IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 2, pp. 289–311. DOI:10.46586/tches.v2022.i2.289-311

Free Fault Leakages for Deep Exploitation: Algebraic Persistent Fault Analysis on Lightweight Block Ciphers

Fan Zhang[1,2,3], Tianxiang Feng[1,4], Zhiqi Li[1], Kui Ren[1] and Xinjie Zhao[5,1]

[1] School of Cyber Science and Technology, College of Computer Science and Technology, Zhejiang University, Hangzhou, China, 310027
[2] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, China, 100878
[3] Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, Hangzhou, China, 310027
[4] Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou, China, 310027
[5] Henan Province Key Laboratory of Cyberspace Situation Awareness, Zhengzhou, China, 450001
fanzhang@zju.edu.cn,fengtianxiang@zju.edu.cn

Abstract. *Persistent Fault Analysis* (PFA) is a new fault analysis method for block ciphers proposed in CHES 2018, which utilizes those faults that persist in encryptions. However, one fact that has not been raised enough attention is that: while the fault itself does persist in the entire encryption, the corresponding statistical analysis merely leverages fault leakages in the last one or two rounds, which ignores the valuable leakages in deeper rounds. In this paper, we propose *Algebraic Persistent Fault Analysis* (APFA), which introduces algebraic analysis to facilitate PFA. APFA tries to make full usage of the free fault leakages in the deeper rounds without incurring additional fault injections. The core idea of APFA is to build similar algebraic constraints for the output of substitution layers and apply the constraints to as many rounds as possible. APFA has many advantages compared to PFA. First, APFA can bypass the manual deductions of round key dependencies along the fault propagation path and transfer the burdens to the computing power of machine solvers such as Crypto-MiniSAT. Second, thanks to the free leakages in the deeper round, APFA requires a much less number of ciphertexts than previous PFA methods, especially for those lightweight block ciphers such as PRESENT, LED, SKINNY, etc. Only 10 faulty ciphertexts are required to recover the master key of SKINNY-64-64, which is about 155 times of reduction as compared to the state-of-the-art result. Third, APFA can be applied to the block ciphers that cannot be analyzed by PFA due to the key size, such as PRESENT-128. Most importantly, APFA replaces statistical analysis with algebraic analysis, which opens a new direction for persistent-fault related researches.

Keywords: Algebraic · PFA · Fault Analysis · Fault Attack · PRESENT · LED · SKINNY · LBlock · AES

1 Introduction

Different from *Side Channel Attack* (SCA), *Fault Attack* can be considered as an active attack method that targets physical cryptographic equipment [JT12]. It performs the *fault injection* (FI) on the device, and conducts the *fault analysis* (FA) to recover the key by analyzing those collected faulty ciphertexts [DLV03]. It was first proposed by Boneh *et al.* in 1997 [BDL97]. And then, Biham and Shamir proposed the well-known *Differential Fault*

Received: 2021-10-15 Accepted: 2021-12-15 Published: 2022-02-15

Analysis (DFA) and applied it to the block cipher DES [BS97]. DFA uses the differential information between the correct and the faulty ciphertext under the same plaintext to recover the key. Actually, it can be applied to most of the mainstream cryptographic algorithms [PQ03].

A typical fault injection can be achieved by changing the power supply voltage [BBBP13], the frequency of the external clock [ABF+02], the temperature [HS13], or exposing the circuit to a laser [SA02]. Such injection is induced within a very short period of time during the encryption process, and it will only affect the result of single encryption. The corresponding fault model is typically named the transient fault model. There are also other models that assume permanent faults. For example, attackers may use more aggressive methods to permanently break the device [Sko10]. In CHES 2018, Zhang *et al.* proposed a new fault analysis method called *Persistent Fault Analysis* (PFA) [ZLZ+18]. Their fault model is sitting between transient and permanent ones, where the fault persists in multiple encryptions and disappears after the device is reset. As for a single encryption, the persistent fault will last for multiple rounds.

Traditional DFA can use a single-byte fault and several pairs of ciphertexts to recover the key. Mukhopadhyay induced a single-byte fault into the input of the 8-th round of AES, and utilized two pairs of faulty ciphertexts to recover the full key [Muk09]. Tunstall *et al.* extended the work in [Muk09] to exploit the relationship of round keys between the 9-th and 10-th round, and used only one pair of faulty ciphertexts to reduce the key space to 2^8 [TMA11]. However, when the fault is injected into the deeper round (*e.g.,* beyond the 7-th round in AES), the fault difference along the propagation path becomes more and more complex, which makes it difficult to analyze.

To improve the efficiency of fault analysis, different automation techniques are taken into consideration. There are two main categories. One is to conduct the DFA with automated analysis. Saha *et al.* proposed an automated analysis framework for DFA to determine whether the injected fault can be utilized [SKMD17]. Khairallah *et al.* proposed a general DFA analysis method for SPN block ciphers, and automated the analysis on various block ciphers [KHN+19]. Another direction is to introduce algebraic techniques for analysis. Courtois *et al.* proposed the algebraic fault analysis (AFA) in 2010 [CJW10], which can be used to analyze those faults injected in deeper rounds. AFA combines algebraic analysis and fault analysis. It transforms the differential fault information into the corresponding algebraic equations, and combines them with the encryption algorithm. AFA can reduce both the search space of the secret key and the number of the faulty ciphertexts that is required.

The recently proposed *Persistent Fault Analysis* (PFA) provides a new direction for fault attacks. In [ZLZ+18], Zhang *et al.* used the RowHammer technology [KDK+14] to conduct the injection, so that a bit in the S-box was persistently flipped. They used this method to recover the AES secret key with less than 2500 faulty ciphertexts. After the debut of PFA in CHES 2018, many cryptanalysts are following this direction. In 2019, Caforio *et al.* extended PFA to those ciphers with Feistel networks [CB19], while the original PFA can only be applied to SPN block ciphers. In 2020, *Enhanced Persistent Fault Analysis* (EPFA) was proposed to take a further step to exploit the fault leakage in the penultimate round of AES [XZY+20]. However, it is very complicated for EPFA or PFA to analyze the fault leakage in the penultimate round, and nearly impossible to go deeper in the inner rounds. Meanwhile, both PFA and EPFA may face another challenge, that is, how to exploit multiple fault leakages in much deeper rounds simultaneously. In addition, the automation of PFA has not been carefully studied.

In this paper, we propose *Algebraic Persistent Fault Analysis* (APFA), which introduces algebraic techniques into the analysis of PFA. APFA inherits the advantages of both PFA and AFA, and makes better use of multiple rounds of fault leakages than PFA. To the best of our knowledge, this is the first time to combine algebraic analysis with persistent analysis.

In fact, the original PFA merely uses statistical methods to analyze faulty ciphertexts, so do other similar analysis methods such as the Statistical Ineffective Fault Analysis (SIFA) [DEK+18]. How to express persistent fault information at the algebraic level has not been investigated yet.

This paper provides a new direction for the subsequent analysis of PFA. The main contributions of this work can be summarized as follows:

- We propose *Algebraic Persistent Fault Analysis* (APFA) as a new analysis method combining PFA and AFA, which can be applied to most lightweight SPN block ciphers and can exploit multi-round fault leakage as much as possible.
- We apply APFA to a variety of lightweight block ciphers, such as PRESENT [BKL+07], LED [GPPR11] and SKINNY [BJK+16]. Among them, the challenge that PFA is difficult to cope with PRESENT-128 is conquered, and the number of ciphertexts required for SKINNY-64-64 is reduced from 1550 to 10.
- We extend APFA method to the Feistel-based block cipher LBlock [WZ11] and the classic block cipher AES [DR99] in order to verify the generality of APFA.
- We investigate APFA under more practical fault scenarios, where the fault value or the fault location might be unknown.

One of the most important contributions of APFA is that it can significantly reduce the number of faulty ciphertexts that are required. Three counterparts are selected for comparison: the original PFA [ZLZ+18], the improved PFA with Maximum Likelihood Estimation [ZZJ+20], and the extended PFA with GPU acceleration [XZY+20]. The results are shown in Table 1, where the listed numbers are the average number of ciphertexts that are required for fully recovering the master key. Compared with previous work, three major conclusions can be summarized. (1) The number of ciphertexts required for successful attack is dramatically reduced, where the most exciting result can be found at SKINNY-64-64: the required number of ciphertexts is reduced by 155 times (*i.e.,* 1550/10, See Row 4 in Table 1) as compared with EPFA. (2) Meanwhile, APFA can manage to work well on PRESENT-128 (Row 2) and SKINNY-64-128 (Row 6), which are assumed as very difficult to break by PFA or EPFA. (3) APFA can also deal with Feistel-based block cipher, *e.g.,* LBlock-80 (Row 6).

Table 1: Comparison of different PFA methods on different ciphers.

Type	RowID	Design	Cipher	Analysis Method					Reduced number of ciphertexts in times
				PFA-18 [ZZJ+20]	PFA-20 [ZZJ+20]	EPFA [XZY+20]	This paper	Section	
Lightweight Block Ciphers	1	SPN	PRESENT-80	-	101	-	18	Sec. 7.3	5.61×
	2		PRESENT-128	-	-	-	28		-
	3		LED-64	-	-	75	23	Sec. 7.4	3.26×
	4		SKINNY-64-64	-	-	1550	10	Sec. 7.5	155.00×
	5		SKINNY-64-128	-	-	-	33		-
	6	Feistel	LBlock-80	-	-	-	112	Sec. 8.1	-
Classic Block Ciphers	7	SPN	AES-128	2281	1641	1000	1300	Sec. 8.2	-1.30×

2 Background

In this section, we will introduce the related background, SPN block cipher, *Persistent Fault Analysis* (PFA) and algebraic cryptanalysis.

2.1 SPN Block Cipher

In block cipher design, *Substitution–Permutation Network* (SPN) is an important structure. Suppose the block cipher is denoted as $\mathbb{B}$. A plaintext $\boldsymbol{P}$ is passed to an SPN to produce

a ciphertext $\boldsymbol{C}$. The block size of $\mathbb{B}$ is n bits, and each element in the block is of w bits. For PRESENT, $w = 4$ (*i.e.,* a nibble) and for AES, $w = 8$ (*i.e.,* a byte).

A round function F is typically used in SPN-based block ciphers. The r-th round of $\mathbb{B}$, which contains three operations. (1) Substitution layer (SB). This operation takes the data block $\boldsymbol{X}$ as input, substitutes the value according to the S-box S, and outputs the data block $\boldsymbol{Y}$. That is, $\boldsymbol{Y} = \mathsf{SB}(\boldsymbol{X})$. This substitution should be invertible, *i.e.,* S is a bijection; (2) Permutation layer (PL). This operation takes the data block as input and permutes the bits according to specific rules. In terms of the whole block, there is an index mapping in PL. *e.g.,* the j-th bit y_j of the output $\boldsymbol{Y}$ is equal to the i-th bit x_i of the input $\boldsymbol{X}$. The inverse operation of PL can be denoted as PL^{-1}; (3) Addition. This operation is generally the only function directly related to the key. It can be represented as $\boldsymbol{Y} = \mathsf{AK}(\boldsymbol{X}, \boldsymbol{K})$. In addition, $\boldsymbol{K}$ can be replaced by a constant, which is denoted as AC.

Besides the encryption with the round function F, the SPN block cipher $\mathbb{B}$ also uses the key schedule to expand the master key into several round keys.

2.2 Persistent Fault Analysis

Persistent Fault Analysis (PFA) mainly performs a statistical analysis on the value distribution of each byte in ciphertexts, thus it can be used as a ciphertext-only cryptanalysis method. The target is aimed at the substitution table, and the attack can be described as follows: (1) The adversary injects a persistent fault into the substitution table S, which persists in the multiple rounds or encryptions. (2) The victim uses a fixed key to encrypt multiple plaintexts via the faulty device. (3) The adversary collects the faulty ciphertext and recovers the key through statistical analysis.

The original PFA focused on the last round of encryption. Since the S-box is injected with a persistent fault, one element value in S will be changed. As a result, the original value of that element will not appear in the output of SubBytes. In other words, the XOR result of the original value and the key byte might not appear in the final ciphertext.

Taking AES-128 as an example, in order to simplify the analysis, the permutation layer of AES after SubBytes is ignored. In the last round, X_i is one byte in the input of S-box, C_i is one byte in the ciphertext $\boldsymbol{C}$. $\boldsymbol{K}^R$ is the last round key (where $R = 10$ rounds) and K_i is one key byte of $\boldsymbol{K}^R$. $\mathsf{S}[X_i]$ and K_i are XORed to get C_i, *i.e.,* $C_i = \mathsf{S}[X_i] \oplus K_i$. The goal of PFA is to use the collected $\boldsymbol{C}$ to recover $\boldsymbol{K}^R$.

Suppose the fault is injected to $\mathsf{S}[l]$ (*i.e.,* the l-th byte of S) whose original value is V. l is also called the fault location. $\mathsf{S}^{'}$ denotes the faulty S-box. The fault causes $\mathsf{S}[l] = V$ to be changed to a new value U by a fault value f, *i.e.,* $\mathsf{S}^{'}[l] = V \oplus f = U$. For the unaffected S-box, since S is bijective, its input and output hold a one-to-one mapping. Therefore, the probability of each output value is 1/256, assuming a uniform distribution for the input. However, for the affected S-box, due to the induced fault, V will never be observed as the output, while the appearance of U will hold a larger probability (appears twice in $\mathsf{S}^{'}$). Therefore, the probability distribution of $\mathsf{S}^{'}[X_i]$ can be represented as shown in Eq.(1):

$$Prob(\mathsf{S}^{'}[X_i]) = \begin{cases} 0, & \text{if} \quad \mathsf{S}^{'}[X_i] = V, \\ \frac{2}{256}, & \text{if} \quad \mathsf{S}^{'}[X_i] = U, \\ \frac{1}{256}, & \text{otherwise.} \end{cases} \tag{1}$$

Since $C_i = \mathsf{S}^{'}[X_i] \oplus K_i$, and $\mathsf{S}^{'}[X_i] \neq V$, we have:

$$\left.\begin{array}{c} \mathsf{S}^{'}[X_i] \neq V \\ C_i = \mathsf{S}^{'}[X_i] \oplus K_i \end{array}\right\} \Longrightarrow K_i \neq C_i \oplus V \tag{2}$$

Taking specific numbers as an illustration, if $V = 0\text{x}63, U = 0\text{x}62$ in Eq.(2) and we get a ciphertext byte of $C_i = 0\text{x}01$, it can be deduced that $K_i \neq C_i \oplus V$, *i.e.,* $K_i \neq 0\text{x}62$. In this case, the search space of K_i is reduced to the set of $\{0\text{x}00, \cdots, 0\text{x}61, 0\text{x}63, \cdots 0\text{xff}\}$.

Note that the value of 0x62 is excluded from the key search space. Then, the total number of key candidate values is reduced from 256 to 255. Therefore, by repeating this analysis process, the size of the key search space can be reduced to one and eventually the actual key byte can be deduced.

2.3 Algebraic Cryptanalysis

In algebraic cryptanalysis [CMR06], plaintexts and ciphertexts are related through a system of polynomial equations. A system of equations on GF(2) can be constructed with binary variables. The values (0 or 1) from plaintexts and ciphertexts that are known can be assigned to those variables. The system is then attempted to be solved. If a solution is found, the secret key can be deduced from the solution. However, finding a solution of the system is not easy for those mainstream ciphers. This is because they are designed to break the correlation between plaintexts and ciphertexts, making it difficult to solve the system within an affordable time duration.

There are several techniques that can be used to solve an equation system, which are based on different mathematical problems. One is to solve the problem based on Grobner basis [BFS04]. Another one is based on the linearization principle (XL) [Cou04]. In this paper, we mainly focus on the satisfiability problem (SAT). Solving the SAT problem can be considered as one of the most efficient techniques for algebraic cryptanalysis [SNC09].

Using those SAT machine solvers such as CryptoMiniSAT, the system of equations is converted into a set of clauses that constitute equivalent SAT instances. Without loss of generality, the equations in SAT can be assumed with a format of *Conjunctive Normal Form* (CNF). This means that the formula includes a set of clauses, all of which are concatenated by conjunctions $\wedge$. Each clause consists of a set of literal, and each literal is either a positive or a negative variable. The literals are concatenated by disjunctions $\vee$.

2.4 Notations

Table 2 defines the notations used in this paper.

Table 2: Notations used in this paper.

Notations	Definitions	Notations	Definitions	Notations	Definitions
n	The block size	AK	Key addition	N_c	The number of ciphertexts
w	The element size	AC	Constant addition	N_r	The depth of fault leakages
r	The r-th round	SB	Substitution layer	N_e	The total number of equations
R	The total number of rounds	PL	Permutation layer	ϕ	The total number of fault leakages
$\boldsymbol{X}^r$,$\boldsymbol{Y}^r$	The r-th round data block	PL^{-1}	Inverse permutation layer	ψ	The fault leakage exploitation rate
X_i^r,Y_i^r	The i-th element of $\boldsymbol{X}^r$,$\boldsymbol{Y}^r$	S	The S-box	N_{AK}	The number of equations for AK
x_i^r,x_i^r	The i-th bit of $\boldsymbol{X}^r$,$\boldsymbol{Y}^r$	S'	The faulty S-box	N_{AC}	The number of equations for AC
$\boldsymbol{K}^r$	The r-th round key	$\tilde{Y}_i$,$\tilde{K}_i$	The i-th element of $\mathsf{PL}^{-1}(\boldsymbol{Y}^r)$, $\mathsf{PL}^{-1}(\boldsymbol{K}^r)$	N_{SB}	The number of equations for SB
$\boldsymbol{C}$	The ciphertext	f	The fault value	N_{PL}	The number of equations for PL
F	The round function	l	The fault location	N_{CA}	The number of equations for the constraint

3 Overview

In this section, we will introduce the fault model, motivation and our core idea for the proposed *Algebraic Persistent Fault Analysis* (APFA).

3.1 Fault Model

The fault model of APFA is exactly the same as that of the original PFA, as mentioned in Section 2. In this paper, we focus our discussion on the scenario of single-fault injection. Although the injection is performed only once, the fault persists for multiple rounds. Accordingly, many outputs of the substitution layer, *i.e.,* S-box in different rounds, might be erroneous. The value distribution of the S-box output in the last round has been analyzed extensively in classical PFA [ZLZ+18, ZZJ+20] and SIFA [DEK+18], and that

in the penultimate round is analyzed in EPFA [XZY+20]. However, there are still many erroneous distributions of S-box output beyond the penultimate round, which do lay there and wait for further exploitation. If these fault leakages can be fully utilized, there is no need for additional injections, making it possible to break the cipher with only one injection. These leakages in the inner rounds are what we call *"free fault leakages"*. Such utilization is termed as *"deep exploitation"* as it penetrates into those deep rounds.

3.2 Motivation and Challenges

When the fault analysis such as DFA meets the deep rounds, the fault difference becomes increasingly complex along the propagation path, requiring a very complicated analysis. Instead, AFA introduces the algebraic technique and relies on machine solvers to cope with the complex fault difference. Due to the automation brought from those machine solvers, AFA can handle the fault injection at deeper rounds and require fewer faulty ciphertexts (*i.e.,* require a less number of injections under transient fault scenario) in comparison with DFA. Many existing works are pursuing the desired case where only one injection is required. In [CJW10], the DES key can be recovered with only one fault injection in 0.01 hour. Zhao *et al.* in [ZGZ+12] used only one fault injection to recover the master key of LED in less than one minute. However, when the round is to be injected with fault goes too deep, the corresponding complexity of fault analysis becomes too high to afford even for an algebraic machine solver. One reluctant solution for the compensation is to increase the required number of ciphertext, which is equivalent to increasing the number of fault injections in AFA under the transient fault scenario.

Therefore, there is one dilemma that we are facing with. On one side, under the transient fault scenario, AFA itself (not to mention DFA) has certain limitations. Sometimes it is difficult to cope with the injection in deep rounds, so we have to increase the number of injections to balance such trade-off. On the other side, under the persistent fault scenario, the single fault injection that persists multiple rounds (actually full rounds) is desired. However, there is no satisfactory solution on how to utilize those *"free fault leakages"* in deep rounds. When taking algebraic techniques into consideration, how to interpret those persistent fault leakages with algebraic equations is also never explored. Such dilemma motivates us to take a further stride towards the automation of PFA using machine solver. Most importantly, the new analysis method (termed as APFA in this paper) should step beyond the penultimate round and reach those *"free fault leakages"* in further deeper rounds, such as the antepenultimate round.

3.3 Core Idea

When facing multiple rounds, we propose the so-called *Algebraic Persistent Fault Analysis* (APFA in short) to combine the persistent analysis with the algebraic analysis.

The core idea is to find a general algebraic representation for the persistent fault that is already injected. Such algebraic representation should hold two properties. First, it should be able to connect the faulty output of **each** substitution layer, therefore taking the usage of free fault leakage. Second, it should be general enough for the simplified representation. The specific expression should not depend on those algebraic representations of next rounds. It should be able to be applied as the constraint only to the current round keys, leaving the machine solver to cope with the interconnection between round keys (by building equations for the key schedule).

Suppose the block cipher $\mathbb{B}$ has R rounds. Figure 1 shows an example of analyzing the r-th intermediate round ($1 \leq r \leq R$). A smaller value of r indicates that the corresponding analysis goes deeper. Suppose the original value of the fault S-box S' is known to be V. In the r-th round function, $\boldsymbol{X}^r$ is XORed with $\boldsymbol{K}^r$ after SB' and PL, *i.e.,*

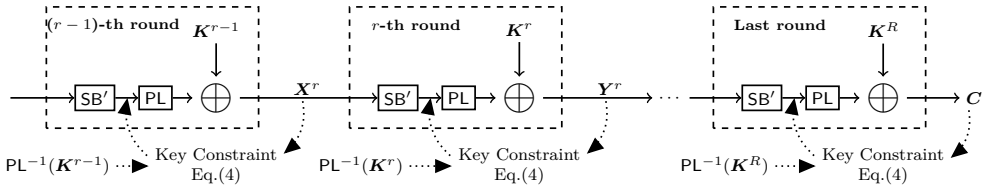

Figure 1: An illustrative SPN block cipher with multiple rounds.

$\boldsymbol{Y}^r = \mathsf{PL}(\mathsf{SB}^{'}(\boldsymbol{X}^r)) \oplus \boldsymbol{K}^r$. $\boldsymbol{Y}^r$ contains the information of both $\mathsf{SB}^{'}(\boldsymbol{X}^r)$ and $\boldsymbol{K}^r$, which means that we can use $\boldsymbol{Y}^r$ to add new constraints to $\boldsymbol{K}^r$, as shown in Eq.(3).

$$\left.\begin{array}{c} \mathsf{S}^{'}[X_i^r] \neq V \\ \boldsymbol{Y}^r = \mathsf{PL}(\mathsf{SB}^{'}(\boldsymbol{X}^r)) \oplus \boldsymbol{K}^r \\ \mathsf{PL}^{-1}(\boldsymbol{Y}^r) = \mathsf{SB}^{'}(\boldsymbol{X}^r) \oplus \mathsf{PL}^{-1}(\boldsymbol{K}^r) \end{array}\right\} \Longrightarrow \tilde{K}_i^r \neq \tilde{Y}_i^r \oplus V, \quad 0 \leq i < \frac{n}{w} \tag{3}$$

In Eq.(3), X_i^r and $\tilde{Y}_i^r$ are the w bits of $\boldsymbol{X}^r$ and $\mathsf{PL}^{-1}(\boldsymbol{Y}^r)$, respectively. $\tilde{K}_i^r$ is the w bits of $\mathsf{PL}^{-1}(\boldsymbol{K}^r)$ and V is a constant. Eq.(3) is independent to the specific value of round r, thus it can be generally represented by the variable r, which is the essence of our APFA. $\tilde{K}_i^r$ is the only key variable of $\boldsymbol{K}^r$ involved in Eq.(3) for the round key $\boldsymbol{K}^r$, and there is no key variable for any other round keys. $\boldsymbol{Y}^r$ is a newly introduced intermediate variable. Eq.(3) is a general formula that can be applied to multiple rounds. The original PFA only applies this formula to the last round (*i.e.,* $r = R$), and assigns the specific value $\boldsymbol{C}$ from ciphertext to $\boldsymbol{Y}^R$. In fact, Eq.(2) can be considered as a special case for Eq.(3) where $V = 0\text{x}63$ and $r = R$.

The beauty of Eq.(3) lies in its simplicity. When applied to different rounds, the only difference is the round index r. This allows us to easily construct new constraints with a generic representation. Recall that in traditional DFA, those constraints to be added normally involve the keys in the next round(s), which become exponentially complicated when the analysis round is getting deeper.

The magic of Eq.(3) that could work well lies in the art of algebraic techniques. On one side, literally, there are no round keys other than $\boldsymbol{K}^r$ in Eq.(3), which is easy for the extension to deep rounds. On the other side, the relationship between $\boldsymbol{K}^r$ and $\boldsymbol{K}^{r+1}$ has been already constructed when we build up the equation system for both the encryption and the key schedule, especially the latter. Such constraints are naturally added without extra manual work.

In practice, the adversary $\mathcal{A}$ can obtain multiple faulty ciphertexts $\boldsymbol{C}$, and introduce both the intermediate variable $\boldsymbol{Y}^r$ and the round key $\boldsymbol{K}^r$ which are unknown. Then, $\mathcal{A}$ establishes algebraic equations for both the encryption and the key schedule. The general constraint Eq.(3) can also be transformed into an algebraic equation, which can be added to the equation set for the encryption and the key schedule. Finally, $\mathcal{A}$ attempts to solve a new system with more algebraic equations, and the key is determined by the final solution. We will discuss the implementation details of APFA in the next section.

4 Algebraic Persistent Fault Analysis

In this section, we will introduce the algebraic design of general SPN-based block cipher, as well as detail how APFA can cope with fault leakages in multiple rounds.

4.1 Algebraic Design of General SPN Block Ciphers

We elaborate the algebraic representation of three major operations, which are the normal components used in SPN block ciphers.

Suppose $\boldsymbol{X}$ represents a data block. x_i and X_i are the single bit and the element (w bits) of $\boldsymbol{X}$, respectively, whose relationship can be described as follows.

$$\boldsymbol{X} = \{X_0, X_1, \cdots, X_{\frac{n}{w}-1}\} = \{x_0 \| x_1 \| \cdots \| x_{w-1}, \cdots, x_{n-w-1} \| x_{n-w} \| \cdots \| x_{n-1}\} \tag{4}$$

"$\cdot$" means the bitwise AND. For the sake of simplicity, "$\cdot$" can be omitted sometimes. For example, $x_i \cdot x_j$ can be simplified as $x_i x_j$.

4.1.1 Representation of Addition

The general addition component is the combination of the input data block $\boldsymbol{X}$ and the key $\boldsymbol{K}$ to get the output data block $\boldsymbol{Y}$. The representation of them is shown in Eq.(5), where x_i and y_i are one bit of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. k_i is used as the one key bit in Eq.(5). "+" denotes the bitwise exclusive or (XOR).

$$x_i + k_i + y_i = 0 \ , \ 0 \leq i < n \tag{5}$$

As for the addition operation, k_i may be a constant in the encryption or the key schedule. This case can be denoted as AC, which is slightly different from AK. It can be represented as shown in Eq(6), where c_i is one bit of the constant that is publicly known.

$$x_i + c_i + y_i = 0 \ , \ 0 \leq i < n \tag{6}$$

For either AK or AC, the number of equations (denoted as N_{AC} and N_{AK}) that are generated is equal to the block size n, *i.e.,* $N_{\mathrm{AK}} = N_{\mathrm{AC}} = n$.

4.1.2 Representation of Substitution Layer

The substitution layer SB is the only non-linear structure in SPN structure. It maps the input $\boldsymbol{X}$ to the output $\boldsymbol{Y}$ using S-box S. An S-box can be organized as a truth table for input and output, each of which has 2^w variables. Truth table is a representation of Boolean function, and it cannot be directly utilized by algebraic solver. However, according to [KM10], the truth table can be transformed into an *Algebraic Normal Form* (ANF) first, which can be later re-transformed to CNF and fed into the general SAT solvers such as CryptoMiniSAT. Suppose $X_i = \{x_0 \| x_1 \| \cdots \| x_{w-1}\}$ and $Y_i = \{y_0 \| y_1 \| \cdots \| y_{w-1}\}$ are the input and output of S, respectively. The relationship between X_i and Y_i can be expressed by ANF as shown in Eq.(7),

$$y_i = \sum_{t=0}^{2^w-1} a_t x_0^{t_0} x_1^{t_1} \cdots x_{w-1}^{t_{w-1}} \tag{7}$$

where $t = \{t_0 \| t_1 \| \cdots \| t_i \| \cdots \| t_{w-1}\}$ and t_i is one bit of t. a_t is an ANF coefficient where $a_t \in \{0, 1\}$. More specifically, $x_i^{t_i}$ ($t_i = 1$) means the bit x_i does appear in $a_t x_0^{t_0} x_1^{t_1} \cdots x_{w-1}^{t_{w-1}}$, and $x_i^{t_i}$ ($t_i = 0$) means x_i does not appear.

Taking the lightweight block cipher PRESENT as an example, its S-box ($w = 4$) can be denoted by ANF as shown in Eq.(8) where the AND operations "$\cdot$" in $x_i \cdot x_j$ is omitted.

$$\begin{aligned} y_0 &= x_0 + x_2 + x_3 + x_1 x_2 \\ y_1 &= x_1 + x_3 + x_1 x_3 + x_2 x_3 + x_0 x_1 x_2 + x_0 x_1 x_3 + x_0 x_2 x_3 \\ y_2 &= 1 + x_2 + x_3 + x_0 x_1 + x_0 x_3 + x_1 x_3 + x_0 x_1 x_3 + x_0 x_2 x_3 \\ y_3 &= 1 + x_0 + x_1 + x_3 + x_1 x_2 + x_0 x_1 x_2 + x_0 x_1 x_3 + x_0 x_2 x_3 \end{aligned} \tag{8}$$

The above ANF contains bitwise AND operations, whereas general SAT solvers (such as CryptoMiniSAT) only support CNF and bitwise XOR operations. As a result, an intermediate variable [SNC09] can be employed to represent those items (like $x_0x_1x_2$ in y_1) that have AND operations. Suppose there is the intermediate variable I_t where $I_t = x_0x_1 \cdots x_{w-1}$. I_t can be expressed in the form of CNF representation as shown in Eq.(9), where $\bar{I}_t$ and $\bar{x}_i$ are complements of I_t and x_i, respectively.

$$(x_0 \vee \bar{I}_t) \wedge (x_1 \vee \bar{I}_t) \wedge \cdots \wedge (x_{w-1} \vee \bar{I}_t) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee \cdots \vee \bar{x}_{w-1} \vee I_t) = 1 \tag{9}$$

Constrained by Eq.(9), for different values of x_i, the result of I_t will be the same as that of $x_0x_1 \cdots x_{w-1}$. Therefore Eq.(9) can be added to the equation system to be sent to the solver, where I_t can be used to represent the item $x_0x_1 \cdots x_{w-1}$. Moreover, a specific I_t can correspond to the t-th item in Eq.(7). For example, y_1 in Eq.(8) can be represented using Eq.(11) where the t in I_t ranges over $\{2, 4, 9, 10, 11, 12, 13\}$.

$$y_1 = I_2 + I_4 + I_9 + I_{10} + I_{11} + I_{12} + I_{13} \tag{10}$$

We can further simplify Eq.(10) by replacing it with the index t in the intermediate variable I_t as shown in Eq.(11). Since the fault in the S-box may be different, the indexes in Eq.(11) will also be different. For the sake of generality, each item ($1 \leq t < 2^w$) in Eq.(7) needs to create intermediate variables to cope with varied indexes.

$$\begin{gathered} y_0 : \{1, 3, 4, 8\} \\ y_1 : \{2, 4, 9, 10, 11, 12, 13\} \\ y_2 : \{0, 3, 4, 5, 7, 9, 12, 13\} \\ y_3 : \{0, 1, 2, 4, 8, 11, 12, 13\} \end{gathered} \tag{11}$$

For different faulty S-boxes, they have to be preprocessed as in Eq.(11). Then, the equations can be constructed for the input X_i and output Y_i of the S-box based on the indexes in Eq.(11) as described previously. This part will produce $2^w - 1$ intermediate variables for each X_i. The power of I_t means the number of x_i. When the power of I_t is j, it needs $j + 1$ CNF equations. There are $\binom{w}{j}$ intermediate variables with the power of j. Therefore, for an SB operation, the number of equations (N_{SB}) that generated can be calculated as shown in Eq.(12), where the first item (*i.e.,* n) is the number of equations that are required for the newly added variable y_i and the second item is number of equations required for the intermediate variables I_t.

$$N_{\mathrm{SB}} = n + \frac{n}{w} \times \sum_{j=1}^{w} (j+1) \times \binom{w}{j} = \frac{n(w+2)}{w} \times 2^{w-1} + \frac{n(w-1)}{w} \tag{12}$$

4.1.3 Representation of Permutation Layer

The permutation layer PL plays a role of diffusion in SPN block ciphers. According to the number of input bits to one output bit, PL can be divided into two types: bit-based permutation (one bit to one bit) and MDS matrix multiplication (multiple bits to one bit) where MDS stands for *Maximum Distance Separable*. Typical example of the former can be ShiftRows in AES and BitPermutation in PRESENT. That of the latter can be MixColumns in AES and MixColumnsSerial in LED.

For the bit-based permutation, the relationship between the input bit x_i and the output bit y_i can be represented by a permutation table T_P. For example, the 64-bit T_P for PRESENT (*i.e.,* 16 nibbles) is $[0, 16, 32, 48, 1, 17, \cdots, 15, 31, 47, 63]$. Then the bit-based permutation in PL can be expressed as two variants as shown in Eq.(13) and Eq.(14), which are mutually inverse operations. Different ciphers may choose one of Eq.(13) and Eq.(14) as the equation for the permutation layer and use the other as that for the inversive

permutation. Since the bit-based permutation does not involve any intermediate variables, both the number of new variables representing y_i and the number of equations (N_{PL}) connecting x_i and y_i are equal to the block size n, *i.e.,* $N_{\mathrm{PL}} = n$.

$$x_i + y_{T_P[i]} = 0 \ , \ 0 \leq i < n \tag{13}$$

$$x_{T_P[i]} + y_i = 0 \ , \ 0 \leq i < n \tag{14}$$

For the MDS matrix multiplications, most of them are related to multiplication operations on finite fields. For example, MixColumnsSerial in the LED uses multiplication operations based on an irreducible polynomial $x^4 + x + 1$ as shown in Eq.(15).

$$\begin{bmatrix} Y_0 & Y_1 & Y_2 & Y_3 \\ Y_4 & Y_5 & Y_6 & Y_7 \\ Y_8 & Y_9 & Y_{10} & Y_{11} \\ Y_{12} & Y_{13} & Y_{14} & Y_{15} \end{bmatrix} = \begin{bmatrix} 0x4 & 0x1 & 0x2 & 0x2 \\ 0x8 & 0x6 & 0x5 & 0x6 \\ 0xb & 0xe & 0xa & 0x9 \\ 0x2 & 0x2 & 0xf & 0xb \end{bmatrix} \begin{bmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{bmatrix} \tag{15}$$

As shown in Eq.(15), the first element Y_0 can be represented as:

$$Y_0 = 0x4 \times X_0 + 0x1 \times X_4 + 0x2 \times X_8 + 0x2 \times X_{12} \tag{16}$$

where "$\times$" means the multiplication in GF(2^4).

The key point when using MDS matrix multiplication is to represent the multiplication of an element in GF(2^4) with a constant. Taking $0x4 \times X_i$ as an example, where the value of X_i ranges from 0x0 to 0xf. First, the values of X_i can be enumerated, and the corresponding result of $0x4 \times X_i$ can also be calculated. Similarly, the relationship between X_i and $0x4 \times X_i$ can be represented by a truth table. This truth table can be processed by the method in Section 4.1.2. For all the constants involved in matrix, this process can be repeated for several times. Finally, the relationship between the multiplication in Eq.(15) and the input X_i is shown in the Table 3.

Table 3: LED's Multiplication over GF(2^4).

	y_0	y_1	y_2	y_3		y_0	y_1	y_2	y_3
0x2	x_3	$x_0 + x_3$	x_1	x_2	0x9	$x_0 + x_1$	x_2	x_3	x_0
0x3	$x_0 + x_3$	$x_0 + x_1 + x_3$	$x_1 + x_2$	$x_2 + x_3$	0xa	$x_1 + x_3$	$x_0 + x_1 + x_2 + x_3$	$x_1 + x_2 + x_3$	$x_0 + x_2 + x_3$
0x4	x_2	$x_2 + x_3$	$x_0 + x_3$	x_1	0xb	$x_0 + x_1 + x_3$	$x_0 + x_2 + x_3$	$x_1 + x_3$	$x_0 + x_2$
0x5	$x_0 + x_2$	$x_1 + x_2 + x_3$	$x_0 + x_2 + x_3$	$x_1 + x_3$	0xc	$x_1 + x_2$	$x_1 + x_3$	$x_0 + x_2$	$x_0 + x_1 + x_3$
0x6	$x_2 + x_3$	$x_0 + x_2$	$x_0 + x_1 + x_3$	$x_1 + x_2$	0xd	$x_0 + x_1 + x_2$	x_3	x_0	$x_0 + x_1$
0x7	$x_0 + x_2 + x_3$	$x_0 + x_1 + x_2$	$x_0 + x_1 + x_2 + x_3$	$x_1 + x_2 + x_3$	0xe	$x_1 + x_2 + x_3$	$x_0 + x_1$	$x_0 + x_1 + x_2$	$x_0 + x_1 + x_2 + x_3$
0x8	x_1	$x_1 + x_2$	$x_2 + x_3$	$x_0 + x_3$	0xf	$x_0 + x_1 + x_2 + x_3$	x_0	$x_0 + x_1$	$x_0 + x_1 + x_2$

Each output Y_i is associated with four intermediate variables to represent the multiplication of the input and the constant. This will generate $4w$ equations for each Y_i. Therefore, for the MDS matrix multiplication, the total number of equations (N_{PL}) can be calculated as shown in Eq.(17), where n is the block size. The first item (*i.e.,* $n/w \times 4w$) in Eq.(17) is the number of equations required for intermediate variables in each Y_i, and the second item is that are required for the newly added variable Y_i.

$$N_{\mathrm{PL}} = \frac{n}{w} \times 4w + n = 5n \tag{17}$$

4.2 Fault Leakage Exploitation for Multiple Rounds

The adversary $\mathcal{A}$ will build the algebraic equations of the encryption for the faulty ciphertext and analyze its fault leakage of multiple rounds. For each round, $\mathcal{A}$ can build a group of algebraic constraint equations based on the constant $V = \{v_0 \| v_1 \| \cdots \| v_{w-1}\}$. Let us take the output $\boldsymbol{Y}^r$ of the r-th round AK and the round key $\boldsymbol{K}^r$ as an example to build the constraint equations. According to Section 3.3, $\tilde{Y}_i^r$ and $\tilde{K}_i^r$ are the element of $\mathsf{PL}^{-1}(\boldsymbol{Y}^r)$ and $\mathsf{PL}^{-1}(\boldsymbol{K}^r)$, respectively, which can be represented as $\tilde{Y}_i^r = \{\tilde{y}_0 \| \tilde{y}_1 \| \cdots \| \tilde{y}_{w-1}\}$ and

$\tilde{K}_i^r = \{\tilde{k}_0 \| \tilde{k}_1 \| \cdots \| \tilde{k}_{w-1}\}$. According to Eq.(3), a set of new constraint equations can be constructed as shown in Eq.(18) and Eq.(19), where d_i is an intermediate variable.

$$d_i + \tilde{y}_i + \tilde{k}_i + v_i = 0 \ , \ 0 \leq i < w \tag{18}$$

$$d_0 \vee d_1 \vee \cdots \vee d_{w-1} = 1 \tag{19}$$

d_i in Eq.(18) can be regarded as one bit of $\tilde{K}_i^r \oplus \tilde{Y}_i^r \oplus V$. Since $\tilde{K}_i^r \neq \tilde{Y}_i^r \oplus V$, the result of $\tilde{K}_i^r \oplus \tilde{Y}_i^r \oplus V$ is not equal to 0. Therefore, there should be at least one bit-1 among $d_0, d_1, \cdots, d_{w-1}$, which is represented in Eq.(19). $\tilde{Y}_i^r \oplus V$ can be removed from the key search space by using these two constraints.

In short summary, for the collected N_c faulty ciphertexts, each ciphertext utilizes the last N_r rounds of fault leakages. Then, using the values of collected faulty ciphertexts, we can build N_c sets of equations (one set of equations for one ciphertext), and assign values to the variables (retrieved from ciphertexts) in the representation equations. When building equations for each ciphertext, the corresponding constraint equations are added in each round at the same time. Note that those constraint equations are with the same expression, therefore such a technique can avoid the manual analysis for each round and possibly utilize the fault leakages in deep rounds. The limit of such technique is then left to the computation power of the solver itself.

4.3 APFA in a Nutshell

The application of APFA on multiple rounds can be described by Algorithm 1. $\mathbb{C}$ denotes the sets for faulty ciphertexts. N_r denotes the depth of fault leakage exploitation. `GenKSR` generates the equations for the round key. `GenAK`, `GenSB`, `GenPL` and `GenInvPL` generate the equations for the addition, the substitution, the permutation and the inverse permutation, respectively. `GenConst` adds constraints to the round key. The output variables in the last round are assigned with real values in faulty ciphertexts (in Line 17). `RunAPFA` will take all the algebraic equations as inputs and send them to the SAT solver. Overall, the solving could finish in a reasonable time and return the secret key. Otherwise, it means that the current constraints are insufficient to recover the key. The specific implementation of the sub-functions in Algorithm 1 has been clarified in the previous sections.

Algorithm 1: APFA on block ciphers using multiple rounds of fault leakages.

```
   input  : ℂ, l, f, N_r
   output : K
 1 V = S[l] ;                                   // Get the original value of the S.
 2 S'[l] = S[l] ⊕ f;
 3 for r = 1; r ≤ R; r++ do
 4 |   GenKSR(r, K^r) ;                  // Generate the equations for the round key.
 5 end
 6 for r = 1; r ≤ R; r++ do
 7 |   K̃^r=GenInvPL(K^r) ;                // Generate the equations for PL^{-1}(K^r).
 8 end
 9 for C ∈ ℂ do
10 |   for r = R − N_r; r ≤ R; r++ do
11 |   |   GenSB(X^r);
12 |   |   GenPL(X^r);
13 |   |   GenAK(X^r, K^r);
14 |   |   X̃^r=GenInvPL(X^r);
15 |   |   GenConst(X̃^r, K̃^r, V) ;          // Add constraints to the round key.
16 |   end
17 |   X^R = C ;                                        // X^R is assigned with C.
18 end
19 K=RunAPFA()
```

4.4 Evaluation Metrics

This section will introduce several metrics to evaluate APFA.

(1) N_c, which refers to the number of faulty ciphertexts used by the adversary. (2) N_r, which refers to the depth of fault leakage exploitation (the number of last rounds with added constraints). N_c and N_r have a negative correlation, that is, N_c will decrease when N_r increases. (3) N_e, which refers to the total number of equations when one round of fault leakage is used. (4) ϕ, which refers to the total number of utilized fault leakages for the block cipher. Its value is the product of N_c and N_r, *i.e.,* $N_c \times N_r$. ϕ can be regarded as N_c required by the original PFA, due to PFA only uses the last round fault leakages (ciphertexts). (5) ψ, the fault leakage exploitation rate, which can be calculated as the number of the equations that can be managed to reduce the key search space (Eq.(18) and Eq.(19)) over the total number of equations in one round. ψ can be used to describe the analysis difficulty of the target cipher. Then we mainly introduce N_e and ψ.

For N_e, without loss of generality, the block ciphers $\mathbb{B}$ contains AK, SB, and PL in one round. Therefore, the composition of N_e includes the number of equations generated by these three operations, which can be referred to Section 4.1. In addition, adding constraints also requires to generate a certain amount of equations, which is denoted as N_{CA}. Then, N_e can be calculated as shown in Eq.(20).

$$N_e = N_{\mathrm{AK}} + N_{\mathrm{SB}} + N_{\mathrm{PL}} + N_{\mathrm{CA}} \tag{20}$$

According to Section 4.2, N_{CA} is composed of the equations produced by $\mathsf{PL}^{-1}(\boldsymbol{Y}^r)$, *i.e.,* Eq.(18) and Eq.(19). The number of equations of $\mathsf{PL}^{-1}(\boldsymbol{Y}^r)$ is equal to N_{PL}. Eq.(18) is an XOR operation, which generates w equations for each element. Since there are n/w elements, the total number of equations can be calculated as $w \times n/w$, which is equal to the block size n. Eq.(19) is a clause composed of disjunctions. For each element, only one equation is needed, and there are n/w elements. Therefore, $N_{\mathrm{CA}} = N_{\mathrm{PL}} + n + n/w$.

The constraints of Eq.(18) and Eq.(19) can be used to reduce one candidate from the key search space. Therefore, the total number of their equations in one round can depict the difficulty of the analysis, which is denoted as ψ, as shown in Eq.(21). The larger ψ is, the easier one impossible value of the element can be reduced from the key search spaces.

$$\psi = \frac{n + \frac{n}{w}}{N_e} \times 100\% = \frac{n(w+1)}{wN_e} \times 100\% \tag{21}$$

5 APFA with Unknown Fault Scenarios

In Section 3 and 4, we introduce APFA based on the assumption that both the fault value f and fault location l are known, which may not hold in practice. In this section, we will introduce the unknown fault scenarios.

5.1 APFA with Unknown Fault Value f

First, let us recall the fault model. The adversary $\mathcal{A}$ will inject a persistent fault into S (it will persist for multiple rounds and encryptions). Suppose that the fault value f changes S to S' at the location l (l-th element of S-box), whose corresponding value changes from V to U. Then, $\mathcal{A}$ will collect multiple faulty ciphertexts for analysis.

In practice, f is possibly unknown to $\mathcal{A}$. In Eq.(1), it was known that the distribution of S' is unbalanced. This will lead to an unbalanced distribution of C_i as well. When analyzing the distribution of C_i, we will get the value that should never appear in C_i, which is denoted as C_i^{min} and $C_i^{min} = V \oplus K_i$. Moreover, there is another value that appears with doubled frequency, which is denoted as C_i^{max} and $C_i^{max} = U \oplus K_i$.

Since $U = V \oplus f$, C_i^{min} and C_i^{max} can be simply XORed to obtain f, which does not require the fault value f to be known. However, there is a prerequisite that the number of ciphertexts to be analyzed needs to be enough. Typically thousands of ciphertexts are desired. When pursuing the least number of ciphertexts that are required, the technique of *Maximum Likelihood Estimation* (MLE) in [ZZJ+20] can be adopted.

MLE is a method of maximizing a likelihood function to estimate the parameters of a probability distribution. MLE can be used to estimate the value of f from all n/w elements of each collected ciphertext, which greatly reduces the number of required ciphertexts. And more details can be found in [ZZJ+20]. Experimental results regarding the fault value f will be analyzed in Section 7.

5.2 APFA with Unknown Fault Location l

When the fault location is unknown, it is still possible to solve the problem using the exhaustive searching for l. For a block cipher, there are 2^w positions for one specific element in $\mathsf{S}^{'}$. A very straightforward method is to enumerate l from 0 to $2^w - 1$, which can be directly supported by the SAT solver. Note that, before the guess on l, the fault value f needs to be determined first. As mentioned in Section 4.1.2, given a specific value f and l (f is already determined and l is assumed as a guess), an equation for $\mathsf{S}^{'}$ can be constructed as shown in Eq.(11). For each enumerated guess on l, we can build the equations according to the steps in Algorithm 1 and let the solver cope with it. When the guess l is wrong, the clause corresponding to Eq.(19) of different ciphertexts will conflict, causing the solver to return an unsolvable report within a few minutes. When the guess l is correct, the solver will return a set of variable values that satisfy the equations, and the value of the key is also in it.

Based on this approach, the number of ciphertexts N_c will affect the enumeration of l. The premise for the solver to return an unsolvable report is that there is a conflict in the equations. When N_c is fewer, the constraint may be insufficient to induce a conflict, and the solver will possibly return a set of variables even with different l (Theoretically only one l is solvable). To avoid this problem, we can add more constraint equations by increasing the depth of the analysis round.

6 APFA on Lightweight Block Ciphers

In this section, we apply APFA to lightweight block ciphers, which include PRESENT, LED and SKINNY. Typically, lightweight block ciphers are constructed with a relatively simple round function, however, a relatively large number of rounds are required to increase their security against different cryptanalyses such as differential and linear cryptanalysis. Since the smallest element of most lightweight block ciphers is nibble ($w = 4$), the number of faulty ciphertexts that are required and the equations that are generated could be less compared with classical ciphers, and the equations generated are smaller in scale. Since there are more rounds of encryption, more free fault leakages could be available for deep analysis. Therefore, the proposed APFA is very suitable for lightweight block ciphers.

6.1 PRESENT Block Cipher

PRESENT is an SPN-based lightweight block cipher proposed by Bogdanov *et al.* in 2007 [BKL+07]. The block size of PRESENT is 64 bits and each element is stored in nibble. PRESENT has two versions: PRESENT-80 and PRESENT-128. PRESENT-128 cannot be attacked by the original PFA due to the cipher complexity. Therefore, in this subsection we focus on the APFA on PRESENT-128 to show the efficiency.

The round function of PRESENT is composed with **addRoundKey** (AK), **sBoxLayer** (SB) and **pLayer** (PL). There are 31 rounds in total. The key schedule used in PRESENT-128 will generate a 64-bit round key from the 128-bit master key. The round function F can be represented as shown in Eq.(22), where $R = 31$.

$$\mathsf{F} = \begin{cases} \mathsf{PL}(\mathsf{SB}(\mathsf{AK}(\boldsymbol{X}^r, \boldsymbol{K}^r))), & 1 \le r < R \\ \mathsf{AK}(\mathsf{PL}(\mathsf{SB}(\mathsf{AK}(\boldsymbol{X}^r, \boldsymbol{K}^r))), \boldsymbol{K}^{r+1}), & r = R \end{cases} \tag{22}$$

6.1.1 Representation of the Round Function

Representing AK: A 64-bit input $\boldsymbol{X}$ is XORed with the 64-bit round key $\boldsymbol{K}$, which can be represented as Eq.(5). $n = 64$.

Representing PL: The 64-bit input $\boldsymbol{X}$ is permuted through a permutation table T_P. The i-th bit of $\boldsymbol{X}$ is moved to bit position $T_P[i]$. The permutation for PL in PRESENT is bit-based, therefore Eq.(13) can be used to represent PL. The specific mapping rule T_P is shown in Eq.(23). Similarly, PL^{-1} moves the bit of $T_P[i]$ to the position i, which can be represented by Eq.(14).

$$T_P[i] = \begin{cases} 16 \times i \bmod 63, & 0 \le i < 63 \\ 63, & i = 63 \end{cases} \tag{23}$$

Representing SB: Each element X_i of the 64-bit input $\boldsymbol{X}$ is substituted through the faulty S-box S'. The fault location of S' is l, and the fault value is f. After determining the values of l and f, we can construct the equations for S' as shown in Eq.(24) where specific values of 0 and 0xc are assigned to l and f for the purpose of illustration.

$$\begin{gathered} y_0 : \{1, 3, 4, 8\} \\ y_1 : \{2, 4, 9, 10, 11, 12, 13\} \\ y_2 : \{1, 2, 6, 8, 10, 11, 14, 15\} \\ y_3 : \{3, 5, 6, 7, 9, 10, 14, 15\} \end{gathered} \tag{24}$$

F will generate $N_{\mathrm{AK}} + N_{\mathrm{SB}} + N_{\mathrm{PL}} = 944$ equations for each round.

6.1.2 Representation of the Key Schedule

We will briefly introduce the key schedule of PRESENT-128. The initial master key will be stored in the register $\boldsymbol{K} = \{K_0, K_1, \cdots, K_{31}\} = \{k_0 \| k_1 \| \cdots \| k_{127}\}$, and its high 64 bits are used as the first round key $\boldsymbol{K}^1 = \{k_0^1 \| k_1^1 \| \cdots \| k_{63}^1\} = \{k_{64} \| k_{65} \| \cdots \| k_{127}\}$. After extracting the round key $\boldsymbol{K}^r$, the key register $\boldsymbol{K}$ is updated as follows:

(1) The register is cyclically shifted by 61 bits to the upper bit.
(2) $K_{31} = \mathsf{S}[K_{31}]$
(3) $K_{30} = \mathsf{S}[K_{30}]$
(4) $[k_{62}k_{63}k_{64}k_{65}k_{66}]$=$[k_{62}k_{63}k_{64}k_{65}k_{66}] \oplus$ `round_counter`

where (1) can be represented by Eq.(13), and $T_P[i] = (i + 61) \bmod 128$.

6.2 LED Block Cipher

The LED [GPPR11] is a lightweight block cipher proposed in CHES 2011, which uses an AES-like SPN structure. It does not have the key schedule, and the master key directly participates in encryption. The size of a LED block is 64 bits, and each element is stored in nibble. The state matrix is 4×4. But unlike AES, its state is loaded in rows. There are two versions of LED: LED-64 and LED-128. We mainly discuss LED-64.

The round function of LED is quite similar to AES, which includes **AddConstants** (AC), **SubCells** (SB), **ShiftRows** and **MixColumnsSerial** (PL), and **addRoundKey**

(AK). Eq.(25) represents the round function of LED-64. Different from other block ciphers, AK will be used per four rounds (*i.e.,* a step). Therefore, LED-64 has 32 rounds and only 8 of them uses the round keys. And LED performs key whitening in the last round.

$$\mathsf{F} = \begin{cases} \mathsf{PL}(\mathsf{SB}(\mathsf{AC}(\mathsf{AK}(\boldsymbol{X}^r, \boldsymbol{K}), [rc]))), & r \equiv 0 \bmod 4 \\ \mathsf{PL}(\mathsf{SB}(\mathsf{AC}(\boldsymbol{X}^r, [rc]))), & r \not\equiv 0 \bmod 4 \end{cases} \tag{25}$$

where $0 \leq r < 32$ and $[rc]$ is the round constant.

6.2.1 Representation of the the Round Function

Representing AK and AC: The state matrix of the LED is XORed with the key or constant, which can be represented by Eq.(5) and Eq.(6) respectively. Since $n = 64$, we have $N_{\mathrm{AK}} = N_{\mathrm{AC}} = 64$.

Representing SB: Due to the fact that LED and PRESENT use the same S-box, this representation for LED can be referred to Section 6.1. And $N_{\mathrm{SB}} = 816$.

Representing PL: PL of LED differs from PRESENT, which includes **ShiftRows** and **MixColumnsSerial**. **ShiftRows** is a bit-based permutation, which can be represented by Eq.(13). It takes a matrix as input and shifts the state matrix i-th row cyclically by i elements. Therefore, T_P could be represented as shown in Eq.(26).

$$T_P[i] = \left\lfloor \frac{i}{16} \right\rfloor \times 16 + (i - \left\lfloor \frac{i}{16} \right\rfloor \times 16 + \left\lfloor \frac{i}{16} \right\rfloor \times 12) \bmod 16, \ 0 \leq i < 64 \tag{26}$$

MixColumnsSerial in LED is a MDS matrix multiplication, whose representation can be referred to Section 4.1.3 and Table 3. $N_{\mathrm{PL}} = 64 + 320 = 384$.

F will generate $N_{\mathrm{AC}} + N_{\mathrm{SB}} + N_{\mathrm{PL}} = 1264$ equations. And a single step will generate $N_{\mathrm{AK}} + 1264 \times 4 = 5120$ equations.

6.2.2 Representation of the Key Schedule

There is no key schedule in LED-64, the master key can be represented by a 64-bit variable.

6.3 SKINNY Block Cipher

SKINNY is an SPN-based tweakable lightweight block cipher proposed in CRYPTO 2016 by Beierle *et al.* [BJK+16], which can be expressed as SKINNY-n-k according to different block size n and key size k. Specifically, it can be divided into three categories: SKINNY-n-n, SKINNY-n-2n and SKINNY-n-3n. Both SKINNY-64-64 and SKINNY-64-128 are analyzed in this paper, and SKINNY-64-64 is utilized for illustration in this subsection.

SKINNY-64-64 has 32 rounds, whose round function F consists of five operations: **SubCells** (SB), **AddConstants** (AC), **AddRoundTweakey** (AK), **ShiftRows** and **MixColumns** (PL). F can be represented as shown in Eq.(27), where $0 \leq r < 32$ and $[rc]$ is the round constant.

$$\mathsf{F} = \mathsf{PL}(\mathsf{AK}(\mathsf{AC}(\mathsf{SB}(\boldsymbol{X}^r), [rc]), \boldsymbol{K}^r)) \tag{27}$$

6.3.1 Representation of the Round Function

The round function in SKINNY is very similar to LED, except for its AK and PL.

Representing AK and AC: SKINNY also uses a state matrix to represent states. AC can be represented by Eq.(6), but AK is slightly different. SKINNY-64-64 only uses half of the round key for each round. Despite the fact that the round key contains 16 element (4×4), AK only selects eight of them (the first two rows of the matrix) in the state matrix. Therefore, AK can be represented by Eq.(28). And $N_{\mathrm{AK}} = N_{\mathrm{AC}} = 64$.

$$\begin{array}{ll} x_i + k_i + y_i = 0, & 0 \leq i < 32 \\ x_i + y_i = 0, & 32 \leq i < 64 \end{array} \tag{28}$$

Representing SB: The S-box of SKINNY is different from PRESENT and LED. For SKINNY S-box, when determining l and f, the relationship between X_i and Y_i is similar to Eq.(11), which can be derived by using the technique in Section 4.1.2. And $N_{\mathrm{SB}} = 816$.

Representing PL: The PL of SKINNY is similar to LED, which contains two types of permutation. **ShiftRows** has a different cyclic shift direction as compared to LED. Therefore it can be represented by Eq.(14). T_P is the same as Eq.(26).

For **MixColumns**, a binary matrix M multiplies each column of the state matrix. However, this step is not on the finite field. Since only the XOR operation is involved, the number of equations generated by **MixColumns** is equal to the block size n. And $N_{\mathrm{PL}} = 64 + 64 = 128$.

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{29}$$

The round function of SKINNY-64-64 can be represented with $N_{\mathrm{AK}}+N_{\mathrm{AC}}+N_{\mathrm{SB}}+N_{\mathrm{PL}} = 1072$ equations.

6.3.2 Representation of the Key Schedule

The key schedule of SKINNY-64-64 only contains permutation operations. The 64-bit master key is stored in the tweakey array TK of 16 nibbles. The round key is the first 8 nibbles of TK in each round. TK is updated as follows. A permutation P is applied to TK: for all $0 \leq i < 16$, we set $TK[i]$ with $TK[P[i]]$ where

$$P = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7]$$

Eq.(14) is also applicable in this case, where $T_P[i] = P[i/4] * 4 + i \bmod 4,\ 0 \leq i < 64$.

7 Experiment and Evaluation

In this section, we will introduce our setup and design of the experiments, which cover a variety of lightweight block ciphers.

7.1 Experiment Setup

In our experiment, we simulate fault injection via software, and use CryptoMiniSAT v5.8.0 [SNC09] to solve the algebraic equations. We implement the experiments on a PC which has 16GB memory and an AMD Ryzen 5 4600H CPU at 3.0GHz. The operating system is a 64-bit Windows 10.

The experiments generally follow the procedures below. (1) We simulate the encryption of the victim. Basically we generate random plaintexts and use the faulty S-box for encryption. (2) We collect a certain number of ciphertexts for analysis. (3) We conduct the analysis using APFA. We first construct a system of algebraic equations for both the encryption and the key schedule, and associate the actual faulty ciphertext. Then we add constraints for multiple rounds and represent them as equations. Finally we solve the algebraic system of equations using CryptoMiniSAT.

7.2 Identify the Injected Fault

Before we perform APFA on the target block cipher, we have to identify the injected fault (*i.e.,* the fault value f and the fault location l). According to Section 5.1, Maximun Likelihood Estimation (MLE) technique in [ZZJ+20] is adopted to identify f. Table 4 shows the results of MLE to estimate f for five different block ciphers. The average number

of ciphertexts that are required to infer f is listed. For the fixed l, we performed 100 simulation experiments for each f ($1 \leq f < 2^w$) and the average number is shown in Table 4. Compared with Table 1, the average number of ciphertexts to identify f is less than the number required to recover the key using APFA. In other words, even in the scenario where the f is unknown, we can use the faulty ciphertexts to recover f first before proceeding to the next analysis, where the ciphertexts that are required in APFA are already enough for deducing f. Once f is identified, the value of unknown l can also be enumerated by the solver according to the method in Section 5.2. The following subsections are based on the preprocessing that both f and l are already identified.

Table 4: The average number of ciphertexts required to obtain f.

Ciphers	PRESENT-80	PRESENT-128	LED-64	SKINNY-64-64	SKINNY-64-128
Numbers	13.5	13.4	12.9	13.3	13.4

7.3 Attack on PRESENT

In [ZZJ+20], the authors conducted PFA on PRESENT-80. When exploiting the fault leakages in the penultimate round of PRESENT, it takes about 101 faulty ciphertexts on average to recover the 80-bit key. And the authors did not manage to attack PRESENT-128 due to full key spanning over three rounds, which might be considered as a challenge in [ZZJ+20]. In APFA, we increased N_r, the number of analysis rounds for PRESENT-80 from 2 to 5. With the use of more rounds of free fault leakages ϕ, we reduced the number of faulty ciphertexts that are required to be under 20. The results are shown in Figure 2. And it is worth mentioning that APFA can extract the full key with 100% success rate. The only difference lies in the slightly different solving times.

PRESENT-128 has a lager key size than PRESENT-80. This expands the search space of the key, and the recovery of the full 128-bit master key requires at least three rounds, *i.e.,* $N_r \geq 3$. In our experiment, about 28 ciphertexts are enough to recover the master key of PRESENT-128, as shown in the blue curve in Figure 2. However, a further reduction on the fault leakage ϕ will cause the solving time to be extremely long (over $3 * 10^4$ seconds). Another important observation to note is that it seems there is a limit on the number of rounds that are required (*i.e.,* the depth N_r). More specifically, with about 35~40 ciphertexts and $N_r = 3$ ($\phi \geq 35 \times 3$), PRESENT-128 can already be broken with just a few seconds as shown in Figure 2, which means three rounds are already enough for APFA. Then in Figure 3, a further increase on ϕ (*e.g.,* N_r ranges from 3 to 10) will not reduce the solving time. Instead, it will increase the solving time (up to 250 seconds) because the solver has to spend more time processing those redundant equations from deeper rounds.

7.4 Attack on LED

The structure of LED is quite different, whose AK operation is repeated every four rounds (termed as a step) instead of every round. The constraint is added only once in one step, which requires 5584 equations. And those round functions without AK in the last step ($N_r = 4$) of the LED will generate a larger number of redundant equations, which will limit the ability of the constraint to reduce the key search space. When the analysis depth is further increased to the penultimate step ($N_r = 8$), the newly added constraints will not provide capability to reduce the key search space. Based on this fact, we can only exploit the last step, *i.e.,* $N_r = 4$. The experimental result of the LED is shown in Figure 4. With the help of the SAT solver, we reduced N_c to be under 23. It can be discovered in the orange curve in Figure 4 that there is an inflection point (*i.e.,* N_c ranges 25~30). More specifically, when N_c is less than 27, ϕ is not enough for the solver to quickly obtain the solution of key. Since only the last step can be exploited, when ϕ is insufficient, it can only rely on the search strategy of the solver itself. In general, APFA has a greater

improvement on LED compared to EPFA [XZY+20], which only needs $23/75 \approx 30.67\%$ of ciphertexts to recover the full 64-bit key.

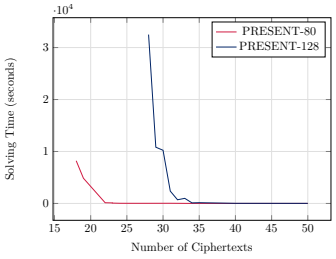

Figure 2: APFA on PRESENT-80 and PRESENT-128.

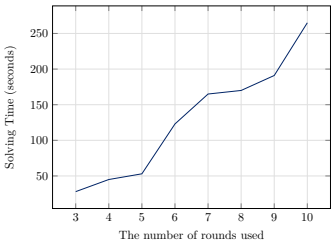

Figure 3: APFA on PRESENT-128 with 40 faulty ciphertexts.

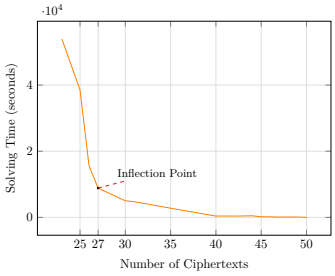

Figure 4: APFA on LED-64.

7.5 Attack on SKINNY

We attack both SKINNY-64-64 and SKINNY-64-128. As compared with LED, SKINNY contains more fault leakages due to AK operation in each round. As compared with PRESENT, the key schedule of SKINNY is simpler, which just uses permutation functions.

The red curve in Figure 5 shows that APFA has a significant improvement on SKINNY-64-64 compared to EPFA [XZY+20], which is about 155 times of reduction. Moreover, we coped with SKINNY-64-128, which cannot be handled by EPFA. For SKINNY-64-128, the round key $\boldsymbol{K}$ can be represented as $\boldsymbol{K} = \{\boldsymbol{K}_1 \| \boldsymbol{K}_2\}$, where $\boldsymbol{K}_i$ is 64-bit sub-key and $\|$ is the concatenation operation. However, in the cipher design, AK is implemented as $\mathsf{AK}(\boldsymbol{X}, \boldsymbol{K}_1 \oplus \boldsymbol{K}_2)$, which makes adding constraints slightly different, *i.e.,* $\mathsf{PL}^{-1}(\boldsymbol{K})$ is replaced by $\mathsf{PL}^{-1}(\boldsymbol{K}_1 \oplus \boldsymbol{K}_2)$.

For SKINNY, how the number of analysis rounds N_r corresponds to N_c is shown in Figure 6, where the red curve can depict the change process of the fault leakages ϕ required by SKINNY-64-64. We use the minimum N_c that required for each N_r (the black node) and calculate the average value of ϕ, which is 117.3. Since the recovery of full key requires at least two rounds, the theoretical ϕ for SKINNY-64-64 is about 98 [ZZJ+20], which is close to 117.3.

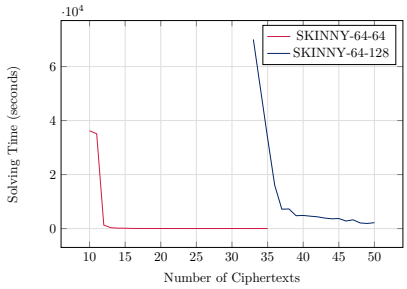

Figure 5: APFA with SKINNY-64-64 and SKINNY-64-128.

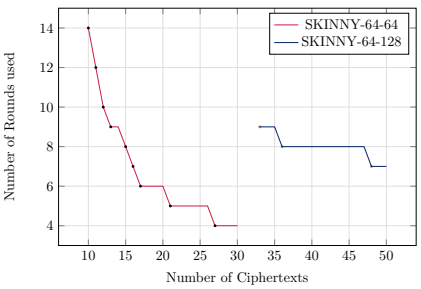

Figure 6: Relationship between N_c and N_r for SKINNY-64-64 and SKINNY-64-128.

7.6 Evaluation

Table 5 lists the evaluation results for the five lightweight block ciphers. APFA can effectively reduce the number of required ciphertexts. More specifically, N_c for PRESENT-80, LED-64 and SKINNY-64-64 are reduced by $101/18 \approx 5.61$ times, $75/23 \approx 3.26$ times and $1550/10 \approx 155$ times, respectively. Since the key sizes of PRESENT and SKINNY only affect their key schedule (See Row 1~2 and 4~5), the values of N_e and ψ are unchanged under different key sizes. Comparing the two versions of PRESENT (Row 1~2) or SKINNY (Row 4~5), it can be found that increasing the key size does not significantly increase the required N_c for APFA. In other words, APFA is not sensitive to the key size.

Table 5: Evaluation metrics for different cipher.

Row	Cipher	N_c			N_r	N_e	ψ
		PFA	EPFA	This paper			
1	PRESENT-80	101	-	18	5	1088	7.35%
2	PRESENT-128	-	-	28	4		
3	LED-64	-	75	23	4	5584	1.43%
4	SKINNY-64-64	-	1550	10	14	1152	6.94%
5	SKINNY-64-128	-	-	33	9		

There is an observation that LED-64 has a greater N_e and a lower ψ than other ciphers. Moreover, LED can only utilize the constraints of the last step, which means a smaller ψ will bring the less capability (for the constraints) to reduce the key search space.

8 APFA on Other Ciphers

In addition to lightweight SPN-based block ciphers, we also explore APFA on the cipher LBlock which is with a Feistel structure and AES which is a classic block cipher.

8.1 APFA on LBlock

LBlock [WZ11] is a Feistel-based lightweight block cipher, proposed by Wu *et al.* in CANS 2011. The block size of LBlock is 64 bits and the key size is 80 bits. It consists of a 32-round iterative structure, where the state of each round is expressed as $\boldsymbol{X}^i \| \boldsymbol{X}^{i-1}$ ($\boldsymbol{X}^i$ is a 32-bit block). And it can be represented as Eq.(30). The round Function F of LBlock consists of **KeyAddition** (AK), **Substitution** (SB) and **LinearPermutation** (PL).

$$\begin{aligned} \boldsymbol{X}^i &= \mathsf{F}(\boldsymbol{X}^{i-1}, \boldsymbol{K}^{i-1}) \oplus (\boldsymbol{X}^{i-1} <<< 8) \\ \mathsf{F} &= \mathsf{PL}(\mathsf{SB}(\mathsf{AK}(\boldsymbol{X}^i, \boldsymbol{K}^i))) \end{aligned} \tag{30}$$

where $2 \leq i \leq 33$ and $<<<$ is the left cyclic shift operation.

When a fault is injected into LBlock, due to the design of Feistel structure, the fault leakage of F is masked by the previous intermediate state $\boldsymbol{X}^{i-1}$ as shown in Eq.(30). In this case, the proposed APFA can still work when those ineffective ciphertexts can be identified and collected. The adversary needs to encrypt the same plaintext for twice, one for the normal encryption and the other for the encryption with faulty S-box S'. Thus he can collect those ciphertexts that have not accessed S' during the whole encryption.

There are 8 different parallel S-boxes $\mathsf{S}_0, \mathsf{S}_1, \cdots, \mathsf{S}_7$ in SB of LBlock. Assume a single fault has been injected into one S-box, the probability of those ciphertexts that do not access the faulty S-box is about $(1 - 1/16)^{32} \times 100\% \approx 12.68\%$. Such probability is still relatively high and cannot be ignored in practice. The APFA on the LBlock is slightly different from those on previous lightweight SPN-based block ciphers. Suppose the faulty S-box is S_i and the fault location is l. Since these ineffective ciphertexts do not visit $\mathsf{S}_i[l]$, we have $X_i \oplus K_i \neq l$ where X_i and K_i are an element of the intermediate block and round key, respectively. Therefore, a new constraint, *i.e.,* $K_i \neq X_i \oplus l$, can be applied to the round key. The difference lies in the constraint construction where the impossible value is the XORed result between the key and the fault location instead of the fault value itself.

We perform multiple fault injections for different S-boxes. For each fault injection, 14 ciphertexts are collected, and a total of 112 ciphertexts are obtained for eight S-boxes. For these ciphertexts we created the corresponding equations and added constraints on the round keys. The full 80-bit key can be recovered when the last nine rounds of fault leakage were utilized, *i.e.,* $N_r = 9$.

8.2 APFA on AES

Advanced Encryption Standard (AES) is a well-known block cipher released at NIST in 2001 [DR99]. For AES-128, $n = 128, w = 8, R = 10$. The round function F of AES includes

AddRoundKey (AK), **SubBytes** (SB), **ShiftRows** and **MixColumns** (PL), which can be represented as follows:

$$\mathsf{F} = \mathsf{AK}(\mathsf{PL}(\mathsf{SB}(\boldsymbol{X}^r)), \boldsymbol{K}^r),\ 1 \leq r \leq 10 \tag{31}$$

Since the round function for AES and LED is quite similar, the construction of F can be referred to Section 6.2. However due to the size w, the number of variables and equations that are generated is relatively high. We get $N_e = 22400$ and $\psi \approx 0.64\%$. AK and SB need 128 and 20592 equations, respectively. PL of AES also consists of two types of permutation functions, and it will generate 768 equations. When applying APFA on AES, there are two major difficulties. The first is the huge amount of equations. For the original PFA in [ZLZ+18], the theoretical number of ciphertexts required to recover the full key is about 1561. If we construct the equations for half of the ciphertexts (roughly 780) to the penultimate round, the script file size is already close to 1GB. The second is the large search space. When pursuing the least number of ciphertexts that are required, once the total amount of ciphertexts is not enough, the unknown search space of the last round key becomes larger (the size of the search space is 256). And the complexity of the system will grow exponentially. Therefore, it is very difficult to use the multi-round fault leakages of AES.

For the sake of completeness, we simply apply APFA to the final round of AES, in order to verify its feasibility on those classic and heavy block ciphers. We add a pair of plaintext and faulty ciphertext as a verification equation, and combine it with the entire equation system of AES. The result is shown in Table 1. And its $N_c = 1300$ and $N_r = 1$, which requires 54514 seconds. In general, as for AES, the improvement from APFA is not that much as compared to EPFA. Note that the acceleration of EPFA adopts a different hardware approach (using GPU). However it is still comparable to the original PFA. Since the S-box is an order of magnitude larger than the lightweight block cipher, this result can still be acceptable.

9 Conclusion

In this paper, we propose *Algebraic Persistent Fault Analysis* (APFA), which is a new type of analysis combining persistent fault analysis and algebraic fault analysis. It uses fewer ciphertexts as well as deeper rounds of fault information to recover the key. We manage to launch APFA on various SPN-based block ciphers. In addition to SPN-based, we also verify the feasibility of APFA on Feistel-based light weight block ciphers (*e.g.,* LBlock) and classic block ciphers (*e.g.,* AES).

APFA requires only one fault injection and can effectively reduce the number of faulty ciphertexts. It is quite effective for the cryptanalysis on lightweight block ciphers, especially when the round key participates in more rounds, and the permutation layer is relatively simple. In particular, as for APFA on SKINNY-64-64, it only requires about 10 ciphertexts to recover the full key, which is reduced by 155 times compared with EPFA.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2020AAA0107700), by National Natural Science Foundation of China (62072398, 62032021, 61772236), by Open Fund of State Key Laboratory of Cryptology (MMKFKT202013), by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by Major Scientific Research Project of Zhejiang Lab (2018FD0ZX01), by Zhejiang Key R&D Plan (2021C01116), by Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang (2018R01005), by Research Institute of Cyberspace Governance in Zhejiang University, by National Key

Laboratory of Science and Technology on Information System Security, by State Key Laboratory of Mathematical Engineering and Advanced Computing, and by Key Laboratory of Cyberspace Situation Awareness of Henan Province.

References

[ABF+02] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and J-P Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 260–275. Springer, 2002.

[BBBP13] Alessandro Barenghi, Guido M Bertoni, Luca Breveglieri, and Gerardo Pelosi. A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA. *Journal of Systems and Software*, 86(7):1864–1878, 2013.

[BDL97] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques*, pages 37–51. Springer, 1997.

[BFS04] Magali Bardet, Jean-Charles Faugere, and Bruno Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Annual International Cryptology Conference*, pages 123–153. Springer, 2016.

[BKL+07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *International workshop on cryptographic hardware and embedded systems*, pages 450–466. Springer, 2007.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[CB19] Andrea Caforio and Subhadeep Banik. A study of persistent fault analysis. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 13–33. Springer, 2019.

[CJW10] Nicolas T Courtois, Keith Jackson, and David Ware. Fault-algebraic attacks on inner rounds of DES. In *E-Smart'10 Proceedings: The Future of Digital Security Technologies*. Strategies Telecom and Multimedia, 2010.

[CMR06] Carlos Cid, Sean Murphy, and Matthew Robshaw. *Algebraic aspects of the advanced encryption standard*. Springer Science & Business Media, 2006.

[Cou04] Nicolas T Courtois. Algebraic attacks on combiners with memory and several outputs. In *International Conference on Information Security and Cryptology*, pages 3–20. Springer, 2004.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 547–572, 2018.

[DLV03] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on AES. In *International Conference on Applied Cryptography and Network Security*, pages 293–306. Springer, 2003.

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED block cipher. In *International workshop on cryptographic hardware and embedded systems*, pages 326–341. Springer, 2011.

[HS13] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 219–235. Springer, 2013.

[JT12] Marc Joye and Michael Tunstall. *Fault analysis in cryptography*, volume 147. Springer, 2012.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

[KHN+19] Mustafa Khairallah, Xiaolu Hou, Zakaria Najm, Jakub Breier, Shivam Bhasin, and Thomas Peyrin. SoK: On DFA Vulnerabilities of Substitution-Permutation Networks. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 403–414, 2019.

[KM10] Lars R Knudsen and Charlotte V Miolane. Counting equations in algebraic attacks on block ciphers. *International Journal of Information Security*, 9(2):127–135, 2010.

[Muk09] Debdeep Mukhopadhyay. An improved fault based attack of the advanced encryption standard. In *International Conference on Cryptology in Africa*, pages 421–434. Springer, 2009.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In *International workshop on cryptographic hardware and embedded systems*, pages 77–88. Springer, 2003.

[SA02] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *International workshop on cryptographic hardware and embedded systems*, pages 2–12. Springer, 2002.

[SKMD17] Sayandeep Saha, Ujjawal Kumar, Debdeep Mukhopadhyay, and Pallab Dasgupta. Differential Fault Analysis Automation. *IACR Cryptol. ePrint Arch.*, 2017:673, 2017.

[Sko10] Sergei Skorobogatov. Optical fault masking attacks. In *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 23–29. IEEE, 2010.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer, 2009.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices*, pages 224–233. Springer, 2011.

[WZ11] Wenling Wu and Lei Zhang. LBlock: a lightweight block cipher. In *International conference on applied cryptography and network security*, pages 327–344. Springer, 2011.

[XZY+20] Guorui Xu, Fan Zhang, Bolin Yang, Xinjie Zhao, Wei He, and Kui Ren. Pushing the limit of PFA: Enhanced Persistent Fault Analysis on Block Ciphers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[ZGZ+12] Xin-jie Zhao, Shize Guo, Fan Zhang, Tao Wang, Zhijie Shi, and Keke Ji. Algebraic Differential Fault Attacks on LED using a Single Fault Injection. *IACR Cryptol. ePrint Arch.*, 2012:347, 2012.

[ZLZ+18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 150–172, 2018.

[ZZJ+20] Fan Zhang, Yiran Zhang, Huilong Jiang, Xiang Zhu, Shivam Bhasin, Xinjie Zhao, Zhe Liu, Dawu Gu, and Kui Ren. Persistent fault attack in practice. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 172–195, 2020.