IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 2, pp. 519–542. DOI:10.46586/tches.v2023.i2.519-542

Efficient Persistent Fault Analysis with Small Number of Chosen Plaintexts

Fan Zhang[1,2], Run Huang[1,3], Tianxiang Feng[1,2], Xue Gong[1], Yulong Tao[5], Kui Ren[1], Xinjie Zhao[4,1] and Shize Guo[1]

[1] School of Cyber Science and Technology, College of Computer Science and Technology, Zhejiang University, Hangzhou, China, 310027
[2] Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, Hangzhou, China, 310027
[3] Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou, China, 310027
[4] Henan Province Key Laboratory of Cyberspace Situation Awareness, Zhengzhou, China, 450001
[5] Shanghai Institute of Satellite Engineering, Shanghai, China, 201109
fanzhang@zju.edu.cn;huangrun@zju.edu.cn

Abstract.
In 2018, Zhang *et al.* introduced the *Persistent Fault Analysis* (PFA) for the first time, which uses statistical features of ciphertexts caused by faulty Sbox to recover the key of block ciphers. However, for most of the variants of PFA, the prior knowledge of the fault (location and value) is required, where the corresponding analysis will get more difficult under the scenario of multiple faults. To bypass such perquisite and improve the analysis efficiency for multiple faults, we propose *Chosen-Plaintext based Persistent Fault Analysis* (CPPFA). CPPFA introduces chosen-plaintext to facilitate PFA and can reduce the key search space of AES-128 to extremely small. Our proposal requires 256 ciphertexts, while previous state-of-the-art work still requires 1509 and 1448 ciphertexts under 8 and 16 faults, respectively, at the only cost of requiring 256 chosen plaintexts. In particular, CPPFA can be applied to the multiple faults scenarios where all fault locations, values and quantity are unknown, and the worst time complexity of CPPFA is $O(2^{8+n_f})$ for AES-128, where n_f represents the number of faults. The experimental results show that when $n_f > 4$, 256 pairs of plaintext-ciphertext can recover the master key of AES-128. As for LED-64, only 16 pairs of plaintext-ciphertext reduce the remaining key search space to 2^{10}.

Keywords: Fault Attack · Persistent Fault Analysis · Multiple Faults · AES · LED

1 Introduction

The security of cryptographic algorithms has long been proven by cryptographers, and it is difficult to obtain the master key using brute force or simple algebraic analysis. However, most cryptographic algorithms rely on physical device platforms such as SIM cards, IC cards, etc. The method of recovering the key by injecting fault into the physical device is called *Fault Injection Attack* (FIA). FIA was first proposed and applied to RSA in 1997 by Boneh *et al.* [BDL97]. In the same year, Biham *et al.* proposed *Differential Fault Analysis* (DFA) on the block cipher DES [BS97]. DFA uses correct and faulty ciphertext pairs to recover the key. DFA is the most extensively used fault analysis technique, which works with almost all of the block ciphers. Besides, the fault analysis methods developed on

Received: 2022-10-15 Accepted: 2022-12-15 Published: 2023-03-06

the basis of FIA include *Ineffective Fault Analysis* (IFA) [Cla07], *Statistical Fault Analysis* (SFA) [FJLT13] and *Statistical Ineffective Fault Analysis* (SIFA) [DEK+18].

The types of fault injection models can be divided into three categories: transient faults, permanent faults, and persistent faults. The majority of the proposed fault attack techniques focus on transient faults, which only affect the system for a short time. Transient faults typically require the injection at a specific time and location, thus changing the intermediate values in the encryption, which affects only the current encryption of the device. Permanent faults result in irreversible faults of the equipment [Sko10]. Different from the first two, persistent faults are between transient faults and permanent faults, which hold for a certain duration and will not disappear until the device reboots. Schmidt *et al.* were the first to use the term persistent fault to describe an attack on AES that erased the non-volatile memory with UV light [SHP09]. Later, in CHES 2018, Zhang *et al.* extended persistent faults and proposed *Persistent Fault Analysis* (PFA) [ZLZ+18].

Transient faults require a high level of precision for attackers, which raises the cost and difficulty of FIA. PFA does not require a tightly-coupled injection, thus it has received a lot of attention in recent years. Persistent faults are often used to attack constants in cryptographic algorithms, and PFA applied this fault to the Sbox. But unfortunately, the fault value and fault location of the Sbox need to be known or obtained through statistics in Zhang's first proposal of PFA. To solve the problem, Zhang *et al.* introduced *Maximum Likelihood Estimation* (MLE) to estimate the fault value and then enumerate the fault location with the reduced number of ciphertexts (PFA-20) [ZZJ+20]. However, neither PFA nor PFA-20 can be extended to the multiple faults setting unless the exact location and value of the faults are known in advance by the attacker. In practice, the application of PFA can meet certain challenges since commonly utilized FIA techniques such as clock glitch and electromagnetic pulse usually create multiple faults per injection [TL21]. To cope with this problem, Engels *et al.* presented *Statistical Persistent Fault Analysis* (SPFA) by merging SFA with PFA [ESP20]. SPFA can bypass the premise that multiple fault values and locations are known, and can reduce the search space of key under multiple faults. However, SPFA is only able to reduce the remaining key search space to 2^{50} and it needs to try all candidate keys in order to get the unique key, so the time complexity is $O(2^{50})$. Soleimany *et al.* proposed a practical analysis method named *Practical Multiple Persistent Faults Analysis* (PMPFA) [SBH+21] to overcome the problems of multiple faults and the high computational complexity of SPFA. PMPFA utilizes multiple faults that have the same distribution in multiple bytes of ciphertext, and reduces the total key search space by enumerating one of the key bytes. PMPFA can effectively utilize multiple faults and reduce the number of required ciphertexts as much as possible. However, its way of reducing the key search space is different from that of the original PFA. In order to demonstrate the flexibility of SPFA and PMPFA, they had also performed their method to the lightweight block cipher LED [GPPR11], besides AES. LED is an AES-like lightweight block cipher proposed in 2011 with an SPN structure and an ultra-light key schedule. Therefore, it can be very fast and have a very efficient software or hardware implementations. To verify the generality of this paper and provide a fair comparison with previous work, LED is also one of the target ciphers of this paper.

In addition to those previous works in the multiple faults, there are other aspects of PFA research. In 2019, Caforio *et al.* extended PFA to block ciphers under the Feistel networks [CB19]. Xu *et al.* combined multiple rounds of fault leakages and used GPU acceleration to reduce the number of ciphertexts to less than 1000 [XZY+20]. Very recently, Zhang *et al.* proposed a new attack called *Algebraic Persistent Fault Analysis* (APFA) [ZFL+22], by making use of the *Algebraic Fault Analysis* (AFA) [CJW10] in PFA. APFA builds algebraic equations of multiple rounds of fault leakages for block ciphers, which are solved by an algebraic solver, thereby reducing the number of required ciphertexts. In addition, Zheng *et al.* combined PFA with collision attack, and proposed a method named as *Persistent Fault-Based Collision Analysis* (PFCA) [ZLZ+21]. PFCA obtains the plaintext collision set by chosen-plaintext [KOS10], which provides a new method for using PFA fault information. However, PFCA is not sufficient for the utilization of fault information and it requires more ciphertexts if the number of persistent faults is increased.

In short summary, with years of evolvement, PFA has developed a sufficient number of theories and applications. However, there still exist some problems for different variants of PFA, especially under scenario of multiple faults. In this paper, we propose a new analysis method called *Chosen-Plaintext based Persistent Fault Analysis* (CPPFA), which introduces the analysis of chosen-plaintext on the basis of PFA with multiple faults. CPPFA can reduce the number of samples required and the search space for the remaining keys very efficiently. The main contributions of this work can be summarized as follows:

- We propose *Chosen-Plaintext based Persistent Fault Analysis* (CPPFA) as a new analysis method capable of recovering the master key in a scenario with multiple unknown faults. The attacker does not need to know the fault locations, values and the number of faults in advance. Furthermore, in comparison to previous methods, CPPFA becomes more effective as the number of faults injected increases.
- We apply CPPFA to a variety of block ciphers, such as AES [Sta01] and LED [GPPR11]. When the number of faults is large enough (≥ 5), the attacker just has to construct 256 plaintexts, which is enough for reducing the key search space of AES down to one. For LED, CPPFA can reduce the space to about 2^{10} by 16 plaintexts.
- We abstract the fault model of CPPFA into several coupon collection problems (CCP) and give the corresponding expectations, including the expectation of the remaining key search space for a fixed number of samples, and the number of ciphertexts required to obtain a unique key for various numbers of faults.
- We discuss the issue of the key schedule that has been ignored by PFA-related works, as well as analyzing the application of CPPFA for having a protected implementation of block ciphers.

One of the most significant contributions of CPPFA is that it can recover the master key under the scenario of multiple unknown faults with only a few encryptions. Table 1 gives a summary of our results on AES-128 in comparison with related works, including the original PFA [ZLZ+18], the PFA with collision analysis [ZLZ+21], the PFA with statistical fault analysis [ESP20] and the practical PFA with multiple faults [SBH+21]. For a fair comparison under the same number of faults n_f, CPPFA requires the least amount of ciphertexts of all PFAs to recover the key, at the only cost of introducing chosen-plaintext.

Table 1: The Table of Comparison with Related Works on AES-128(†: It is assumed the faults are known in advance)

n_f	Chosen-Plaintexts/Ciphertexts					Remaining Key Candidates				
	PFA† [ZLZ+18]	PFCA [ZLZ+21]	SPFA [ESP20]	PMPFA [SBH+21]	CPPFA this work	PFA† [ZLZ+18]	PFCA [ZLZ+21]	SPFA [ESP20]	PMPFA [SBH+21]	CPPFA this work
2	-/2000	3330/3330	-/7775	-/1552	256/1041	2^{16}	1	2^{50}	2^{23}	1
4	-/2000	3667/3667	-	-/1010	256/524	2^{35}	1	-	$2^{8.71}$	1
8	-/2000	3871/3871	-/2008	-/1509(671)	256/256	2^{50}	1	2^{50}	$2^{8}(2^{14.56})$	1
16	-/2000	-	-/1643	-/1448(477)	256/256	2^{64}	-	2^{50}	$2^{8}(2^{24.52})$	1

The rest of this paper is organized as follows. Section 2 introduces the background. Section 3 talks about the fault model, motivation and core idea. Section 4 describes the general analysis model for CPPFA on block ciphers. Section 5 gives the calculation of two related expectations and analyzes the time complexity of CPPFA. Section 6 and Section 7 apply the proposed CPPFA to AES-128 and LED-64 at both theoretical and experimental levels, respectively. Section 8 concludes the paper.

2 Background

In this section, we will introduce the related background including *Persistent Fault Analysis* (PFA) and *Persistent Fault-Based Collision Analysis* (PFCA).

2.1 Persistent Fault Analysis

Persistent Fault Analysis (PFA) is the first practical persistent fault analysis framework proposed by Zhang *et al.* at CHES 2018. PFA will inject a single persistent fault into the Sbox of AES, causing an initial value v in the Sbox becomes v^*. v will no longer appear, and v^* will appear with doubled frequency. As a result, the probability distribution for Sbox output can be calculated as in the left part of Eq.(1). Then, the victim uses a fixed key to encrypt multiple plaintexts (unknown to the attacker) via the faulty AES. Finally, the attacker collects the faulty ciphertexts to recover the key.

$$Pr(x) = \begin{cases} 0 & \text{if } \ x = v \\ \frac{2}{256} & \text{if } x = v^* \\ \frac{1}{256} & \text{otherwise} \end{cases} \quad Pr(c_j) = \begin{cases} 0 & \text{if } \ c_j = v \oplus k_j^{R-1} \\ \frac{2}{256} & \text{if } c_j = v^* \oplus k_j^{R-1} \\ \frac{1}{256} & \text{otherwise} \end{cases} \tag{1}$$

Because of the unbalanced distribution in the Sbox output, the ciphertext of AES will have a similar distribution for the j-th byte c_j, which can be shown in the right part of Eq.(1). According to the distribution, three strategies are proposed for key recovery.

- **Strategy 1**: Utilizing c_{min}. Since the distribution of c_j is unbalanced, there is a value that never appears that is noted as c_{min}. It is accepted that $c_{min} = v \oplus k_j^{R-1}$. If v is known and there are enough ciphertexts, k_j^{R-1} can be recovered uniquely in the following way:
$$k_j^{R-1} = v \oplus c_{min} \tag{2}$$
- **Strategy 2**: Utilizing c_{max}. Similar to **Strategy 1**, a value in c_j has twice the frequency, which is called c_{max}. According to the distribution, $c_{max} = v^* \oplus k_j^{R-1}$. In other words, if v^* can be known, k_j^{R-1} can be inferred by Eq.(3).
$$k_j^{R-1} = v^* \oplus c_{max} \tag{3}$$
- **Strategy 3**: Utilizing impossible values $c_j \neq v \oplus k_j^{R-1}$. According to **Strategy 1**, c_{min} will never appear. This means that $c_j \neq c_{min}$, *i.e.*, $c_j \neq v \oplus k_j^{R-1}$. Therefore, using Eq.(4) for each c_j of ciphertexts, which will reduce the search space of k_j^{R-1} by one.
$$k_j^{R-1} \neq v \oplus c_j \tag{4}$$

  In practice, **Strategy 3** is the most commonly adopted.

Observing the above three strategies, they all have a similar premise: knowing the actual injected fault v^* or the initial value v. If n_f faults are injected in the Sbox, there will be n_f indistinguishable disappearance values and other n_f values with doubled frequency for the ciphertext c_j. In the case where the initial values of n_f faults $\{v^0, v^1, \cdots, v^{n_f-1}\}$ (called as the impossible value set) are known, strategy 3 still works. This means that a ciphertext can reduce n_f impossible values in the search space of k_j^{R-1} by Eq.(5) (called as **Strategy 3 with multiple faults**).

$$\begin{cases} k_j^{R-1} \neq v^0 \oplus c_j \\ \quad\quad \vdots \\ k_j^{R-1} \neq v^{n_f-1} \oplus c_j \end{cases} \tag{5}$$

2.2 Persistent Fault-Based Collision Analysis

Persistent Fault-Based Collision Analysis (PFCA) is proposed by Zheng *et al.* in 2021. The fault model of PFCA is somewhat different from that of PFA. It abandons the advantage of ciphertext-only analysis, and analyzes the key by plaintexts. PFCA introduces chosen-plaintext, and circumvents the premise that PFA requires knowledge of the fault. Suppose a single fault is injected into the Sbox of AES, and let $\text{Sbox}[l] = v$ becomes $\text{Sbox}[l] = v^*$. It can be found that there is another initial Sbox value $\text{Sbox}[l'] = v^*$. And then PFCA randomly generates the plaintexts. For these plaintexts, the first byte takes any of 256 possible values, and the other bytes are fixed. There will be two different plaintexts, whose first bytes are $p_0 = k_0 \oplus l$ and $p'_0 = k_0 \oplus l'$. In other word, ciphertexts encrypted by these two plaintexts will be equal. Similarly, for each byte, the chosen plaintexts for that byte can be constructed (The total number of plaintexts is $256 \times 16 = 4096$). In each byte, there will be two plaintexts whose corresponding ciphertexts are the same, and a relationship such as Eq.(6) can be obtained.

$$\begin{cases} p_0 \oplus k_0 = p_1 \oplus k_1 = \cdots = p_{15} \oplus k_{15} = l \\ p'_0 \oplus k_0 = p'_1 \oplus k_1 = \cdots = p'_{15} \oplus k_{15} = l' \end{cases} \tag{6}$$

Taking the first row in Eq.(6) as an example, it can be found that when the first byte k_0 of the key is determined, the entire key is determined.

$$\begin{cases} k_1 = p_1 \oplus p_0 \oplus k_0 \\ k_2 = p_2 \oplus p_0 \oplus k_0 \\ \quad \vdots \\ k_{15} = p_{15} \oplus p_0 \oplus k_0 \end{cases} \tag{7}$$

According to Eq.(7), k_0 traverses all possible values, and 256 candidates of the initial round key K can be obtained. Then, PFCA examines each key candidate with a given plaintext-ciphertext pair, *i.e.,* if the encryption result of the plaintext using a key candidate equals the ciphertext, the key candidate is what we are searching for. PFCA can reduce the master key search space to 2^8 with 4096 plaintexts. Unlike original PFA, it does not use ciphertexts to reduce the search space of the key, but just compares whether the same ciphertexts exist to filter out the corresponding plaintexts, and discards these ciphertexts after comparison. In addition, PFCA can also be extended to multi-fault cases.

2.3 Rear-Round Collision

There is another crucial issue that cannot be ignored when using the chosen-plaintext technique in persistent fault analysis. For two plaintexts, even if the intermediate states are not equal after the first round, the subsequent internal states and the corresponding ciphertexts may collide, which is called the rear-round collision. In addition to the collisions in the first round, rear-round collisions may also make the output identical ciphertexts. In this case, the chosen-plaintext technique is invalid. When the number of fault $n_f = 1$, it is assumed that two plaintexts have different intermediate states after the first round Sbox. After **Mixcolumns**, the intermediate states will be four bytes different. In the best case, two bytes of the first column collide after the second round Sbox (the probability is $(4/2^{16})^2$), and the remaining two affect 8 bytes of the states after **Mixcolumns**. Therefore, the probability of all bytes colliding after the third round is less than $(4/2^{16})^{2+8} = 7.17e - 43$ [ZLZ+21]. When $n_f > 1$ and all the faults are independent (n_f elements change to another n_f elements), in the best case of the rear-round collision, all four bytes of the two intermediate states will be the same after the second round Sbox. At this time, the probability of each pair of bytes in the two intermediate states colliding is $(2n_f/2^8) \times (2/2^8) = 4n_f/2^{16}$ ($2n_f/2^8$ is the probability of all elements related with faults, and $2/2^8$ is the probability of two elements with the same value), the probability of rear-round collision after the second round is less than $(4n_f/2^{16})^4$. Furthermore, PFCA [ZLZ+21] is speculated that rear-round collisions can be neglected when the number of faults is greater than 2, and this view is experimentally verified. The experimental results show no rear-round collision is detected when the number of faults is less than 48, which ensured that the chosen-plaintext technique is feasible in the vast majority of cases.

2.4 Notations

Table 2 defines the notations used in this paper.

Table 2: Notations used in this paper.

Notations	Definitions	Notations	Definitions
Parameters:		K	The master secret key
w	The element size	k_i	The i-th element of K
m	The number of elements	K^{R-1}	The last round key
θ	The θ-th element	k_i^{R-1}	The i-th element of K^{R-1}
R	The total number of rounds	**Analysis:**	
SBox	The real S-Box that is publicly known	n_f	The number of injected faults
Samples:		n_s	The number of sets
$\mathbb{P}$, $\mathbb{C}$	2^w chosen plaintexts and corresponding ciphertexts	λ_i	The size of i-th set $\mathcal{V}_i$
C^j	The j-th ciphertext	$\mathcal{V}_i$	The i-th set of initial value of the Sbox corresponding to v_i^*
$\mathbb{P}_i$	The i-th plaintext set ($= \{P_i^0, \cdots, P_i^{\lambda_i-1}\}$) with same ciphertexts	v_i^j	The j-th initial value of the i-th original set $\mathcal{V}_i$
P_i^j	The j-th plaintext of $\mathbb{P}_i$	$\mathcal{V}^*$	The set of the injected value of the Sbox
p_i^j	The θ-th element of the plaintext P_i^j	v_i^*	The i-th injected value of Sbox corresponding to $\mathcal{V}_i$
$\mathcal{P}_i$	The θ-th elements set of $\mathbb{P}_i$ ($= \{p_i^0, \cdots, p_i^{\lambda_i-1}\}$)	$\mathcal{D}$	The impossible value set of Sbox ($=\{v^0, \cdots, v^{n_f}\}$)
$P^{(j)}$	The j-th plaintext of $\mathbb{P}$	**Relation:**	
$p^{(j)}$	The θ-th element of $P^{(j)}$ ($=j$)	$A \to B$	B can be inferred from A

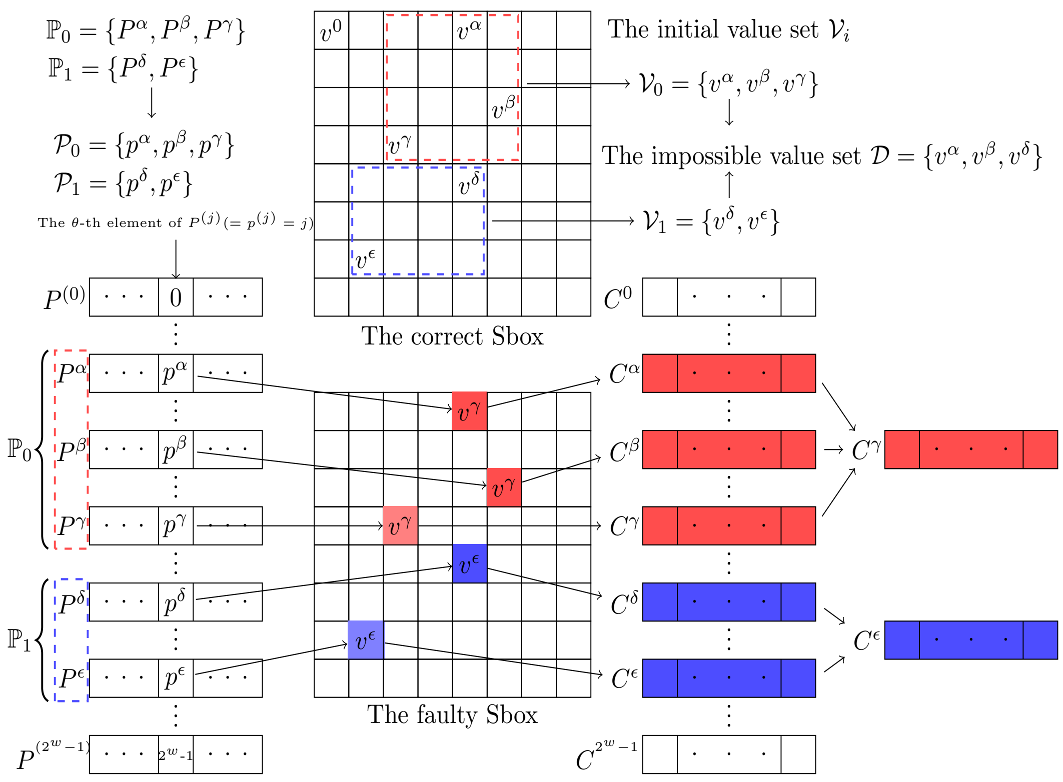
3 Overview

In this section, we will introduce the fault model, our motivation and core idea for the proposed *Chosen-Plaintext based Persistent Fault Analysis* (CPPFA).

3.1 Fault Model

Suppose n_f faults are injected into distinct Sbox elements, which are changed to other n_s elements in the Sbox that are not affected by the faults. Furthermore, the faults are persistent, *i.e.,* the injected faults will remain in the Sbox for a certain time until the Sbox is refreshed. As illustrated in Figure 1, three faults are injected after one injection. Those faults could also be injected through multiple injections as they are persistent and could be accumulated. For the first two faults, the initial values v^α, v^β in Sbox become v^γ (marked as red blocks in Figure 1) where α and β are the index of corresponding faulty elements, respectively. Note that there is already one element in Sbox whose initial value is exactly v^γ. Therefore there are three elements in total in the faulty Sbox whose values will be the same as v^γ. Similarly, for the third fault, the initial value v^δ becomes v^ϵ (marked as blue blocks in Figure 1) where there are actually two elements in the faulty Sbox whose value is exactly v^ϵ.

Therefore, the values v^α, v^β (marked in red) and v^δ (marked in blue) will not exist in the output of the faulty Sbox. These values can form a set of impossible values for faulty Sbox outputs and can be denoted as $\mathcal{D}$ (the impossible value set). Note that $\mathcal{D} = \{v^\alpha, v^\beta, v^\delta\}$. We assume the attacker has the capability to control the plaintexts and obtain the corresponding ciphertexts. Specifically, the attacker can use random plaintexts, or construct some chosen plaintexts. For chosen plaintexts, the attacker will construct 2^w special ones for encryption, denoted as $\{P^{(0)}, P^{(1)}, \cdots, P^{(2^w-1)}\}$, where each element at an arbitrary index θ in the plaintexts has 2^w different values which ranges from 0 to $2^w - 1$. Without loss of generality, we assume the θ-th element value of $P^{(j)}$ is j and values of all



$\mathbb{P}_0 = \{P^\alpha, P^\beta, P^\gamma\}$
$\mathbb{P}_1 = \{P^\delta, P^\epsilon\}$
$\mathcal{P}_0 = \{p^\alpha, p^\beta, p^\gamma\}$
$\mathcal{P}_1 = \{p^\delta, p^\epsilon\}$
The θ-th element of $P^{(j)} (= p^{(j)} = j)$
$P^{(0)}$
$P^{(2^w-1)}$
The correct Sbox
The faulty Sbox
The initial value set $\mathcal{V}_i$
$\mathcal{V}_0 = \{v^\alpha, v^\beta, v^\gamma\}$
The impossible value set $\mathcal{D} = \{v^\alpha, v^\beta, v^\delta\}$
$\mathcal{V}_1 = \{v^\delta, v^\epsilon\}$
C^0
C^α
C^β
C^γ
C^δ
C^ϵ
C^{2^w-1}

Figure 1: The overview of Fault Model and Core Idea

other elements are the same for different plaintexts. Then, the attacker can collect the corresponding ciphertexts for subsequent analysis.

3.2 Motivation

In fact, there exist several problems in the previous work, which actually motivates us to propose more efficient analysis method under the scenario of general PFA. These problems can be summarized as followings.

1). For those analysis methods which inherited from the original PFA [ZLZ+18], such as EPFA [XZY+20] and APFA [ZFL+22], they require either a strong attacker who should hold the prior knowledge of the faults, or a complicated process to recover the fault locations and values first before the fault analysis. We are motivated to pursue a solution to relax such constrain and recover the secret key directly. In this case, it is not required to know the fault locations and values. And more surprisingly, such information could be gained as by-products during the key recovery.

2). For those analysis methods which do not require the fault locations and values to be known as prerequisite, such as SPFA [ESP20] and PFCA [ZLZ+21], they meet certain difficulties in fully exploiting the fault leakages and still require a larger number of encryptions (overall complexity). Specially, SPFA is only able to reduce the remaining key search space to 2^{50}. To obtain the unique key requires one cryptographic verification for each candidate key, and the time complexity can be considered as $O(2^{50})$, which is a considerable runtime. Although PFCA can achieve the best time complexity $O(2^{12})$ in most cases, the worst time complexity is $O(2^{12} + 2^{128})$, which is no better than exhaustive search. We are motivated to propose new methods which fully utilize the leakages from multiple persistent faults and require extremely low number of plaintexts-ciphertexts. In addition, the time complexity of the new methods should be more stable while satisfying feasibility.

3). For those analysis methods which rely on the statistics of the ciphertexts, such as PMPFA [SBH+21], it is difficult for them to control the whole analysis process, and the possibly-correct candidates among the remaining key search space are not unique. Accordingly, it is mandatory for them to ask for a pair of correct plaintext-ciphertext for the purpose of additional verification. We are motivated to reduce the research space directly down to one, therefore no verification pair of encryptions is required.

In general, there is a lack of an analysis method that can bypass the prior knowledge of the faults, only require a small number of data complexity, and could be stable enough to reduce the key search space to the unique one. Naturally, we can think that PFA is not only applicable to ciphertext-only scenarios. With such motivation, it is possible for us to add some plaintexts information as auxiliaries which can help us solve the above problems.

3.3 Core Idea

In Sec.2.1, the application of PFA under multiple faults is mentioned and it is assumed that the attacker knows the fault locations and values in advance. In Sec.2.2, PFCA obtained the relationship between different key bytes through chosen plaintexts. However, their method did not pay enough attention to these plaintexts where the information about the desired fault locations and values is actually buried inside and could be further explored.

In a nutshell of our core idea, our method exploits such hidden leakages through chosen plaintexts and utilizes the guess of the first round key byte to generate some candidates of the impossible value set $\mathcal{D}$, denoted as $\hat{\mathcal{D}}$. For each $\hat{\mathcal{D}}$, a remaining key search space of last round can be generated from ciphertexts by **Strategy 3 with multiple faults** (see Eq.(5) in Sec.2.1).

More specifically, we first construct 2^w chosen plaintexts on the θ-th ($0 \leq \theta < m$) element, and encrypt them to obtain the corresponding 2^w ciphertexts under the fault model shown in Figure 1 (three faults were injected into the Sbox). There are three plaintexts P^α, P^β and P^γ (surrounded by red dotted lines on the left side of Figure 1) whose θ-th element $(p^\alpha, p^\beta, p^\gamma)$ satisfy Eq.(8):

$$\begin{aligned} &\text{Sbox}[p^\alpha \oplus k_\theta] = v^\alpha \rightarrow v^\gamma && //\text{faulty Sbox element} \\ &\text{Sbox}[p^\beta \oplus k_\theta] = v^\beta \rightarrow v^\gamma && //\text{faulty Sbox element} \\ &\text{Sbox}[p^\gamma \oplus k_\theta] = v^\gamma \rightarrow v^\gamma && //\text{correct Sbox element} \end{aligned} \tag{8}$$

The left side of "$\rightarrow$" represents the initial outputs of the Sbox v^α, v^β and v^γ (v^γ is not affected by the fault injection and remains the same), and the right side represents the output v^γ after fault injection. Therefore, v^α, v^β and v^γ form a set $\mathcal{V}_0 = \{v^\alpha, v^\beta, v^\gamma\} \rightarrow v^\gamma$, which is a set of initial values of Sbox output with the same output after fault injection and named as the initial value set. Similarly, we can have $\mathcal{V}_1 = \{v^\delta, v^\epsilon\} \rightarrow v^\epsilon$.

Here, we can obtain **Feature 1**: Only one member of the set $\mathcal{V}_i$ (*i.e.,* $\mathcal{V}_0 : v^\gamma$; $\mathcal{V}_1 : v^\epsilon$) will appear in the output of the faulty Sbox, and the rest members will exist in the impossible value set $\mathcal{D}$ (*i.e.,* $v^\alpha, v^\beta, v^\delta$). So the impossible value set $\mathcal{D}$ can be inferred from the initial value set $\mathcal{V}_i$.

On the one hand, P^α, P^β and P^γ are almost the same. The only slight difference lies in the θ-th element. On the other hand, due to the faulty Sbox, the θ-th element of the Sbox output in the first round will be equal. This means that the ultimate ciphertexts corresponding to the three plaintexts $(P^\alpha, P^\beta, P^\gamma)$ are identical after the entire encryption, *i.e.,* $C^\alpha = C^\beta = C^\gamma$ (the red ciphertexts on the right side of Figure 1).

The three identical ciphertexts $(C^{\alpha}, C^{\beta}, C^{\gamma})$ can be found directly in 2^w ciphertexts, so that their corresponding plaintexts $(P^{\alpha}, P^{\beta}, P^{\gamma})$ can be obtained. Next the θ-th element $(p^{\alpha}, p^{\beta}, p^{\gamma})$ of these three plaintexts will form the set $\mathcal{P}_0 = \{p^{\alpha}, p^{\beta}, p^{\gamma}\}$, which is a set of the θ-th element of plaintexts with the same ciphertexts. Similarly, for other identical ciphertexts illustrated in Figure 1, a corresponding yet different set $\mathcal{P}_1 = \{p^{\delta}, p^{\epsilon}\}$ can be obtained. Moreover, the corresponding ciphertext for $\mathcal{P}_0$ and $\mathcal{P}_1$ will be different.

If the value of k_θ is known, $\mathcal{V}_0$ and $\mathcal{V}_1$ can be easily determined by Eq.(8). Since the search space of k_θ is 2^w, we can try every possible value $\hat{k}_\theta$ (*i.e.,* the guess for k_θ, $0 \leq \hat{k}_\theta < 2^w$) without the knowledge of k_θ. For each $\hat{k}_\theta$, two corresponding guess sets $\hat{\mathcal{V}}_0$ and $\hat{\mathcal{V}}_1$ can also be calculated by Eq.(9). $\hat{\mathcal{V}}_0$ and $\hat{\mathcal{V}}_1$ are considered as the deducing of the output values of Sbox through the chosen plaintexts, and the faults remain unchanged during encryption due to persistent fault, thus they can be verified through the analysis on the last round using the final ciphertexts. Some candidate values of $\hat{k}_\theta$ will be eliminated and some will be kept. The overall key search space will be reduced gradually.

$$\begin{gathered}
\hat{\mathcal{V}}_0 = \left\{\hat{v}^{\alpha}, \hat{v}^{\beta}, \hat{v}^{\gamma}\right\} = \mathrm{Sbox}[\mathcal{P}_0 \oplus \hat{k}_\theta] \\
\hat{\mathcal{V}}_1 = \left\{\hat{v}^{\delta}, \hat{v}^{\epsilon}\right\} = \mathrm{Sbox}[\mathcal{P}_1 \oplus \hat{k}_\theta] \\
\hat{v}^{\alpha} = \mathrm{Sbox}[p^{\alpha} \oplus \hat{k}_\theta], \quad \hat{v}^{\beta} = \mathrm{Sbox}[p^{\beta} \oplus \hat{k}_\theta], \quad \hat{v}^{\gamma} = \mathrm{Sbox}[p^{\gamma} \oplus \hat{k}_\theta] \\
\hat{v}^{\delta} = \mathrm{Sbox}[p^{\delta} \oplus \hat{k}_\theta], \quad \hat{v}^{\epsilon} = \mathrm{Sbox}[p^{\epsilon} \oplus \hat{k}_\theta]
\end{gathered} \tag{9}$$

In this example, when $\hat{\mathcal{V}}_0 = \{\hat{v}^{\alpha}, \hat{v}^{\beta}, \hat{v}^{\gamma}\}$ and $\hat{\mathcal{V}}_1 = \{\hat{v}^{\delta}, \hat{v}^{\epsilon}\}$ are determined, only one member of $\hat{\mathcal{V}}_i$ will appear in the output of the faulty Sbox due to **Feature 1**, so six different possible candidates $\hat{\mathcal{D}}$ can be obtained, as shown in Eq.(10).

$$\begin{array}{lll}
\hat{\mathcal{D}}_0 = \{\hat{v}^{\beta}, \hat{v}^{\gamma}, \hat{v}^{\delta}\} & \hat{\mathcal{D}}_1 = \{\hat{v}^{\alpha}, \hat{v}^{\gamma}, \hat{v}^{\delta}\} & \hat{\mathcal{D}}_2 = \{\hat{v}^{\alpha}, \hat{v}^{\beta}, \hat{v}^{\delta}\} \\
\hat{\mathcal{D}}_3 = \{\hat{v}^{\beta}, \hat{v}^{\gamma}, \hat{v}^{\epsilon}\} & \hat{\mathcal{D}}_4 = \{\hat{v}^{\alpha}, \hat{v}^{\gamma}, \hat{v}^{\epsilon}\} & \hat{\mathcal{D}}_5 = \{\hat{v}^{\alpha}, \hat{v}^{\beta}, \hat{v}^{\epsilon}\}
\end{array} \tag{10}$$

For each possible candidate $\hat{\mathcal{D}}$, a corresponding remaining key search space of last round can be conducted by the original PFA under multiple faults (see Eq.(5) in Sec.2.1). When all possible candidate $\hat{\mathcal{D}}$ have been scrutinized, the keys that satisfy the guess value $\hat{k}_\theta$ in this six remaining spaces are collected as candidate keys, and the rest are excluded. After all $\hat{k}_\theta$ have been enumerated, the total key search space can be obtained.

Compared with the original PFA under single fault, which can only exclude one impossible value of each element using a ciphertext, in the case of multiple faults, each faulty ciphertext can exclude multiple (in this case is three) impossible values for each element. Therefore, the key search space can be reduced to a relatively small size or even one by merely using 2^w faulty ciphertexts corresponding to the 2^w chosen plaintexts.

4 Chosen-Plaintext based Persistent Fault Analysis

Suppose n_f faults are injected into the Sbox, resulting in n_s sets $\{\mathcal{V}_0, \mathcal{V}_1, \cdots, \mathcal{V}_{n_s-1}\}$ $(n_s \leq n_f)$. $\mathcal{V}_i$ is the set of initial values which are associated with the same output of Sbox after fault injection. The size of each set $\mathcal{V}_i$ is denoted as λ_i $(\lambda_i \geq 2)$ where $\mathcal{V}_i = \{v_i^0, v_i^1, \cdots, v_i^{\lambda_i - 1}\}$. For the original PFA, it is just a special case where $n_f = 1$, $n_s = 1$ and $\lambda_0 = 2$ as there is only one injected fault. But in practice, there are possibly multiple faults, which might cause multiple sets of $\mathcal{V}_i$. As shown in Eq.(11), v_i^j is the initial value of the j-th member in the set $\mathcal{V}_i$. After the injection, it becomes v_i^* (Note that there is one member in $\mathcal{V}_i$ whose initial value is exactly v_i^*. Such v_i^* is not caused by the injected fault).

$$\begin{aligned}
\mathcal{V}_0 &= \{v_0^0, v_0^1 \cdots, v_0^{\lambda_0-1}\} \rightarrow v_0^* \\
\mathcal{V}_1 &= \{v_1^0, v_1^1 \cdots, v_1^{\lambda_1-1}\} \rightarrow v_1^* \\
&\vdots \\
\mathcal{V}_{n_s-1} &= \{v_{n_s-1}^0, v_{n_s-1}^1 \cdots, v_{n_s-1}^{\lambda_{n_s-1}}\} \rightarrow v_{n_s-1}^*
\end{aligned} \tag{11}$$

Based on the explanation of Sec.3.3, after encrypting 2^w chosen plaintexts, n_s ciphertext sets will appear in the 2^w ciphertexts, and the ciphertexts in each ciphertext set are equal. For each ciphertext set (denoted as $\mathbb{C}_i$ as shown in the left part of Figure 2), we can find the corresponding plaintext set (denoted as $\mathbb{P}_i$ as shown in the center of Figure 2). For each plaintext set $\mathbb{P}_i$, there will be λ_i plaintexts whose θ-th element are different while other elements are exactly same. These different elements can form a set named $\mathcal{P}_i$ as shown in the right part of Figure 2. The subsequent analysis is based on the set $\mathcal{P}_i$. Algorithm 1 describes the pseudo code for finding n_s sets $\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_s-1}\}$ from $\mathbb{P}$ and $\mathbb{C}$. The attacker uses a tag array $\boldsymbol{B}$ to record the set index corresponding to the plaintexts. Algorithm 1 traverses all ciphertexts and puts the θ-th element of plaintexts with the same ciphertexts into the set $\mathcal{P}_i$.

Algorithm 1: get $\mathcal{P}_i$ from $\mathbb{P}$ and $\mathbb{C}$.

```
input  : $\mathbb{C}$, $\mathbb{P}$
output : $\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_s-1}\}$
1  $n_s = 0$;
2  $\boldsymbol{B} = \{-1, -1, \cdots, -1\}$ ;                      // the size of $\boldsymbol{B}$ is $2^w$.
3  for $i = 0; i < 2^w; ++i$ do
4      for $j = i - 1; j \geq 0; --j$ do
5          if $C^i == C^j$ and $\boldsymbol{B}_j == -1$ then
6              $\boldsymbol{B}_i = \boldsymbol{B}_j = n_s$;
7              $\mathcal{P}_{n_s}$.append($p^{(i)}$) ;             // $p^{(i)}$ is the $\theta$-th element of $P^{(i)}$.
8              $\mathcal{P}_{n_s}$.append($p^{(j)}$) ;    // $P^{(i)}$ is the plaintext corresponding to $C^i$.
9              $n_s$++;
10             break;
11         else if $C^i == C^j$ and $\boldsymbol{B}_j \neq -1$ then
12             $\boldsymbol{B}_i = \boldsymbol{B}_j$;
13             $\mathcal{P}_{\boldsymbol{B}_j}$.append($p^{(i)}$);
14             break;
15         end
16     end
17 end
18 return $\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_s-1}\}$;
```

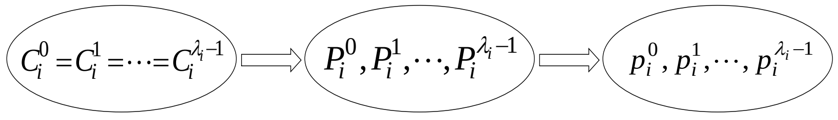

$C_i^0 = C_i^1 = \cdots = C_i^{\lambda_i-1}$
$P_i^0, P_i^1, \cdots, P_i^{\lambda_i-1}$
$p_i^0, p_i^1, \cdots, p_i^{\lambda_i-1}$

Figure 2: The relationship of plaintexts.

After having $\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_s-1}\}$, the next step is to try all possible candidate $\hat{k}_\theta$ ($0 \leq \hat{k}_\theta < 2^w$), assuming k_θ is unknown. For each candidate $\hat{k}_\theta$, a corresponding $\hat{\mathcal{V}}_i$ can be generated by Eq.(12), where $\hat{\mathcal{V}}_i$ is the guess value for $\mathcal{V}_i$. Since there are n_s number of $\mathcal{V}_i$, there will be n_s number of $\hat{\mathcal{V}}_i$ accordingly.

$$\begin{aligned}\hat{\mathcal{V}}_i &= \text{Sbox}[\mathcal{P}_i \oplus \hat{k}_\theta] \\ &= \{\text{Sbox}[p_i^0 \oplus \hat{k}_\theta], \text{Sbox}[p_i^1 \oplus \hat{k}_\theta], \cdots, \text{Sbox}[p_i^{\lambda_i - 1} \oplus \hat{k}_\theta]\} \\ &= \{\hat{v}_i^0, \hat{v}_i^1 \cdots, \hat{v}_i^{\lambda_i-1}\}\end{aligned} \tag{12}$$

Algorithm 2: get $\mathbb{D}$ from $\{\hat{\mathcal{V}}_0, \hat{\mathcal{V}}_1, \cdots, \hat{\mathcal{V}}_{n_s-1}\}$.

```
input  : $\{\hat{\mathcal{V}}_0, \hat{\mathcal{V}}_1, \cdots, \hat{\mathcal{V}}_{n_s-1}\}$
output : $\mathbb{D}$
1  $\hat{\mathbb{D}} = \emptyset$;
2  for $i_0 = 0$; $i_0 < \lambda_0$; $++i_0$ do
3      for $i_1 = 0$; $i_1 < \lambda_1$; $++i_1$ do
4          ⋮
5          for $i_{n_s-1} = 0$; $i_{n_s-1} < \lambda_{n_s-1}$; $++i_{n_s-1}$ do
6              $\hat{\mathcal{D}} = \{\}$;
7              for $j_0 = 0$; $j_0 < \lambda_0$; $++j_0$ do
8                  if $j_0 \neq i_0$ then
9                      $\hat{\mathcal{D}}$.append($\hat{v}_0^{j_0}$);    // Pick the impossible Sbox output value from $\hat{\mathcal{V}}_0$.
10                 end
11             end
12             ⋮
13             for $j_{n_s-1} = 0$; $j_{n_s-1} < \lambda_{n_s-1}$; $++j_{n_s-1}$ do
14                 if $j_{n_s-1} \neq i_{n_s-1}$ then
15                     $\hat{\mathcal{D}}$.append($\hat{v}_{n_s-1}^{j_{n_s-1}}$);    // The size of each $\hat{\mathcal{D}}$ is $\sum_{i=0}^{n_s-1}(\lambda_i - 1)$
16                 end
17             end
18             $\mathbb{D}$.append($\hat{\mathcal{D}}$) ;    // $\hat{\mathcal{D}} = \{\hat{v}^0, \hat{v}^1, \cdots\}$
19         end
20         ⋮
21     end
22 end
23 return $\mathbb{D}$;
```

Because of **Feature 1**, for each $\hat{\mathcal{V}}_i$, one member in $\hat{\mathcal{V}}_i$ is chosen as the Sbox element which is not affected by the fault, and the remaining members are put into the set $\hat{\mathcal{D}}$ (guess for the impossible values set $\mathcal{D}$). As shown in Algorithm 2, after completing the same operation for all sets $\hat{\mathcal{V}}_i$, a candidate $\hat{\mathcal{D}}$ can be obtained. Algorithm 2 describes the pseudo code for filtering $\hat{\mathcal{D}}$ from $\{\hat{\mathcal{V}}_0, \hat{\mathcal{V}}_1, \cdots, \hat{\mathcal{V}}_{n_s-1}\}$. The algorithm selects $\lambda_i - 1$ members from each set $\hat{\mathcal{V}}_i$ and adds them to $\hat{\mathcal{D}}$, and $\hat{\mathcal{D}}$ is stored in $\mathbb{D}$ for output. Each candidate $\hat{k}_\theta$ can generate $\prod_{i=0}^{n_s-1} \lambda_i$ different candidates $\hat{\mathcal{D}}$. The i-th candidate set $\hat{\mathcal{D}}$ obtained base on $\hat{k}_\theta$ is labeled as $\hat{\mathcal{D}}_i$.

Based on the guess value $\hat{k}_\theta$, for each $\hat{\mathcal{D}}_i$, a remaining key search space of last round (denoted as $\mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}_i}$, which is the search space associated with $\hat{k}_\theta$ and $\hat{\mathcal{D}}_i$) can be determined by **Strategy 3 with multiple faults** (see Eq.(5) in Sec.2.1). After obtaining $\prod_{i=0}^{n_s-1} \lambda_i$ remaining key search spaces $\mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}_i}$ $(0 \leq i < \prod_{j=0}^{n_s-1} \lambda_j)$ corresponding to $\hat{k}_\theta$, the union of all these spaces $(\mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}_0} \cup \mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}_1} \cup \cdots)$ is denoted as $\mathbb{K}^{R-1}_{\hat{k}_\theta}$. When we enumerate all $\hat{k}_\theta$, the total key search space of last round (denoted as $\mathbb{K}^{R-1}$) is the union of all $\mathbb{K}^{R-1}_{\hat{k}_\theta}$.

Algorithm 3: CPPFA on block ciphers using multiple faults on Sbox.

```
input  : $\mathbb{C}, \mathbb{P}$
output : $\mathbb{K}$
$\mathbb{K} = \emptyset$ ;
$\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_s-1}\}$ = Algorithm 1($\mathbb{P}, \mathbb{C}$);                       // Step 1.
for $\hat{k}_\theta = 0$; $\hat{k}_\theta < 2^w$; $++\hat{k}_\theta$ do
    $\mathbb{K}^{R-1}_{\hat{k}_\theta} = \emptyset$;                                                  // Step 2.
    for $i = 0$; $i < n_s$; $++i$ do
        $\hat{\mathcal{V}}_i = \text{Sbox}[\hat{k}_\theta \oplus \mathcal{P}_i]$;
    end
    $\mathbb{D}$ = Algorithm 2($\{\hat{\mathcal{V}}_0, \hat{\mathcal{V}}_1, \cdots, \hat{\mathcal{V}}_{n_s-1}\}$) ;
    for $\hat{\mathcal{D}} \in \mathbb{D}$ do
        $\mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}} = \{\{0, 1, \cdots, 2^w - 1\}_0, \cdots \{0, 1, \cdots, 2^w - 1\}_{m-1}\}$ ;
        for $C \in \mathbb{C}$ do
            for $j = 0$; $j < m$; $++j$ do
                for $\hat{v} \in \hat{\mathcal{D}}$ do
                    $\mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}}[j]$.remove($c_j \oplus \hat{v}$);     // $\hat{k}^{R-1}_j \neq c_j \oplus \hat{v}$   // Step 3.
                end
            end
        end
        $\mathbb{K}^{R-1}_{\hat{k}_\theta} = \mathbb{K}^{R-1}_{\hat{k}_\theta} \cup \mathbb{K}^{R-1}_{\hat{k}_\theta, \hat{\mathcal{D}}}$;
    end
    for $\hat{K}^{R-1} \in \mathbb{K}^{R-1}_{\hat{k}_\theta}$ do
        $\hat{K}$ = InverseKeySchedule ($\hat{K}^{R-1}$);
        if $\theta$-th element of $\hat{K}$ == $\hat{k}_\theta$ then
            $\mathbb{K}$.append($\hat{K}$);                                          // Step 4.
        end
    end
end
return $\mathbb{K}$ ;
```

In addition, for the last round key candidate $\hat{K}^{R-1}$ in $\mathbb{K}^{R-1}_{\hat{k}_\theta}$, the master key candidate $\hat{K}$ can be obtained by inverse key schedule algorithm. Since this candidate $\hat{K}$ is obtained on the basis of $\hat{k}_\theta$, if the θ-th element of $\hat{K}$ is not equal to $\hat{k}_\theta$, this candidate can be excluded to further narrow the remaining key search space.

The complete steps of CPPFA are shown in Algorithm 3. Algorithm 3 describes the pseudo code of the whole process of CPPFA. Step 1, CPPFA obtains the sets $\mathcal{P}_i$ by Algorithm 1 (in Line 2). Step 2, enumerating $\hat{k}_\theta$ from $0 \sim 2^w - 1$, obtaining the impossible values set $\hat{\mathcal{D}}$ for each $\hat{k}_\theta$ by Algorithm 2 (in Line $3 \sim 8$). Step 3, filtering the last round key with $\hat{\mathcal{D}}$ (in Line $9 \sim 18$). Step 4, using the inverse key schedule algorithm, further reduce the last round key search space $\mathbb{K}^{R-1}_{\hat{k}_\theta}$ (in Line $19 \sim 24$). Finally, the search space for the master key is returned.

5 Theoretical Analysis

The n_f faults are injected into n_f distinct Sbox elements and which are changed to other n_s elements of Sbox that are not affected by the faults. For $\{\mathcal{V}_0, \cdots, \mathcal{V}_{n_s-1}\}$ (the initial value sets) in this case, there is $\sum_{i=0}^{n_s-1} \lambda_i = (n_f + n_s)$ (λ_i is the size of $\mathcal{V}_i$) and the size of $\mathcal{D}$ (the impossible value set) is n_f.

5.1 Analysis of the Size of $\mathcal{D}$

When $n_f = 2$ ($\mathcal{D} = \{v^0, v^1\}$), $v^0 \oplus k_j^{R-1}$ and $v^1 \oplus k_j^{R-1}$ must not exist in the j-th element of the ciphertext. When the key search space of j-th element is reduced by $k_j \neq v^i \oplus c_j$, it is inevitable that there is another key candidate $k_j'^{R-1} = v^0 \oplus v^1 \oplus k_j^{R-1}$ will be left. Therefore, the key search space of j-th element will eventually drop to 2. The reason for this phenomenon is that the j-th element of ciphertext c_j is not equal to $v^0 \oplus k_j^{R-1}$ and $v^1 \oplus k_j^{R-1}$. However, suppose the j-th element of last round key is equal to $v^0 \oplus v^1 \oplus k_j^{R-1}$, the c_j will also have a distribution under k_j^{R-1}, *i.e.*, c_j is not equal to $v^0 \oplus k_j^{R-1}$ and $v^0 \oplus k_j^{R-1}$ (see Eq.(13)).

$$\begin{aligned} k_j'^{R-1} &= v^0 \oplus v^1 \oplus k_j^{R-1} \\ k_j'^{R-1} \oplus v^0 &= v^0 \oplus v^1 \oplus k_j^{R-1} \oplus v^0 = v^1 \oplus k_j^{R-1} \\ k_j'^{R-1} \oplus v^1 &= v^0 \oplus v^1 \oplus k_j^{R-1} \oplus v^1 = v^0 \oplus k_j^{R-1} \end{aligned} \tag{13}$$

For the case $n_f = 3$ ($\mathcal{D} = \{v^0, v^1, v^2\}$), suppose there is another element $k_j'^{R-1}$ ($\neq k_j^{R-1}$), both of which and k_j^{R-1} are left in the remaining key search space of j-th element. If $k_j'^{R-1}$ exists, $k_j'^{R-1} \oplus \{v^0, v^1, v^2\}$ needs to be equal to $k_j^{R-1} \oplus \{v^0, v^1, v^2\}$. However, if there is $k_j'^{R-1} \oplus v^0 = v^1 \oplus k_j^{R-1}$, then $k_j'^{R-1} \oplus v^1 = v^0 \oplus k_j^{R-1}$. For the v^2, $k_j'^{R-1} \oplus v^2$ must equal to $v^2 \oplus k_j^{R-1}$, which contradicts $k_j'^{R-1} \neq k_j^{R-1}$. In other words, for the case $n_f = 3$, the remaining key search space of each element can be reduced to the unique one. In addition, the conclusion is the same when n_f is some other odd number.

For the case $n_f = 4$ ($\mathcal{D} = \{v^0, v^1, v^2, v^3\}$), it is similar to $n_f = 2$ or $n_f = 3$. If $k_j'^{R-1}$ exists, $k_j'^{R-1}$ needs to satisfy the relation in Eq.(14). As long as there is such a relationship between these four different values, $k_j'^{R-1}$ cannot be removed by **Strategy 3 with multiple faults** of PFA (see Sec.2.1). For AES, if $n_f = 4$, the probability of Eq.(14) is 1/253 ($\approx 0.39\%$) of all possible fault combinations, which is also consistent with our experimental data.

$$\begin{array}{ccc} k_j'^{R-1} \oplus v^0 = v^1 \oplus k_j^{R-1} & & k_j'^{R-1} \oplus v^2 = v^3 \oplus k_j^{R-1} \\ k_j'^{R-1} \oplus v^1 = v^0 \oplus k_j^{R-1} & & k_j'^{R-1} \oplus v^3 = v^2 \oplus k_j^{R-1} \\ & \downarrow & \\ v^0 \oplus v^1 & = & v^2 \oplus v^3 \end{array} \tag{14}$$

The problem can also be abstracted as the probability of drawing 4 different balls from 256 balls numbered from 0 to 255 whose numbers satisfy Eq.(14). The total number of combinations is $256 \times 255 \times 254 \times 253$. There are $256, 255, 254$ possibilities for the first three balls, respectively. After the first three balls are determined, the number of the fourth ball satisfying Eq.(14) is unique, so the probability is $(256 \times 255 \times 254 \times 1)/(256 \times 255 \times 254 \times 253) = 1/253$. Using the same method that when $n_f = 6$, the probability is $(256 \times 255 \times 254 \times 1 \times 252 \times 1)/(256 \times \cdots \times 251) = 1/(253 \times 251)$. Therefore, when the number of faults n_f is even and n_f>2, the probability is $1/(253 \times 251 \times \cdots \times (257 - n_f))$.

In summary, it can be considered that when the size of $\mathcal{D}$ is 2, *i.e.*, $n_f = 2$, the remaining search space of each byte can only be reduced to 2. As for the rest of the cases, it can be reduced to unique excluding some special cases. For the simplification of the problem, in the following analysis, we ignored the discussion of these special cases when $n_f > 2$.

5.2 Analysis of the Remaining Key Search Space

Suppose n_f faults are injected, resulting in an uneven distribution of the output of Sbox. Since for the j-th element of the ciphertext, $c_j = s_j \oplus k_j^{R-1}$ (s_j is Sbox output, ignore the linear transformation) and k_j^{R-1} is constant during the encryption process, then the uneven distribution of s_j eventually leads to the uneven distribution of c_j. The probability distribution of c_j is shown in Eq.(15).

$$Pr(c_j) = \begin{cases} \frac{\lambda_0}{2^w} & \text{if } c_j = v_0^* \oplus k_j^{R-1} \\ \frac{\lambda_1}{2^w} & \text{if } c_j = v_1^* \oplus k_j^{R-1} \\ \vdots & \quad\vdots \\ \frac{\lambda_{n_s-1}}{2^w} & \text{if } c_j = v_{n_s-1}^* \oplus k_j^{R-1} \\ 0 & \text{if } c_j \in \mathcal{D} \oplus k_j^{R-1} \\ \frac{1}{2^w} & \text{otherwise} \end{cases} \tag{15}$$

Let T denote the number of different values of c_j in the 2^w ciphertexts encrypted by 2^w chosen plaintexts. Then the expectation of T can be calculated by Eq.(16). The first part in Eq.(16) is 2^w different values of c_j, and the second part is the expectation that a certain value does not appear in the 2^w ciphertexts.

$$\begin{aligned} E(T) &= 2^w - \sum_{c_j=0}^{2^w-1} [1 - Pr(c_j)]^{2^w} \\ &= 2^w - \left\{ (1 - \tfrac{\lambda_0}{2^w})^{2^w} + \cdots (1 - \tfrac{\lambda_{n_s-1}}{2^w})^{2^w} + \sum_{i=0}^{n_s-1} (\lambda_i - 1) + (2^w - \sum_{i=0}^{n_s-1} \lambda_i) \times (1 - \tfrac{1}{2^w})^{2^w} \right\} \\ &= 2^w - \left\{ \sum_{i=0}^{n_s-1} (\tfrac{2^w - \lambda_i}{2^w})^{2^w} + n_f + (2^w - \sum_{i=0}^{n_s-1} \lambda_i) \times (1 - \tfrac{1}{2^w})^{2^w} \right\} \end{aligned} \tag{16}$$

For the last round key K^{R-1}, the initial key search space of j-th element is $\{0, 1, \cdots, 2^w - 1\}$. What is the size of the remaining search space after using $k_j \neq c_j \oplus v^i$ to eliminate impossible values for T distinct c_j and n_f distinct v^i (v^i is the i-th element in $\mathcal{D}$)?

This problem can be abstracted into a probability model, *i.e.,* there are $2^w - 1$ distinct balls in the box, each time T balls are taken out and put back, how many balls have not been taken out after n_f times? The probability that value k_j remains in the search space of j-th element is denoted by $Pr(k_j)$ in Eq.(17). Especially in the case of $n_f = 2$, as we explained in Sec.5.1, the size of the remaining key search space of j-th element can only be reduced to 2.

$$Pr(k_j) = \begin{cases} (\frac{2^w-1-T}{2^w-1})^{n_f} & \text{if } k_j \neq k_j^{R-1} \\ 1 & \text{if } k_j = k_j^{R-1} \end{cases} \quad n_f \neq 2$$

$$Pr(k_j) = \begin{cases} (\frac{2^w-2-T}{2^w-2})^{n_f} & \text{if } k_j \neq k_j^{R-1} \text{ and } k_j \neq k_j^{R-1} \oplus v^0 \oplus v^1 \\ 1 & \text{if } k_j = k_j^{R-1} \text{ or } k_j = k_j^{R-1} \oplus v^0 \oplus v^1 \end{cases} \quad n_f = 2 \tag{17}$$

Taking $n_f \neq 2$ as an example, except that k_j^{R-1} will not be removed from the search space, the remaining $2^w - 1$ elements have the same probability of not being removed under n_f times, and the expectation of remaining space can be expressed as Eq.(18).

$$E_1 = \begin{cases} 1 + (2^w - 1) \times (\frac{2^w-1-E(T)}{2^w-1})^{n_f}, \ n_f \neq 2 \\ 2 + (2^w - 2) \times (\frac{2^w-2-E(T)}{2^w-2})^{n_f}, \ n_f = 2 \end{cases} \tag{18}$$

Therefore, the expectation of the remaining key spaces size of j-th element can be obtained by E_1. However, the above derivation is based on the fact that our guesses $\hat{k}_\theta$ and $\hat{\mathcal{D}}$ are both correct. In other cases, *i.e.,* when one of $\hat{k}_\theta$ or $\hat{\mathcal{D}}$ guesses wrong, the probability that k_j remains in the search space is Eq.(19). When the guess is wrong, k_j^{R-1} may also be removed from the key search space, causing the size of the search space to be 0.

$$Pr(k_j) = (\frac{2^w - T}{2^w})^{n_f} \tag{19}$$

Similarly, we will have the expectation of a wrong guess, as shown in Eq.(20).

$$E_2 = 2^w \times (\frac{2^w - E(T)}{2^w})^{n_f} \tag{20}$$

The total number of guesses is $2^w \times \prod_{i=0}^{n_s-1} \lambda_i$, of which only one case is correct. The last round key search space is $\mathbb{K}^{R-1}$. The expectation of $\mathbb{K}^{R-1}$ can be represented as Eq.(21). The first item (*i.e.,* $(E_1)^m$) in $E_{\mathbb{K}^{R-1}}$ is the number of key search space for the correct guess (the m-th power of E_1), and the second item is that case for the incorrect guesses.

$$E_{\mathbb{K}^{R-1}} = (E_1)^m + [(2^w \times \prod_{i=0}^{n_s-1} \lambda_i) - 1] \times (E_2)^m \tag{21}$$

5.3 The Required Number of Faulty Ciphertexts

For each element, suppose that the faulty ciphertexts are randomly and uniformly distributed and each element of ciphertext can eliminate n_f impossible values in the search space of each element, then the number of faulty ciphertexts required to eliminate $2^w - 1$ impossible values is N.

This problem can be abstracted as a variant of the coupon collection problem [BHS94]: Suppose there are $2^w - 1$ kinds of coupons, each of which has the same probability of being obtained and the coupons are also in unlimited supply. For each draw of n_f coupons, how many draws will it take to collect all $2^w - 1$ coupons?

Note that event $A_{t,i}$ is the case where the i-th coupon is still missed after the t-th draw, and N is the number of draws, then $Pr(N > t)$ represents the probability of the number of draws $N > t$, *i.e.,* after t times of draws, there are still coupons that have not been drawn.

$$\begin{aligned} Pr(N > t) &= Pr(\bigcup_{i=1}^{2^w-1} A_{t,i}) \\ &= \sum_{1 \le i \le 2^w-1} Pr(A_{t,i}) - \sum_{1 \le i < j \le 2^w-1} Pr(A_{t,i} \cap A_{t,j}) \\ &+ \sum_{1 \le i < j < k \le 2^w-1} Pr(A_{t,i} \cap A_{t,j} \cap A_{t,k}) + \cdots + (-1)^{2^w} Pr(\bigcap_{i=1}^{2^w-1} A_{t,i}) \end{aligned} \tag{22}$$

Among them, $Pr(\bigcap_{s=1}^{r} A_{t,i_s}) = (\frac{\binom{2^w-1-r}{n_f}}{\binom{2^w-1}{n_f}})^t, (1 \le r \le 2^w - 1)$, put it into Eq.(22) to get:

$$Pr(N > t) = \sum_{r=1}^{2^w-1} (-1)^{r+1} \binom{2^w - 1}{r} (\frac{\binom{2^w-1-r}{r}}{\binom{2^w-1}{t}})^t \tag{23}$$

The expectation of the number of draws N can be obtained as:

$$\begin{aligned} E(N) &= \sum_{t=0}^{\infty} Pr(N > t) = \sum_{t=0}^{\infty} \sum_{r=1}^{2^w-1} (-1)^{r+1} \binom{2^w - 1}{r} (\frac{\binom{2^w-1-r}{n_f}}{\binom{2^w-1}{n_f}})^t \\ &= \sum_{r=1}^{2^w-1} (-1)^{r+1} \binom{2^w - 1}{r} \sum_{t=0}^{\infty} (\frac{\binom{2^w-1-r}{n_f}}{\binom{2^w-1}{n_f}})^t = \sum_{r=1}^{2^w-1} (-1)^{r+1} \binom{2^w - 1}{r} \frac{\binom{2^w-1}{n_f}}{\binom{2^w-1}{n_f} - \binom{2^w-1-r}{n_f}} \end{aligned} \tag{24}$$

When $n_f = 1$, this is the original PFA with single fault, $E(N) = (2^w - 1) \cdot (1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{2^w-1})$, which also agrees with the result given in [ZLZ+18]. If n_f can satisfy its $E(N) \le 2^w$, the key search space can be reduced to a relatively small level by using 2^w faulty ciphertexts encrypted by 2^w chosen plaintexts.

5.4 Complexity Analysis of CPPFA

CPPFA enumerates the θ-th element k_θ $(0 \sim 2^w - 1)$ of the master key, utilizes Algorithm 2 to generate $\prod_{i=0}^{n_s-1} \lambda_i$ sets $\hat{\mathcal{D}}$ for each $\hat{k}_\theta$ and uses Eq.(5) to reduce the search space for each $\hat{\mathcal{D}}$ to obtain the final key. Therefore, the time complexity of CPPFA is $O(2^w \times \prod_{i=0}^{n_s-1} \lambda_i)$, and the worst time complexity is $O(2^w \times \prod_{i=0}^{n_f-1} 2) = O(2^{w+n_f})$ when $n_f = n_s$ $(n_f + n_s = \sum_{i=0}^{n_s-1} \lambda_i, \lambda_i \geq 2)$. Theoretically, the time complexity of CPPFA will increase with the increase of the number of faults n_f. When $n_f = 16$, the worst time complexity of CPPFA is $O(2^{24})$, which can be done on a personal computer in less than one hour.

6 CPPFA on Block Ciphers

Suppose we inject n_f faults into the Sbox of a block cipher, resulting in n_s sets $\mathcal{V}_i$ and the size of $\mathcal{V}_i$ is λ_i. In this section, we take into account the worst scenario, when $n_f = n_s$, $\lambda_i = 2$, and CPPFA has the worst time complexity of $O(2^{w+n_f})$.

6.1 CPPFA on AES-128

Without loss of generality, we can construct 256 chosen plaintexts on the first byte of AES as shown in Eq.(25), and encrypt them to obtain the corresponding 256 faulty ciphertexts.

$$\begin{array}{c} \text{0x00 112233445566778899AABBCCDDEEFF} \\ \text{0x01 112233445566778899AABBCCDDEEFF} \\ \vdots \\ \text{0xFF 112233445566778899AABBCCDDEEFF} \end{array} \tag{25}$$

Firstly, $\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_f-1}\}$ can be obtained by using Algorithm 1 for these 256 pairs of plaintext-ciphertext. The next step is to enumerate k_0. For each $\hat{k}_0$ (guess for k_0), we will utilize Eq.(12) to derive the corresponding $\{\hat{\mathcal{V}}_0, \hat{\mathcal{V}}_1, \cdots, \hat{\mathcal{V}}_{n_f-1}\}$ and these n_f $\hat{\mathcal{V}}_i$ can be used to identify certain candidate sets $\hat{\mathcal{D}}$ (guess for the impossible value set $\mathcal{D}$), as mentioned in Algorithm 2. When $\hat{k}_0 = k_0$ and $\hat{\mathcal{D}} = \mathcal{D}$, E_1 may be calculated using Eq.(18), and in other cases, E_2 can be calculated using Eq.(20). In the end, the total key search space $E_{\mathbb{K}^{R-1}}$ can be obtained by the formula $E_{\mathbb{K}^{R-1}} = (E_1)^{16} + (256 \times 2^{n_f} - 1) \times (E_2)^{16}$. The results of different n_f are shown in Table 3.

According to Table 3, it can be found that when $n_f \geq 5$, the remaining key search space $\mathbb{K}^{R-1}$ under 256 ciphertexts is rather tiny. At this point, as mentioned in Sec.4, $\mathbb{K}^{R-1}$ can be further reduced by the inverse key schedule algorithm without incurring significant time overhead. Because the key schedule algorithm of AES-128 is complex enough, it can be considered that $\mathbb{K}^{R-1}$ will become unique after optimization. The comparison of expected and simulation results is presented in the Table 7 in Sec.7.2, and it is found that the results are consistent with our conclusions. In addition, when $n_f < 5$, $\mathbb{K}^{R-1}$ is too large to enumerate the correct key by the inverse key schedule algorithm. By adding some extra ciphertexts generated from random plaintexts, CPPFA can further lower $\mathbb{K}^{R-1}$. Therefore, according to Sec.5.3, we can calculate the required number of ciphertexts for AES-128 as shown in the Table 4. The result in Table 4 minus 2^8 is the amount of extra ciphertexts required.

Table 3: The remaining key search space of AES-128.

Number of Faults (n_f)	Ciphertexts	E_1	E_2	$E_{\mathbb{K}^{R-1}}$
2	256	36.03309245	35.10030117	$2^{92.135}$
3	256	13.85204758	13.16230477	$2^{70.494}$
4	256	5.82847978	4.97713571	$2^{49.049}$
5	256	2.82932152	1.89771410	$2^{27.889}$
6	256	1.69885697	0.72956404	$2^{12.264}$
7	256	1.26920464	0.28278424	$2^{5.503}$
8	256	1.10455634	0.11050558	$2^{2.295}$
16	256	1.00007237	0.00008009	$2^{0.002}$

Table 4: The required number of faulty ciphertexts of AES-128.

Number of Faults (n_f)	$\lvert\mathbb{K}^{R-1}\rvert$	Ciphertexts	Number of Faults (n_f)	$\lvert\mathbb{K}^{R-1}\rvert$	Ciphertexts
2	2^{16}	$2^{9.617}$	6	1	$2^{8.000}$
3	1	$2^{9.028}$	7	1	$2^{7.807}$
4	1	$2^{8.607}$	8	1	$2^{7.577}$
5	1	$2^{8.299}$	16	1	$2^{6.539}$

6.2 CPPFA on LED

Table 5: The remaining key search space of LED-64.

Number of Faults	Ciphertexts	E_1	E_2	$E_{\mathbb{K}^{R-1}}$
2	16	3.45223536	2.64796188	$2^{29.530}$
3	16	1.91974848	1.29159325	$2^{15.346}$
4	16	1.47296749	0.70127570	$2^{8.942}$
5	16	1.27206405	0.42025498	$2^{5.555}$
6	16	1.17340719	0.27593595	$2^{3.691}$
7	16	1.12147714	0.19723916	$2^{2.646}$
8	16	1.09287734	0.15262403	$2^{2.050}$

Similar to AES-128, we construct 16 chosen plaintexts for the first nibble of LED-64. On LED, Algorithm 1 and Algorithm 2 are identical to AES. We can get the first elements of plaintext sets $\{\mathcal{P}_0, \mathcal{P}_1, \cdots, \mathcal{P}_{n_f-1}\}$, and the guess impossible set $\hat{\mathcal{D}}$ for $\hat{k}_\theta$. After the last round **SubCells (SB)** of LED, the state matrix will pass **ShiftRows (SR)** and **MixColumnsSerial (MC)**. In other words, the inverse **MC** and **SR** must be performed before analyzing the ciphertexts with Algorithm 3. In addition, LED lacks a key schedule algorithm, and its key is equivalent to the master key in each round. Therefore, we can directly compare whether the candidate value k_θ^{R-1} in $\mathbb{K}^{R-1}$ is equal to the guessed value $\hat{k}_\theta$. We also estimated the expectations mentioned in Section 5 for the LED, which are presented in the Table 5 and Table 6.

Table 6: The required number of faulty ciphertexts of LED-64

Number of Faults	$\lvert\mathbb{K}^{R-1}\rvert$	Ciphertexts	Number of Faults	$\lvert\mathbb{K}^{R-1}\rvert$	Ciphertexts
2	2^{16}	$2^{4.470}$	6	1	$2^{2.847}$
3	1	$2^{3.979}$	7	1	$2^{2.574}$
4	1	$2^{3.523}$	8	1	$2^{2.324}$
5	1	$2^{3.158}$			

According to the analysis steps of CPPFA, for each guessed key $\hat{k}_\theta$ $(0 \leq \hat{k}_\theta < 16)$, there will be a candidate key space $\mathbb{K}_{\hat{k}_\theta}^{R-1}$. If $\hat{k}_\theta$ does not exist in the remaining key search space of θ-th nibble, we can directly exclude the whole $\mathbb{K}_{\hat{k}_\theta}^{R-1}$.

6.3 Fault Recovery

Before this part, we only focused on key recovery in block ciphers. Another incidental property of CPPFA is that it can recover fault information at the same time as the key is recovered. When we utilize CPPFA to reduce the size of $\mathbb{K}$ to be unique, we can able to determine the $k_\theta = \hat{k}_\theta$, $\mathcal{D} = \hat{\mathcal{D}}$ and $\mathcal{V}_i = \hat{\mathcal{V}}_i$, where $\hat{k}_\theta$, $\hat{\mathcal{D}}$, and $\hat{\mathcal{V}}_i$ are the values corresponding to the unique remaining key. Furthermore, the size λ_i of each $\mathcal{V}_i$ and the number n_s of $\mathcal{V}_i$ are the information available to us by analyzing 2^w ciphertexts. Also, there is $n_f + n_s = \sum_{i=0}^{n_s-1} \lambda_i$, which means that the faults only affect the initial value of the members in $\mathcal{V}_i$. For each $\mathcal{V}_i$, a member v_i^s of $\mathcal{V}_i$ is the same output value after fault injection, and this member does not exist in the impossible value set $\mathcal{D}$. So the fault situation in Sbox can be summarized as: $\text{Sbox}[l] = v_i^j \to v_i^s$, $v_i^s \notin \mathcal{D}$, $v_i^s \in \mathcal{V}_i$ and $v_i^j \in \mathcal{V}_i$ $(0 \leq j < \lambda_i)$. Through the above method, we can recover the real faults in turn.

6.4 The Key Schedule Algorithm with Faulty Sbox

Previous works about PFA have ignored the fact that the faulty Sbox may also be used by the key schedule algorithm, such as AES. When the encryption algorithm and the key schedule algorithm share the same faulty Sbox, it is impossible to get the master key directly through the key of the last round by inverse key schedule algorithm without knowing the fault. The faulty Sbox will be accessed 40 times during the whole key schedule algorithm. Where $n_f = 1$, the probability that the faulty Sbox element is accessed is about $1 - (255/256)^{40} \approx 14.5\%$. And it will increase with the increase of the number of faults. Therefore, this problem is more prominent under the scenario of multiple faults.

With this in mind, let us first recall the inverse key schedule algorithm (See Eq.(26)) of AES. In generally, $\{W_{40}, W_{41}, W_{42}, W_{43}\}$ is input as the key of the last round K^{R-1}, and the master key $K^0 = \{W_0, W_1, W_2, W_3\}$ is output.

$$\begin{cases} W_{i-4} = W_i \oplus \textbf{SubWord}(\textbf{RotWord}(W_{i-1})) \oplus \text{Rcon} & i \bmod 4 \equiv 0 \\ W_{i-4} = W_i \oplus W_{i-1} & i \bmod 4 \not\equiv 0 \end{cases} \tag{26}$$

where Rcon is the round constant and the faulty Sbox is used in **SubWord**.

Due to **Subword** uses the same faulty Sbox in the inverse and forward key schedule algorithm, the derivation from K^{R-1} to K^0 is unique with a known faulty Sbox. Most of the PFA-related works that can recover the key of the last round under the scenario of multiple faults will bypasses faults identification and recovers the K^{R-1} directly. Without information about the faulty Sbox, even if K^{R-1} is obtained, the K cannot be obtained by K^{R-1}. However, CPPFA can perfectly solve this issue. Recall that Algorithm 2 (Line 7 $\sim$ 11) of CPPFA, according to **Feature 1**, we can additionally record the fault information $(\hat{v}^{j_0} \to \hat{v}^{i_0}, i_0 \neq j_0)$ for the $\hat{\mathcal{V}}_0$ and do the same for other $\hat{\mathcal{V}}_i$. When the remaining key search space of the last round is reduced to unique, the recorded fault at this time is the real fault. We can use this fault information and key of the last round to recover the real master key. In addition, CPPFA does not require additional information and does not increase the complexity of the analysis.

6.5 Attacks on Boolean Masking Countermeasure

If faulty value v' is injected into the x-th element of Sbox, where the initial value $\text{Sbox}[x] = v \to v'$, it results in a correspondingly computed error element in the masking Sbox, where $\text{Sbox}'[x \oplus m] = v \oplus m' \to v' \oplus m'$. Therefore, $v \oplus m'$ element is missing in the Sbox, and $v' \oplus m'$ elements are doubled. Furthermore, subsequent operations can remove the influence of m'. With the above information, it can be known that the boolean masking does not affect the collision of faulty ciphertexts. Therefore, the attacker does not need to consider whether the block cipher implementation is protected by basic boolean masking.

7 Experiment and Evaluation

7.1 Experiment Setup

We implemented 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 simulated injecting unknown faults into the Sbox of block cipher (2) We constructed 2^w chosen plaintexts and encrypted them. (3) Estimated the number of faults n_f from plaintexts and ciphertexts, if n_f is too small ($n_f \leq 4$ for AES and $n_f \leq 4$ for LED), added some additional random encrypted ciphertexts (4) We used CPPFA for analysis.

7.2 Attack on AES

We injected n_f faults into the Sbox of AES, and we performed 10,000 simulations for each fault. The experiments of CPPFA on AES can be divided into two cases. **Case 1**: Only use 256 chosen plaintexts and corresponding ciphertexts (Sec.5.2). Figure 3 shows the comparison of the average size of the remaining key search space per byte (Experiment) with the theoretical value (Theory) under different n_f. It is very obvious to notice that the two curves overlap very well. This in turn can confirm the correctness of our theoretical expectations (Sec.5.2). Besides, when $n_f = 5$, the size of the remaining key search space per byte is relatively small (approximately equal to 3). It is worth noting that the remaining search space for the last full key can be considered as the 16 power of one byte ($3^{16} \approx 2^{25.36}$). Conversely, when $n_f \leq 4$, the key search space is still too large to recover the unique key with only 256 samples. So, there is **Case 2**: The average number of ciphertexts N required to reduce the remaining key search space per byte to 1 under n_f faults (Sec.5.3). The number of ciphertexts required for one byte is overall in better agreement with the theoretical expectation (Sec.5.2). Furthermore, N for the last round full key search space $\mathbb{K}^{R-1}$ is represented by the blue curve ($|\mathbb{K}^{R-1}| = 1$). N for one byte and a full key search space is different. Comparing the blue and brown curves in Figure 4, N for a full key is about 1.53 times that of one byte.

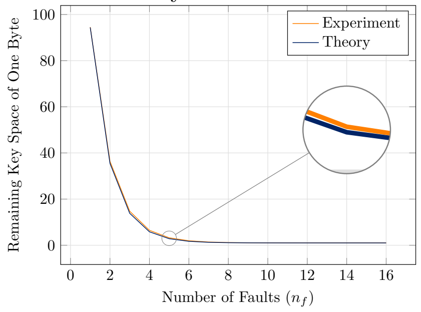


Figure 3: Single byte remaining key search space of 256 chosen plaintexts (AES-128).

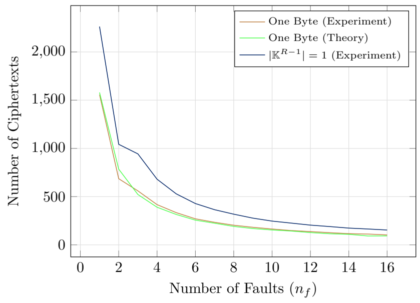


Figure 4: The number of required ciphertexts for the size of key search space to be one (AES-128).

In **Case 1**, if $\hat{k}_\theta \neq k_\theta$, there may still be some candidate keys K^{R-1} added to the space of the last round keys $\mathbb{K}^{R-1}$. For $K^{R-1} \in \mathbb{K}^{R-1}$, we can convert K^{R-1} into the master key K using the inverse key schedule algorithm. Then compare whether the θ-th byte in K are equal to $\hat{k}_\theta$, and if so, add it to the candidate space of the master key $\mathbb{K}$. This can further reduce the search space of the master key. Table 7 shows our experimental results under the two cases. It can be found that the key schedule algorithm can effectively reduce the search space of the AES master key, and the larger the n_f, the better the effect. In other words, when $n_f > 4$, 256 samples can reduce the search space of the master key to a sufficiently small amount. When $n_f \leq 4$, only a small number of additional random ciphertexts need to be added.

Table 7: Experimental Results of AES-128 and LED-64

Case 1								**Case 2**					
AES-128				LED-64				AES-128			LED-64		
n_f	$\lvert\mathbb{K}^{R-1}\rvert$	$\lvert\mathbb{K}\rvert$	N	n_f	$\lvert\mathbb{K}^{R-1}\rvert$	$\lvert\mathbb{K}\rvert$	N	n_f	$\lvert\mathbb{K}\rvert$	N	n_f	$\lvert\mathbb{K}\rvert$	N
5	$2^{27.04}$	$2^{11.06}$	256	5	$2^{17.58}$	$2^{10.31}$	16	1	1	1874.918	1	1	92.650
6	$2^{15.24}$	$2^{6.37}$	256	6	$2^{17.02}$	$2^{10.63}$	16	2	1	1041.390	2	1	40.750
7	$2^{7.69}$	1	256	7	$2^{17.53}$	$2^{10.43}$	16	3	1	905.505	3	1	54.110
8	$2^{3.70}$	1	256	8	$2^{19.16}$	$2^{10.31}$	16	4	1	524.037	4	1	35.642

7.3 Attack on LED

Unlike AES, CPPFA on LED only requires 16 chosen plaintexts. With the two cases, we also conducted 10,000 simulation experiments on LED. In **Case 1**, the experimental result of the LED (marked as orange curve) resembles a parabola, which does not comply to the theoretical expectation (5.2) when n_f is relatively large. In practical, when n_f is larger, the number of different ciphertexts is smaller. Assuming that only n_f ciphertexts in the 16 ciphertexts are the same, one ciphertext can reduce n_f key search spaces. Then the remaining key search space can be simply expressed as $16 - 16n_f + n_f^2$. The experimental results (marked as orange curve on Figure 5) are similar to the shape of this quadratic curve. It can be seen from Figure 5 that when n_f is $5 \sim 6$, the effect is best. Therefore, too many faults are not necessarily a good thing for LED. In **Case 2**, the experimental curves in Figure 6 appear less smooth compared to Figure 4. As we discussed in Sec.5.1, when $n_f = 2$, the key search space will become $\{k, v^0 \oplus v^1 \oplus k\}$. When n_f is odd, although the remaining key search space can be reduced to 1, the probability of the above-mentioned (Sec.5.1) for any two impossible values (v^i, v^j) will increase, making the reduction of the key search space less efficient and N increases. But overall, the larger n_f, the less N.

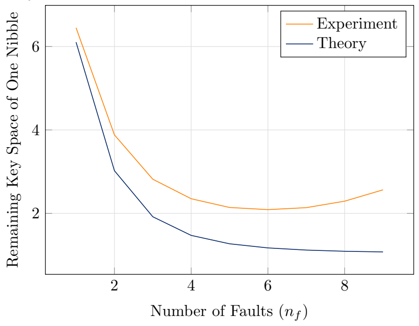


Figure 5: Single nibble remaining key search space of 16 chosen plaintexts (LED-64).

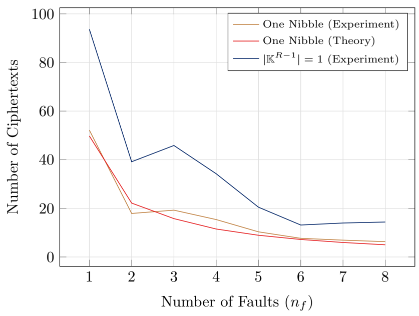


Figure 6: The number of required ciphertexts for the size of key search space to be one (LED-64).

The data of LED-64 with two cases are also shown in Table 7. In **Case 1**, because there is no key schedule algorithm in the LED, the master key and the last round key are identical. This relationship can be used to further reduce the remaining key search space by excluding the candidate key in $\mathbb{K}^{R-1}$ whose θ-th element is not equal to $\hat{k}_\theta$, but the effect is not as good as AES. Under 16 samples, the search space of the LED final master key $\mathbb{K}$ can be reduced to about 2^{10}, which is adequate for brute-force verification.

7.4 Comparison with Related Works

7.4.1 About the Actual Attack

The implementation of CPPFA on actual devices can be divided into three phases:

1). Fault injection. The difficulty of CPPFA is easier than PFA-20 [ZZJ+20], and the same difficulty as PMPFA [SBH+21]. PFA-20 uses Laser Fault Injection (LFI) to perform single-bit fault injection on the SRAM of an ATmega163L microcontroller.

The experimental results of PFA-20 show that LFI injecting faults at adjacent positions of two elements will cause multiple faults ($n_f = 2$) that can not be handled by PFA-20. Both original PFA and PFA-20 require accurate injection techniques and additional fault identification mechanisms. However, CPPFA can be more easily implemented due to its more relaxed fault model (unknown multiple faults) and less required ciphertexts. PMPFA [SBH+21] utilizes Electromagnetic Fault Injection (EMFI) to inject persistent faults on STM32F407VG microcontrollers. The advantage of EMFI is that fault injection can be done in a completely non-invasive manner, but the injection accuracy is worse than that of LFI and EMFI would cause multiple faults. Specifically, experimental results of PMPFA show that more than 87% of the faults affected multiple AES Sbox elements ($2 < n_f \leq 16$) and the ratio of single element fault was less than 10%. Furthermore, less than 3% of the faults affected more than half of the elements of the Sbox, which violated the fault model defined by the PMPFA and could not be handled. CPPFA and PMPFA have similar fault model, *i.e.,* they are suitable for multiple faults scenario and do not need to know the fault in advance.

2). Constructing chosen plaintexts. The chosen-plaintext technique is a kind of collision attack, which has been very mature for a long time [BK06]. The difficulty of CPPFA is more easier than PFCA [ZLZ+21]. For each byte of a random plaintext, PFCA takes any of 256 possible values, and the other bytes are fixed. In other words, PFCA needs to construct $256 \times 16 = 4096$ plaintexts. However, CPPFA only needs to construct 256 plaintexts for one byte. CPPFA reduces the number of chosen plaintexts from 4096 to 256, which makes the actual attack less difficult and more friendly. In addition to PFCA, Caforio and Banik [CB19] extended PFA to the Feistel networks in 2019, where they assumed that the adversary had access to faulty and fault-free ciphertexts. The adversary needs to encrypt twice with fixed plaintexts. In summary, the chosen-plaintext technique does not make PFA more difficult, and the cost of chosen-plaintext is worth it in terms of CPPFA results.

3). Fault analysis. The analysis of CPPFA is less difficult than PFA-20, EPFA [XZY+20] and APFA [ZFL+22], etc. These previous works require that the number of faults is one and the fault value is known in advance, which means that the adversary has a stronger capability. CPPFA has the ability to recovering the master key in a scenario with multiple unknown faults. CPPFA reduces the number of required ciphertexts to a constant 2^w, which is much smaller than previous works. In multiple faults scenario, one ciphertext of CPPFA can exclude n_f elements in the key search space, thus greatly improving the efficiency of PFA analysis. Compared to PMPFA [SBH+21] with the same number of faults and ciphertexts, CPPFA can reduce the key remaining search space to unique by the inverse key expansion algorithm without additional verification.

In short, CPPFA does not make it more difficult to attack on an actual device compared to previous works. It is even easier or the same on all phases.

7.4.2 About Time Complexity

Some previous PFA-related work is based on the statistics of ciphertexts, whose analysis complexity is difficult to estimate. SPFA can also cope with the multiple faults scenario, but its time complexity is $O(2^{50})$, which is a considerable runtime. For PFCA, which also introduces the chosen-plaintext technique, the time complexity in a single-byte fault scenario is $O(2^{23})$. Furthermore, under the scenario of multiple faults, the time complexity of PFCA is $O(2^{12} + 2^8 \times |\Lambda^*|^{15})$. The size of Λ^* is related to the hamming distance between the n_f faults. Assume that n_f faults make n_f values in the Sbox become the same, *i.e.,* there are $n_f + 1$ identical values in the Sbox. In this case, the time complexity of CPPFA is $O(2^8 \times (n_f + 1))$, which is only related to the number of faults. However, the time complexity of PFCA depends on the distribution of hamming distances between these n_f faults. The time complexity of PFCA falls into the worst case $O(2^{12} + 2^8 \times (n_f + 1)^{15})$ when faults affect adjacent bytes, which is a scenario that easily occurs in an actual fault. In addition, under the random fault model, CPPFA is only related to the fault set size λ_i, while PFCA is related to the first fault selected and the hamming distance between the faults. The time complexity ranges from $O(2^{12} + 2^8)$ to $O(2^{12} + 2^8 \times (2n_f)^{15})$, while the CPPFA is fixed. Therefore, the time complexity of CPPFA is more stable than PFCA.

8 Conclusion

In this paper, we introduced the chosen-plaintext technique and proposed a new analysis method called *Chosen-Plaintext based Persistent Fault Analysis* (CPPFA) under the problem that previous PFAs have difficulty identifying and utilizing multiple faults information. CPPFA does not need to know the fault in advance (allow for a more relaxed fault model). Instead, CPPFA estimates the fault by constructing chosen plaintexts and utilizes the obtained fault to recover the key. We applied CPPFA to AES-128 and LED-64. For both ciphers, CPPFA performed well. More specifically, if the number of faults is larger than 4, AES-128 only needs 256 pairs of plaintext and ciphertext to recover the full 128-bit key. For LED-64, 16 samples can reduce the key space to about 2^{10}. Furthermore, we analyzed the expectation of the remaining key search space and the number of required ciphertexts under the scenario of multiple faults. Therefore, it can be considered that this good performance can be verifed at both the theoretical and practical levels. In addition, we analyzed the time complexity of CPPFA to ensure the feasibility of CPPFA in most cases. Compared to the state-of-the-art works, CPPFA requires a much smaller sample size and is adaptable to the scenario of any number of faults. These advantages may be sufficient to offset the expense of plaintexts.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2020AAA0107700), by National Natural Science Foundation of China (62072398), by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, 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

[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.

[BHS94] Gunnar Blom, Lars Holst, and Dennis Sandell. *Problems and Snapshots from the World of Probability.* Problems and Snapshots from the World of Probability, 1994.

[BK06] Johannes Blömer and Volker Krummel. Fault based collision attacks on aes. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 106–120. Springer, 2006.

[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.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 181–194. Springer, 2007.

[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.

[ESP20] Susanne Engels, Falk Schellenberg, and Christof Paar. SPFA: SFA on multiple persistent faults. In *2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 49–56. IEEE, 2020.

[FJLT13] Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–118. IEEE, 2013.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[KOS10] Eike Kiltz, Adam ONeill, and Adam Smith. Instantiability of RSA-OAEP under chosen-plaintext attack. In *Annual Cryptology Conference*, pages 295–313. Springer, 2010.

[SBH$^+$21] Hadi Soleimany, Nasour Bagheri, Hosein Hadipour, Prasanna Ravi, Shivam Bhasin, and Sara Mansouri. Practical Multiple Persistent Faults Analysis. *Cryptology ePrint Archive*, 2021.

[SHP09] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on AES: A threat in violet. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 13–22. IEEE, 2009.

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

[Sta01] Nio Standardstechnology. Advanced Encryption Standard. *FIPS 197*, 2001.

[TL21] Honghui Tang and Qiang Liu. MPFA: an efficient multiple faults-based persistent fault analysis method for low-cost FIA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[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.

[ZFL$^+$22] Fan Zhang, Tianxiang Feng, Zhiqi Li, Kui Ren, and Xinjie Zhao. Free Fault Leakages for Deep Exploitation: Algebraic Persistent Fault Analysis on Lightweight Block Ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 289–311, 2022.

[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.

[ZLZ$^+$21] Shihui Zheng, Xudong Liu, Shoujin Zang, Yihao Deng, Dongqi Huang, and Changhai Ou. A persistent fault-based collision analysis against the advanced encryption standard. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(6):1117–1129, 2021.

[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.