealthy Trojan injected inside the cryptographic circuits.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is sponsored in part by the National Natural Science Foundation of China (no. 61272491, no. 61309021, no. 61472357, and no. 61571063), by the China Scholarship Council (Grant no. CSC201606325012), by the Science and Technology on Communication Security Laboratory (9140C110602150C11053), and by the Major State Basic Research Development Program (973 Plan) of China under Grant 2013CB338004.

References

[1] P. Sethi and S. R. Sarangi, "Internet of things: architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, Article ID 9324035, pp. 1–25, 2017.

[2] A. Juels, "RFID security and privacy: a research survey," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 381–394, 2006.

[3] P. H. Cole, D. C. Ranasinghe, and C. Damith, *Networked RFID Systems and Lightweight Cryptography: Raising Barriers to Product Counterfeiting*, Springer, Berlin, Germany, 2008.

[4] A. Bogdanov, L. R. Knudsen, G. Leander et al., "PRESENT: an ultra-lightweight block cipher," in *Proceeding of CHES 2007*, vol. 4727 of *Lectures in computer science*, pp. 450–466, Heidelberg, 2007.

[5] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, "The LED block cipher," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 6917, pp. 326–341, 2011.

[6] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The SIMON and SPECK lightweight block cIPhers," *Cryptology ePrint Archive*, 2013, http://eprint.iacr.org/.

[7] C. H. Lim and T. Korkishko, "mCrypton – A lightweight block cipher for security of low-cost RFID tags and sensors," in *Information Security Applications*, vol. 3786 of *Lecture Notes in Computer Science*, pp. 243–258, Springer, Berlin, Heidelberg, 2006.

[8] D. Hong, J. Sung, S. Hong et al., "HIGHT: a new block cipher suitable for low-resource device," in *Cryptographic Hardware and Embedded Systems—CHES 2006: 8th International Workshop, Yokohama, Japan, October 10–13*, vol. 4249 of *Lecture Notes in Computer Science*, pp. 46–59, Springer, Berlin, Germany, 2006.

[9] International Organization for Standardization, ISO/IEC18033-3:2005, Information technology-Security techniques – Encryption algorithms-Part 3: Block ciphers (2005).

[10] R. Kumar, P. Jovanovic, W. Burleson, and I. Polian, "Parametric trojans for fault-injection attacks on cryptographic hardware," in *Proceedings of the 11th Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014*, pp. 18–28, Republic of Korea.

[11] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[12] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[13] W. Danesh, J. Dofe, and Q. Yu, "Efficient hardware Trojan detection with differential cascade voltage switch logic," *VLSI Design*, vol. 2014, Article ID 652187, 2014.

[14] M. L. Flotters, "On the effectiveness of hardware trojan detection via sidel-channel analysis," *Information Security Journal: A Global Perspective*, no. 22, pp. 226–236, 2013.

[15] S. Bhunia, M. Abramovici, D. Agrawal et al., "Protection against hardware trojan attacks: Towards a comprehensive solution," *IEEE Design and Test*, vol. 30, no. 3, pp. 6–17, 2013.

[16] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proceeding of CRYPTO 1997*, vol. 1294 of *Lecture Notes in Computer Science*, pp. 513–525, Springer, Heidelberg, Berlin, Germany, 1997.

[17] N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *Advances in cryptology ASIACRYPT 2002*, vol. 2501 of *Lecture Notes in Computer Science*, pp. 267–287, Springer, Berlin, Berlin, 2002.

[18] N. T. Courtois, D. Ware, and K. Jackson, "Fault-Algebraic Attacks on Inner Rounds of DES," in *Proceedings of the eSmart 2010*, pp. 22–24, 2010.

[19] M. S. E. Mohamed, S. Bulygin, and J. Buchmann, "Using SAT solving to improve differential fault analysis of Trivium," *International Journal of Security and Its Applications*, vol. 6, no. 1, pp. 29–38, 2012.

[20] S. Sarkar, S. Banik, and S. Maitra, "Differential fault attack against grain family with very few faults and minimal assumptions," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1647–1657, 2015.

[21] G. Piret and J. Quisquater, "A differential fault attack technique against spn structures, with application to the AES and khazad," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, vol. 2779 of *Lecture Notes in Computer Science*, pp. 77–88, Springer, Berlin, Heidelberg, Germany, 2003.

[22] P. Jovanovic, M. Kreuzer, and I. Polian, "An Algebraic Fault Attack on the LED Block Cipher," *Cryptology ePrint Archive*, 2012, http://eprint.iacr.org/2012/400.pdf.

[23] X. J. Zhao, S. Z. Guo, F. Zhang, Z. J. Shi, C. J. Ma, and T. Wang, "Algebraic differential fault attacks on LED using a single fault injection," *Cryptology ePrint Archive*, http://eprint.iacr.org/2012/347.pdf.

[24] Z. Wang, X. Dong, K. Jia, and J. Zhao, "Differential fault attack on KASUMI cipher used in GSM telephony," *Mathematical Problems in Engineering*, vol. 2014, Article ID 251853, 2014.

[25] F. Zhang, X. J. Zhao, S. Guo, T. Wang, and Z. J. Shi, "Improved algebraic fault analysis: a case study on piccolo and applications to other lightweight block ciphers," in *Proceedings of the COSADE 2013*, vol. 7864 of *Lecture Notes in Computer Science*, pp. 62–79, Springer.

[26] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice,

and countermeasures," Tech. Rep. 11, Politecnio di Milano, Milan, Italy, 2012.

[27] National Institute of Advanced Industrial Science and Technology (AIST), *Side-channel Attack Standard Evaluation Board SASEBO-GII Specification*, 1.01 edition, 2009.

[28] J. Lu, "Cryptanalysis of reduced versions of the HIGHT block cipher from CHES 2006," in *Information security and cryptology (ICISC 2007)*, vol. 4817 of *Lecture Notes in Computer Science*, pp. 11–26, Springer, Berlin, Germany, 2007.

[29] O. Özen, K. Varici, C. Tezcan, and C. Kocair, "Lightweight block ciphers revisited: cryptanalysis of reduced round PRESENT and HIGHT," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 5594, pp. 90–107, Springer, Heidelberg, Berlin, Germany, 2009.

[30] P. Zhang, B. Sun, and C. Li, "Saturation attack on the block cipher HIGHT," in *Proceedings of the CANS 2009*, vol. 5888, pp. 76–86.

[31] B. Koo, D. Hong, and D. Kwon, "Related-Key Attack on the Full HIGHT," in *in Proceeding of ICISC, 2010., LNCS*, vol. 6829, pp. 49–67, Springer, Heidel-berg, Berlin, Germany, 2011.

[32] D. Hong, B. Koo, and D. Kwon, "Biclique Attack on the Full HIGHT," in *in Proceeding of ICISC, 2011, LNCS*, vol. 7259, pp. 365–374, Springer, Heidel-berg, Berlin, Germany, 2012.

[33] Y. Lee, J. Kim, J. H. Park, and S. Hong, "Differential fault analysis on the block cipher HIGHT," *Lecture Notes in Electrical Engineering*, vol. 164, no. 1, pp. 407–416, 2012.

[34] M. S. Mohamed, W. S. Mohamed, J. Ding, and J. Buchmann, "MXL2: solving polynomial equations over GF(2) Using an improved mutant strategy," in *Post-quantum cryptography*, vol. 5299 of *Lecture Notes in Computer Science*, pp. 203–215, Springer, Heidelberg, Berlin, 2008.

[35] M. S. E. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter, and J. Buchmann, "Improved algebraic side-channel attack on AES," *Cryptology ePrint Archive*, pp. 146–151, 2011, http://eprint.iacr.org/2012/084.pdf.

[36] M. Renauld and F.-X. Standaert, "Algebraic side-channel attacks," in *Information security and cryptology*, vol. 6151 of *Lecture Notes in Computer Science*, pp. 393–410, Springer, Berlin, 2010.

[37] J. C. Faugère, "Gröbner Bases," in *FSE 2007, Invited Talk (2007)*, Applications in Cryptology, 2007, http://fse2007.uni.lu/slides/faugere.pdf.

[38] X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia, "Sequential hardware trojan: side-channel aware design and placement," in *Proceedings of the 29th IEEE International Conference on Computer Design 2011, ICCD 2011*, pp. 297–300, USA, November 2011.

[39] V. B. Gregory, *Algebraic Cryptanalysis*, Springer, 2009.

[40] SAT, Sat Race Competition, http://www.satcompetition.org/.

[41] S. Paul and B. Preneel, *Solving Systems of Differential Equations of Additions*, vol. 3574 of *Proceeding of ACISP 2005*, LNCS, Springer, Heidelberg, Berlin, Germany, 2005.

[42] J. Ding, J. Buchmann, M. S. E. Mohamed, M. Mohamed, and R. P. Weinmann, "MutantXL algorithm," in *Proceedings of the 1st International Conference in Symbolic Computation and Cryptography*, pp. 16–22, 2008.