A Framework for the Analysis and Evaluation of Algebraic Fault Attacks on Lightweight Block Ciphers

Fan Zhang, Shize Guo, Xinjie Zhao, Tao Wang, Jian Yang, Francois-Xavier Standaert, Dawu Gu

Abstract—*Algebraic fault analysis* (AFA), which combines algebraic cryptanalysis with fault attacks, has represented serious threats to the security of lightweight block ciphers. Inspired by an earlier framework for the analysis of side-channel attacks presented at EUROCRYPT 2009, a new generic framework is proposed to analyze and evaluate algebraic fault attacks on lightweight block ciphers. We interpret AFA at three levels: the target, the adversary and the evaluator. We describe the capability of an adversary in four parts: the fault injector, the fault model describer, the cipher describer and the machine solver. A formal fault model is provided to cover most of current fault attacks. Different strategies of building optimal equation set are also provided to accelerate the solving process. At the evaluator level, we consider the approximate information metric and the actual security metric. These metrics can be used to guide adversaries, cipher designers and industrial engineers. To verify the feasibility of the proposed framework, we make a comprehensive study of AFA on an ultra-lightweight block cipher called LBlock. Three scenarios are exploited which include injecting a fault to encryption, to key scheduling, or modifying the round number or counter. Our best results show that a single fault injection is enough to recover the master key of LBlock within the affordable complexity in each scenario. To verify the generic feature of the proposed framework, we apply AFA to three other block ciphers, i.e., DES, PRESENT and Twofish. The results demonstrate that our framework can be used for different ciphers with different structures.

Index Terms—Algebraic fault analysis (AFA), Lightweight block cipher, LBlock, CryptoMiniSAT, Security evaluation

1 Introduction

1.1 Background

Data security gets more demanding under resource-constrained environments. Lightweight block ciphers are a cutting-edge technology to provide an efficient and power-saving solution. Frequently used lightweight block ciphers include PRESENT, Piccolo, LED, and LBlock. Most of these ciphers can be implemented with less than 3000 gate equivalents. The complexity of traditional cryptanalysis increases exponentially with the number of rounds. From a theoretical point of view, these ciphers are deemed secure if the number of rounds is sufficiently high.

Fault attack can retrieve secret information by actively injecting faults into the cryptosystem. Faults can be generated by changing the power supply voltage, changing the frequency of the external clock, varying the temperature or exposing the circuits to lasers during the computation [1]. The idea was first reported on RSA-CRT by Boneh et al. in 1996 [2]. Later, Biham and Shamir proposed a *differential fault analysis* (DFA) attack on the block cipher DES, which combines a fault attack with differential cryptanalysis [3]. Since then, DFA has been used to break various block ciphers. Traditionally, DFA on block ciphers is mostly conducted through manual analysis. When facing fault injection in a deep round, the fault propagation paths will overlap. The complexity of the analysis among overlapping paths increases exponentially, which is very difficult for the further manual analysis. This also happens when the number of flipped bits is large. A large size is easy for the injections, but it increases the difficulty of the analysis.

To overcome the difficulty of DFA, recent work [4] shows that algebraic cryptanalysis [5] can be combined with fault analysis. A machine solver can be used to automatically recover the secret key. This technique is referred to as *algebraic fault analysis* (AFA). AFA was proposed by Courtois et al. [4] in 2010. They showed that if 24 key bits are known and two bits in the 13-th round are altered, DES can be broken with a single fault injection in 0.01 hour. The full attack requires about 2^{19} hours and works 10 times as fast as the brute force attack. Considering their design principles, cryptographic devices with lightweight block ciphers are more vulnerable to fault attacks. Moreover, it is less complicated to solve the algebraic equations for

Fan Zhang is with the College of Information Science and Electrical Engineering, Zhejiang University, China. He is also with the Science and Technology on Communication Security Laboratory. E-mail: fanzhang@zju.edu.cn.
Shize Guo and Xinjie Zhao are with the Institute of North Electronic Equipment, Beijing, China. E-mail: nsfgsz@126.com, zhaoxinjieem@163.com.
Tao Wang is with the Department of Information Engineering, Ordnance Engineering College, Hebei, China. E-mail: twangdrsjz@aliyun.com
Jian Yang is with the Department of Computer Science and Engineering, University of Notre Dame, USA. Email: jyang9@nd.edu.
Francois-Xavier Standaert is with the UCL Crypto Group, Belgium. E-mail: fstandae@uclouvain.be.
Dawu Gu is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. E-mail: dwgu@sjtu.edu.cn.

lightweight block ciphers due to their relatively simple structure, making fault exploitations much easier. Zhao et al. [6] and Jovanovic et al. [7] extended AFA to lightweight block ciphers, such as LED. In [6], they used only one fault injection to recover the master key of LED in one minute with a PC. In 2013, Zhang et al. [8] proposed an improved AFA, which showed that the secret key of Piccolo can be recovered with only one fault injection. Zhao et al. [9] also got a more precise estimation of the LED key search space using AFA.

1.2 Motivation

Previous AFA mostly focused on one particular block cipher. The motivation of this paper is to standardize the process of AFA and provide a generic framework to analyze fault attacks on different block ciphers, especially on the lightweight ones. In practice, many situations are more challenging. Usually, faults are injected into a state before the linear layer that will bring the diffusion. For example in AES, a fault can be injected into the output of the key addition or substitution, as long as the place for the injection is before the MixColumn layer. However from the adversary's point of view, it is straightforward to ask the question: *where else can I inject a fault during the encryption?* A smart attacker may jump out of the box at a specific state and focus on a local index variable referred to as the *round counter*. Lightweight ciphers have a simple structure for efficiency reasons, but require more rounds to guarantee security. We aim to investigate how fault injections can modify the number of rounds, and how leakages could be used in algebraic fault attacks. The extended case of injecting faults both inside and outside the encryption module therefore requires a thorough study.

1.3 Our Work

In this paper, we make a comprehensive study on algebraic fault attacks on block ciphers.

In Section 2, we first give the formal description of algebraic fault analysis on block ciphers. We can describe AFA from three levels: the target, the adversary and the evaluator. At the target level, the design and implementation of cryptographic schemes are considered from three aspects. At the adversary level, we describe the capability of an adversary in four parts. At the evaluator level, we consider two metrics: the *approximate information metric* and the *actual security metric*. These metrics can help us to answer two types of questions: for adversaries, *What faults should I inject and how?* For cipher designers and industrial engineers, *How secure is my design?* and *How secure is my implementation?*

To verify the feasibility of the proposed framework, we make a comprehensive study of AFA on an ultra-lightweight block cipher called LBlock [10]. In Section 3, we first describe LBlock and related fault attacks. Then, we present how to build the algebraic equation set and provide the strategies on how to solve the equation set. Different fault models are considered. In Section 4, we evaluate LBlock against fault injections in the encryption procedure. In Section 5, we conduct fault attacks on the key scheduling of LBlock. Inspired by previous work [11], [12], in Section 6 we finally investigate four cases where faults are injected to a round number or a round counter. Under each case, our best results show that we can recover the key with only one fault injection.

To verify the generic feature of the proposed framework, we apply AFA to evaluate some other block ciphers against fault attacks in Section 7. The first target is DES [13], a standard cipher selected by NIST. The result shows that, under the single bit fault model, the attack efficiency is quite different when the fault location varies. If a single fault is injected into a certain bit in the 12-th round or any bit in the 11-th round of DES, the remaining entropy of the master key can be reduced to 5 or 0, respectively. The second one is PRESENT [14]. It has become a standard lightweight block cipher and has been deployed in many applications. The results show that, if single fault is injected in the 28-th round of PRESENT with 80-bit key length, the remaining entropy of the master key can be reduced to less than 30 with 35% probabilities. For most of the instances, two injections can recover the master key within three minutes. The third one is Twofish [15], a very complicated cipher and one of the AES candidates. The results show that if a single byte fault is injected into the last round of Twofish, about 280 fault injections can recover the key within 24 hours.

In Section 8, we conclude the paper and list some future work.

2 Proposed AFA Framework

In order to overcome the disadvantage of DFA, we propose a generic framework for AFA, which considers three levels: the target, the adversary and the evaluator. The framework tries to standardize the process of AFA and provides a unified solution which could evaluate different targets and adversaries.

2.1 The Target Level

The target level covers two aspects: *design* and *implementation*. The cryptographic design refers to the *cipher* which utilizes some ideal functions to solve cryptographic problems. For example, LBlock [10] is a cipher. The cryptographic implementation includes two parts: *code* and *device*. The cryptographic *device* refers to the hardware platform to implement the encryption/decryption functions of the *cipher*. For example, a smart card running the LBlock algorithm can be a target device. The cryptographic *code* comprehends the engineering effort of converting the theoretical *cipher* into practical programming code running on the *device*. For example, LBlock has size-optimized and speed-optimized versions in terms of programming code. The target level depicts

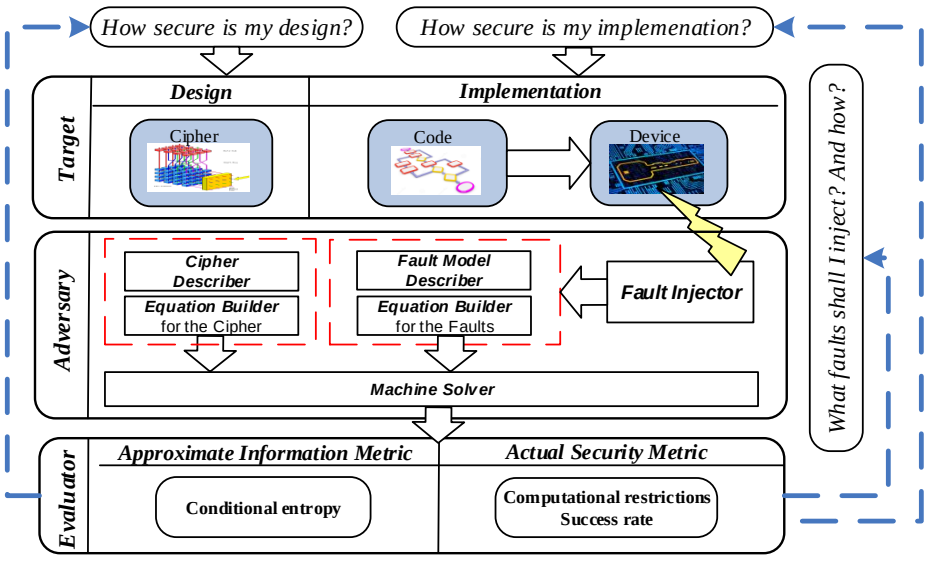


Figure 1: The proposed framework for AFA

how a cryptographic code is implemented on a specific device.

Possible targets include block ciphers, stream ciphers, hash functions, message authentication codes (MACs) etc. For this paper, we carefully chose four block ciphers: LBlock [10], DES [13], PRESENT [14] and Twofish [15]. LBlock, DES and Twofish have a Feistel structure while PRESENT has an SPN structure. LBlock is quite new but efficient. There is not much work known about it. DES is quite old but well-known. Twofish requires some complicated operations such as modulo addition, key-dependent S-Boxes, and the Pseudo-Hadamard Transform (PHT), which make fault attacks difficult. PRESENT is one of the most famous lightweight block ciphers with an SPN structure. Both LBlock and PRESENT are lightweight. The large number of applications to the aforementioned ciphers demonstrates the universality of our framework.

2.2 The Adversary Level

In our framework, an adversary's capability is characterized by four factors: *the cipher describer*, *the fault model describer*, *the fault injector* and *the machine solver*. The cipher describer refers to its capability of giving the formalizations of the cryptographic codes. The fault model describer depicts the attributes of faults to be injected. Both describers are implemented as public interfaces and supported by *equation builders* which automatically transfer those from describers into algebraic equations. *The fault injector* is in charge of injecting the fault into the device [1]. Finally, *the machine solver* takes the equations as inputs and solves them using mathematical automata.

There are three important stages at this level: ① *Fault Injection*, ② *Equation Building* and ③ *Equation Solving*, which are performed by the fault injector, the describer/builder, and the machine solver in Figure 1, respectively. Figure 2 shows the details of how the adversary level works.

2.2.1 The fault injector

In Figure 2, Stage ①, i.e., *Fault Injection*, indicates where the fault is injected. Previous work focused on the injections in encryptions. It is possible to extend the scenarios. Inside the encryption, the fault, denoted as f, could be injected into an intermediate state for different linear or non-linear operations, or a state for storing the total number of rounds, or an instant state called *round counter*. Outside the encryption, f might also be induced to other components such as key scheduling.

There are many practical methods to inject faults, such as optical radiation, clock glitch, critical temperature change, and electromagnetic emission. How to inject faults is discussed in [1], which is out of the scope of this paper. We focus here on three fault models (bit-based, nibble-based and byte-based) and conduct injections with simulations.

2.2.2 The fault model describer and its equation builder

In Stage ②, i.e., *Equations Building*, the adversary needs to build the equations for the faults.

A formal model F describes what the fault is and how it is related to the cipher. Here, X is an intermediate state. X_i is a unit of X, which determines how X is organized. f is the injected fault. w is the width of f. The fault width w is the maximal number of bits affected by one fault. The value of w might be 1, 4, 8, which refers to *bit-based*, *nibble-based*, and *byte-based fault models*, respectively. X^* is a faulty state where faults are injected. r is the index for a specific round. r_{max} is the round number, i.e., the total number of rounds. X has different meanings. It can be a state in r-th round of the key scheduling or encryption, thus X is written as X_r^{ks} or X_r^{en}. It can also be a state referred to as the

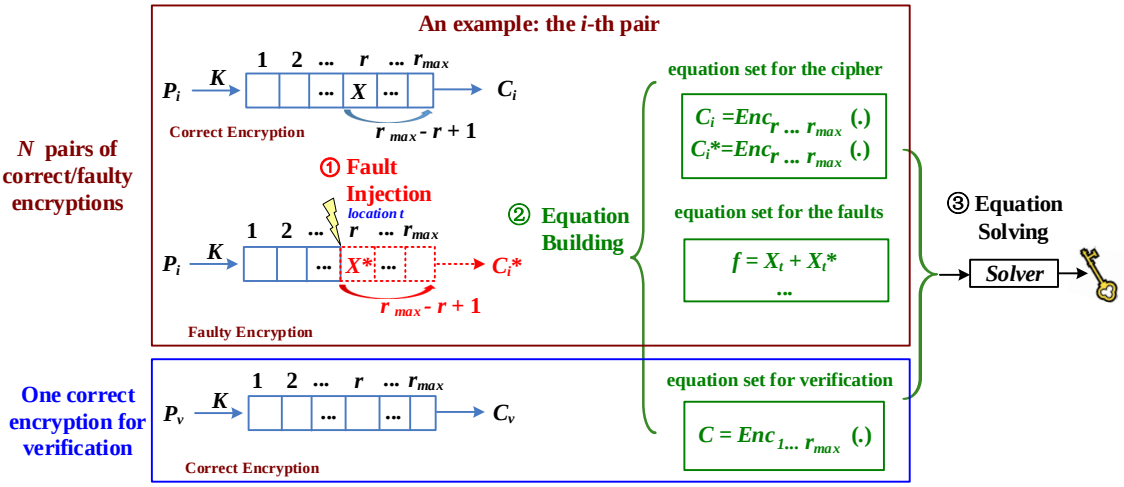


Figure 2: The adversary level of AFA framework

round counter, thus X can be depicted as $X_{\text{rc}}^{\text{ks}}$ or $X_{\text{rc}}^{\text{en}}$ respectively.

Two terms are used throughout this paper. ***Position***, denoted by X, is the state where the fault is situated. It refers to the round in most of cases. ***Location***, denoted by t, is the place where the fault is located inside a state. As in most previous fault attacks [3], we assume that only one unit of X, i.e., X_t, is erroneous with a single fault injection in this paper. This usually happens in fault attacks to the software implementations of block ciphers. For hardware implementations, multiple units of X might become faulty after a single fault injection. In general, λ, the size of the state, is larger than w. Thus there are m possible locations for f where $m = \lambda/w$. m denotes the maximal value for the number of possible locations for fault injection. t can be known or unknown depending on the scenarios.

A formal fault model can be described as a tuple of five elements $\mathsf{F}(X, \lambda, w, t, f)$. Basically, it tells us that a fault with value f and width w is injected at location t with respect to a state (or position) X having λ bits.

The injected faults are also represented with algebraic equations. Different parameters such as width w and location t should be considered. The equation set for the faults can be merged with the one for the entire encryption, which can significantly reduce the computation complexity. There is an option to build an additional equation set for verification purposes. It is based on the correct full round encryption of a known plaintext P_v, resulting in a corresponding ciphertext C_v. This equation set enforces the number of solutions to be one.

2.2.3 The cipher describer and its equation builder

Stage ②, i.e., *Equations Building*, specifies how to construct the equation sets for the cipher. Enc stands for the encryption function. The plaintext, the ciphertext, the master key and the state are denoted by P, C, K, X respectively. On the one hand, the building work has to include *all* the major components in both encryption and key scheduling. On the other hand, it should represent *every* operation. The most difficult part is how to represent the non-linear operations such as S-Box and modulo addition. More details can be found in [16]. In order to accelerate the solving speed, different strategies can be applied to the solver. For example, as to AFA on block ciphers with SPN structure, it is better to use the pair of correct and faulty ciphertexts to build the equations reversely [9]. In Figure 2, a fault is injected to X in the r-th round. The equation set is built for the last $(r_{max} - r + 1)$ rounds.

2.2.4 The machine solver

Stage ③, i.e., *Equation Solving*, specifies how to solve the entire equation set. Many automatic tools, such as Grobner basis-based [19] and SAT-based [18] solver, can be leveraged. The adversary could choose his own according to his skill set.

2.3 The Evaluator Level

The evaluator level takes the output of machine solvers and evaluates two metrics: the *approximate information metric* and the *actual security metric*. The evaluator answers two types of questions: for adversaries, *What faults should I inject and how?* For cipher designers and industrial engineers, *How secure is my design?* and *How secure is my implementation?*

2.3.1 Actual security metric

There are two types of security metrics. One is the computational restrictions. The possible criteria of the restrictions can be time complexity (such as the threshold for the timeout and the entire solving time, denote by t_{out} and t_{sol} respectively), the data complexity (such as the number of fault injections, denoted by N), and the space complexity (such as the memory cost). The

other is the success rate (denoted by SR) for extracting the master key. All these objective metrics are either measurable or computable, thus they can be used to evaluate and compare different factors that may affect algebraic fault attacks.

2.3.2 Approximate information metric

The *information metric* refers to the conditional entropy of the secret key after N fault injections. It is denoted by $\phi(K)$. In traditional DFAs, the adversary cannot analyze deeper rounds due to the overlap among propagation paths. The full utilization of all faults can be easily done in our AFA framework. The remaining key search space (denoted by $2^{\phi(K)}$) is equivalent to the number of satisfiable solutions if the multiple solution output is supported. Note that if the number of fault injections is small or the fault position is deep, the number of solutions might be too big to search them all. In this case, we can feed κ guessed bits of the secret key into the equation set. As opposed to [17], our information metric actually calculates an approximation to the theoretical complexity of the key search, which can serve as an additional criterion to conduct the evaluations.

3 Preliminaries of AFA on LBlock

LBlock [10] is an ultra-lightweight block cipher presented by Wu et al. in CANS 2011. It uses a 32-round Feistel structure with a block size of 64 bits and a key size of 80 bits. The design of LBlock well balances the trade-off between security and performance. On the one hand, only 1320 gate equivalents and 3955 clock cycles are required for hardware and software implementation respectively, which is outperforming many proposed lightweight block ciphers under mainstream architectures [18], [19]. The good efficiency makes it very suitable for resource constrained environments. On the other hand, LBlock remains still secure under modern cryptanalysis. It is worth taking a comprehensive investigation to its security features. We are interested in its resilience against fault attacks.

In this section, we first provide the design of LBlock and list related cryptanalysis. Then, the general representations of the equation set for both LBlock and the faults are described.

3.1 The Cipher of LBlock

Algorithm 1 shows the encryption of LBlock. Let $P = X_1 \| X_0$ denote the 64-bit plaintext and $C = X_{32} \| X_{33}$ denote the ciphertext, where X_i is 32 bits. $r_{max} = 32$ is the total number of rounds. rc is the round counter.

The round function F is a non-linear function with a 32-bit input. It consists of Key Addition (AK), Substitution (SB) and Linear Permutation (PM). $F = PM(SB(AK(X, K_i)))$.

- AK: the leftmost 32 bits of F function input are bitwise exclusive-ORed with a round key

Algorithm 1: The Encryption of LBlock

```
r_max = 32 ;
P = X_1 || X_0 ;
for rc = 0; rc < r_max; rc++ do
    X_{rc+2} = F(X_{rc+1}, K_{rc+1}) + (X_rc <<< 8) ;
end
C = X_32 || X_33 ;
```

- SB: the substitution uses every 4 bits of the exclusive-OR results as index for eight different 4-bit S-Boxes, $s_0, s_1, \ldots, s_7$
- PM: a permutation of eight 4-bit words Z ($Z = Z_7||Z_6||\ldots||Z_0$) to U ($U = U_7||U_6||\ldots||U_0$), and it can be illustrated as the following equations:

$$\begin{aligned} &U_7 = Z_6, U_6 = Z_4, U_5 = Z_7, U_4 = Z_5, \\ &U_3 = Z_2, U_2 = Z_0, U_1 = Z_3, U_0 = Z_1 \end{aligned} \tag{1}$$

Algorithm 2 shows the key scheduling of LBlock. The master key is denoted by $K = k_{79}||k_{78}||\ldots||k_0$. The leftmost 32 bits of K are used as the first round key K_1. Left32(L) denotes a function to get the leftmost 32 bits of L, where L is a state register of 80 bits. l_i is one bit of L. The other round keys K_{i+1} ($i = 1, 2 \ldots 31$) are generated according to Algorithm 2.

Algorithm 2: The Key Scheduling of LBlock

```
r_max = 32 ;
L = K ;
K_1 = Left32(L) ;
for rc = 1; rc < r_max; rc++ do
    L <<< 29 ;
    [l_79||l_78||l_77||l_76] = s_9[l_79||l_78||l_77||l_76] ;
    [l_75||l_74||l_73||l_72] = s_8[l_75||l_74||l_73||l_72] ;
    [k_50||k_49||k_48||k_47||k_46] ⊕ [rc] ;
    K_{rc+1} = Left32(L) ;
end
```

LBlock has two software implementations [10]. In the size-optimized implementation, eight 4-bit S-Boxes and 4-bit word permutations are used. In the speed-optimized implementation, the eight S-Boxes and the permutations can be implemented as four 8-bit lookup tables. No additional permutation is required. In the rest of this paper, we mainly focus on fault attacks on the software implementation of LBlock.

3.2 Related Fault Attacks on LBlock

Regarding the fault attacks, Zhao et al. [20] proposed the first fault attack on LBlock with DFA. Their best results showed that if a single-bit fault is injected into any round between the 24th and the 31st round, at least 8 fault injections are required to extract the master key. In 2013, Jeong et al. [21] presented an improved DFA on LBlock under nibble-based fault model. It requires 5 fault injections into the left input register of the 29th round, or 7 injections into the one of the 30th round. Chen et al. [22] built eight 8-round integral distinguishers of LBlock and proposed several integral based fault attacks.

When faults are induced into the right part at the end of the 24th round under random nibble fault model, 24 fault injections are required to recover the master key of LBlock. When faults are induced into the right part at the end of the 23rd round under semi-random nibble model, 32 fault injections are required. Li et al. [23] presented the first AFA on LBlock. Under nibble-based fault model in the 27th round, two fault injections are enough to recover the 80-bit master key.

3.3 Building the Equation Set for LBlock

3.3.1 Representing the overall encryption

The equations for the overall encryption have already been listed in Algorithm 1 (Line 4) where $0 \leq i \leq 31$.

$$X_{i+2} = F\left(X_{i+1}, K_{i+1}\right) + \left(X_i <<< 8\right) \tag{2}$$

3.3.2 Representing AK

Suppose $X = (x_1, x_2, ..., x_{32})$ and $Y = (y_1, y_2, ..., y_{32})$ are the two 32-bit inputs to the AK of LBlock. $Z = (z_1, z_2, ..., z_{32})$ is the output. AK can be represented as

$$x_i + y_i + z_i = 0,\ 1 \leq i \leq 32 \tag{3}$$

Note that the XOR operation in key scheduling (Line 8 in Algorithm 2) can also be represented with Equation 3. rc can be considered as one input whose value is known.

3.3.3 Representing SB

In LBlock, eight S-Boxes $s_0, s_1, \cdots, s_7$ are used in encryption and the other two s_8, s_9 are used in key scheduling. Let the input of S-Box be $(x_1\|x_2\|x_3\|x_4)$ and the output be $(y_1\|y_2\|y_3\|y_4)$. We adopt the method in [24] and represent each S-Box with four equations. For example, the equations for s_0 can be represented as:

$$\begin{aligned} 1 + x_1x_2x_4 + x_1 + x_1x_3 + x_3x_4 + x_2x_4 + y_1 = 0 \\ 1 + x_1x_2x_4 + x_1x_2x_3 + x_1 + x_4 + x_1x_2 \\ + x_2x_3 + x_2x_4 + x_1x_4 + y_2 = 0 \\ 1 + x_1 + x_2 + x_4 + x_2x_3 + x_2x_4 + y_3 = 0 \\ x_1 + x_2 + x_3 + x_4 + x_1x_2 + y_4 = 0 \end{aligned} \tag{4}$$

3.3.4 Representing PM

Let the input and output of PM be $(x_1\|x_2\|\ldots\|x_{32})$ and $(y_1\|y_2\|\ldots\|y_{32})$ respectively. The i-th bit of the input can be mapped to the i-th bit of the vector M using Table 1.

Table 1: Permutation Sector M

i	1	2	3	4	5	6	7	8
$M[i]$	9	10	11	12	1	2	3	4
i	9	10	11	12	13	14	15	16
$M[i]$	13	14	15	16	5	6	7	8
i	17	18	19	20	21	22	23	24
$M[i]$	25	26	27	28	17	18	19	20
i	25	26	27	28	29	30	31	32
$M[i]$	29	30	31	32	21	22	23	24

The PM function can be expressed as

$$x_i + y_{M[i]} = 0,\ 1 \leq i \leq 32 \tag{5}$$

3.3.5 Representing l-bit left cyclic shift

Suppose there is an l-bit left cyclic shift to a state register of m bits. LBlock adopts one 8-bit left cyclic shift in encryption ($l = 8, m = 32$) and one 29-bit left cyclic shift in key scheduling ($l = 29, m = 80$). Both can be written as the following equation when the input is $(x_1\|x_2\|\ldots\|x_m)$ and the output is $(y_1\|y_2\|\ldots\|y_m)$. $\%$ stands for a modulo operation.

$$x_{(l+i-1)\ \%\ m\ +\ 1} + y_i = 0,\ 1 \leq i \leq m \tag{6}$$

Using Equations 2 to 6, each round of key scheduling can be represented with 196 variables and 244 CNF equations; while each round of encryption can be represented with 304 variables and 496 CNF equations. The script size of one full LBlock encryption is 449KB.

3.4 Building the Equation Set for Faults

Let X denote the λ-bit correct data unit of LBlock. $X = x_1\|x_2\|\ldots\|x_\lambda$. X might represent a 32-bit left state register in the encryption ($\lambda = 32$), or an 80-bit key register in the key scheduling ($\lambda = 80$). Let Y denote the faulty value of X. $Y = y_1\|y_2\|\ldots\|y_\lambda$. There are m possible locations for the injected faults where $m = \lambda/w$. Let Z denote the fault difference of X and Y:

$$Z = z_1\|z_2\|\cdots\|z_\lambda, z_i = x_i + y_i,\ 1 \leq i \leq \lambda \tag{7}$$

Then, Z can be divided into m parts: $Z_1\|Z_2\|\ldots\|Z_m$

$$Z_i = z_{w\times(i-1)+1}\|z_{w\times(i-1)+2}\|\cdots\|z_{w\times i}, 1 \leq i \leq m \tag{8}$$

According to whether the adversary knows the exact location t ($1 \leq t \leq m$) or not, the algebraic equation representation of Z may have different formats.

3.4.1 Representing the fault with known t

Suppose t is known. Then Z can be denoted as

$$Z_i = 0,\ 1 \leq i \leq m, i \neq t \tag{9}$$

Z_t has a nonzero value of w-bits. We introduce a single bit variable u_t to represent that Z_t is faulty.

$$u_t = (1 \oplus z_{w\times(t-1)+1})(1 \oplus z_{w\times(t-1)+2})\cdots(1 \oplus z_{w\times t}) = 0 \tag{10}$$

Using Equations 9 and 10, Z can be represented with $w + 1$ variables and $w(m + 1) + 2$ CNF equations.

3.4.2 Representing the fault with unknown t

In practical attacks, the fault location t may be unknown. We introduce a variable u_i of m bits to represent whether Z_i is faulty or not.

$$\begin{aligned} u_i = &\left(1 \oplus z_{w\times(i-1)+1}\right)\left(1 \oplus z_{w\times(i-1)+2}\right)\cdots \\ &\left(1 \oplus z_{w\times i}\right),\ 1 \leq i \leq m \end{aligned} \tag{11}$$

If $u_i = 0$, Z_i will be the variable that is associated with the w-bit fault. Assuming that one and only one

fault is injected, there should be only one zero among $u_1, u_2, \ldots, u_m$. This constraint can be represented as:

$$\begin{aligned} &(1-u_1) \vee (1-u_2) \vee \cdots \vee (1-u_m) = 1, \\ &u_i \vee u_j = 1,\ 1 \le i < j \le m \end{aligned} \tag{12}$$

Using Equations 11 and 12, Z can be represented with $m(w+2)$ variables and $m(2w+0.5m+1.5)+1$ CNF equations. These equations can also be represented when different values of w, m and λ are given.

3.5 Equation Solving Strategies

In this paper, we choose CryptoMiniSAT v2.9.6 as our equation solver. It has two modes. Mode A works with a pair of known plaintext P_v and corresponding ciphertext C_v, which enforces the number of solutions to be one all the time. The purpose of this mode is to get the statistics of different solving times with different numbers of fault injections, which is one type of the actual security metrics mentioned in Section 2.3.1. Mode B works without (P_v, C_v). The solver is running a multiple solution mode to estimate $\phi(K)$, the remaining entropy of the master key. It is the approximate information metric mentioned in Section 2.3.2.

Next we describe how to use CryptoMiniSAT to roughly estimate $\phi(K)$ given N fault injections under Mode B. Let len denote the key length and κ denote the number of guessed secret bits fed into the solver. To estimate $\phi(K)$, κ is usually chosen from a larger value to a smaller one. Let $\eta(\kappa)$ denote the number of solutions for given κ. When the number of solution for one AFA is larger than 2^{18}, it is difficult for CryptoMiniSAT to find out all possible solutions within affordable time. In this case, a threshold τ for the maximal number of solutions can be set as $\tau = 2^{18}$. The detailed algorithm is shown in Algorithm 3.

Algorithm 3: Estimate $\phi(K)$ under Mode B

```
input : len, N, τ
output: φ(K)
GenerateAFAES(N);
GenKnownKeySet (S_k);
for κ=len; κ >-1; κ − − do
    FeedRandKeyBits(S_k) ;
    RemoveRandKeyBit(S_k);
    RunAFAModeB();
    CalcSolutionCount(η(κ));
    if η(κ) ≥ τ and κ > 0 then
        φ(K)=κ + log2(η(κ));
        break;
    end
    if η(κ) ≤ τ and κ==0 then
        φ(K)=log2(η(κ));
    end
end
```

In Algorithm 3, `GenerateAFAES` generates the equation set of the last few rounds after the fault is injected. `GenKnownKeySet` generates the value of the known key bits into set S_k. S_k is initialized to len (80 for LBlock) bits of the secret key. `FedRandKeyBits` feeds the value of κ key bits in S_k to the equation set. `RemoveRandKeyBit` removes one random key bit from S_k. `RunAFAModeB` means using CryptoMiniSAT to solve for all possible solutions. `CalcSolutionCount` represents counting the solutions of the secret key from the output file of CryptoMiniSAT. From Algorithm 3, we can see that when $\eta(\kappa) \ge \tau$ and $\kappa > 0$, $\phi(K)$ can be roughly estimated as $\kappa + log_2\eta(\kappa)$. If $\kappa = 0$ and $\eta(\kappa) \le \tau$, the accurate value of $\phi(K)$ is $log_2\eta(\kappa)$.

4 Application to LBlock: Fault Injection to Encryption (Scenario 1)

4.1 Fault Model

In this scenario, the fault f is injected into X_{rc+1} in LBlock encryption which is marked with a red double box in Algorithm 4. The fault model can be described as $\mathsf{F}(X_r^{\text{en}}, \lambda, w, t, f)$. More specifically, a fault is injected into the left 32-bit register of the encryption ($\lambda = 32$), whose value f is unknown. We consider three cases for the fault width ($w = 1, 4, 8$) and two cases for the location (t is known or unknown).

Algorithm 4: Fault Injection to Encryption

```
r_max = 32 ;
P = X_1 ‖ X_0 ;
for rc = 0; rc < r_max; rc++ do
    X_{rc+2} = F( f ⇝ X_{rc+1} , K_{rc+1}) + (X_rc <<< 8);
end
C = X_32 ‖ X_33 ;
```

4.2 AFA Procedure

The attack is described in Algorithm 5. Both Mode A and Mode B of CryptoMiniSAT are considered. We define an *instance* as one run of our algorithm under one specific fault model. In one *instance*, the algorithm may be repeated many times, each of which requires one pair of plaintext and ciphertext, and one fault injection. We define N as the number of fault injections. For these N fault injections, the fault model F is the same. As for the inputs of Algorithm 5, b_t is a flag to indicate whether the location t is known or not. r is the specific round of X_r^{en}. If the fault is induced into a deeper round, the value of r is smaller. If the solver is under Mode A, the output is the solving time t_{sol} if it is successful. Otherwise the algorithm stops at a time out t_{out}. We define a success rate SR for extracting the master key, which is the number of instances with a successful solving within t_{out} over the number of all instances. If it is under Mode B, the output is the remaining key entropy $\phi(K)$.

In Algorithm 5, $\mathbb{P}$ and $\mathbb{K}$ denote the sets for plaintexts and round keys respectively. `KS` and `Enc` denote the key scheduling and encryption function respectively. `RandomPT` generates one or more random plaintexts. `InjectFault` induces one fault. A function in Algorithm 5 will generate an equation set if its name is prefixed with `Gen` and suffixed with `ES`. The attack can be described as follows. The adversary A generates

N pairs of plaintext/ciphertext and starts constructing equations. First, he builds the equations for key scheduling (GenKSRdES in Line #1). For each P_i, he will build the equation set for the correct encryption (R_r to R_{32}) using C_i (GenEnRdES in Line #2). For each injection, he needs to build the equation set for the faulty encryption (R_r to R_{32}) using C_i^* (in Line #3) together with the one for the fault itself (GenFaultyES in Line #4). Besides that, A has to generate the equation set for a full round encryption (in Line #5). The equation set based on a pair of (P_v, C_v) in Line #6 is for the verification purpose under Mode A. Finally, these combined equation sets are fed into the solver for key recovery (RunAFA in Line #7).

Algorithm 5: The AFA Procedure of Scenario 1

input : N, r, w, b_t
output: t_{sol} in Mode A, $\phi(K)$ in Mode B

1 RandomPT($\mathbb{P}$) ;
2 $\mathbb{K}$=KS (K, L) ;
3 **for** $rc = 1$; $rc < r_{max}$; rc++ **do**
4 | GenKSRdES (rc, K_{rc+1}) ; // #1
5 **end**
6 **for** $i = 0$; $i < N$; i++ **do**
7 | C_i=Enc (P_i, K) ;
8 | **for** $rc = r - 1$; $rc < r_{max}$; rc++ **do**
9 | | GenEnRdES ($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #2
10 | **end**
11 | GenInputES (C_i) ;
12 | C_i^*=InjectFault(Enc(P_i, K), X_r);
13 | **for** $rc = r - 1$; $rc < r_{max}$; rc++ **do**
14 | | GenEnRdES ($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #3
15 | **end**
16 | GenInputES (C_i^*) ;
17 | GenFaultyES ($f = X_r + X_r^*$) ; // #4
18 **end**
19 RandomPT(P_v) ;
20 C_v=Enc (P_v, K) ;
21 **for** $rc = 0$; $rc < r_{max}$; rc++ **do**
22 | GenEnRdES ($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #5
23 **end**
24 GenInputES (P_v, C_v) ; // #6
25 ($T_{sol}, \phi(K)$) = RunAFA() ; // #7

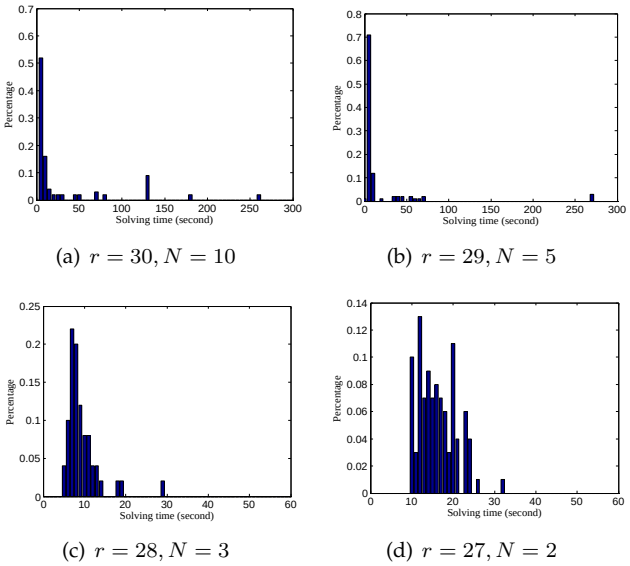
4.3 Case Study 1: Bit-based Fault Model

Under bit-based fault model, we consider different fault positions and known/unknown locations.

4.3.1 The location t is unknown

For a specific state X_r^{en}, we decrease r from 30 to 24. For each r, 100 instances of AFA are conducted under Mode A. For each instance, there are N fault injections. The statistics of different values of (r, N) are shown in Figure 3. The horizontal axis is the solving time in seconds. The vertical axis is the percentage.

In Figure 3, the statistics seem to follow an exponential distribution. N can be reduced when r is smaller, which means that an injection to a deeper round could reduce the number of faults that are required. When $(r, N) = (30, 10)$ or $(29, 5)$, the 80-bit master key of LBlock can be recovered within five minutes, $SR = 100\%$. If $(r, N) = (28, 3)$ or $(27, 2)$, it can be extracted in one minute, $SR = 100\%$.



(a) $r = 30, N = 10$ (b) $r = 29, N = 5$

(c) $r = 28, N = 3$ (d) $r = 27, N = 2$

Figure 3: Distribution of solving time under bit-based fault model, t is unknown (Mode A)

Note that the single-bit fault model in [20] can be converted to our fault model in this paper. The work in [20] assumed that a single-bit fault is randomly injected into the internal state at the end of the $(r-1)$-th round. This is equivalent to our bit-based fault model in the r-th round, where single-bit fault is randomly injected into the left input register of the r-th round. The comparison with [20] is shown in Table 2. With our framework, we first verify the result in previous work for specific rounds. In contrast, the efficiency and effectiveness of our work are demonstrated when the fault is injected into the same round. Our attack requires only a few injections. For example, two injections are enough for our AFA in R_{27}, while about 8 injections or more are required for most cases in [20].

Table 2: The number of injections in comparison with previous work under random bit-based fault model

r	DFA in [20]	AFA in this paper
30	24	10
29	24	5
28	24	3
27	12	3
26	8	2
25	24	5

4.3.2 The location t is known

If t is known, we first conduct 100 AFA instances for each r under Mode A, $t_{out} = 3600$ seconds. The results in Table 3 show that N becomes smaller compared to that for the same r when t is unknown. For example, $(r, N) = (29, 5)$ in Figure 3 while $(r, N) = (29, 4)$ in Table 3. Moreover, fault injections in deeper rounds can help retrieve the key. For instance, when r is decreased from 28 to 27, N can also be reduced from 3 to 2.

Table 3: Bit-based fault model, t is known (Mode A)

r	N	t_{sol} (seconds)	Success Rate
30	10	7	100%
29	4	15	100%
28	3	6	100%
27	2	10	100%
26	2	15	100%
26	1	1997	92%
25	2	221	91%
24	5	321	85%
23	50	654	65%

In particular, when a single bit fault is injected into the left register in R_{26}, it might be possible to recover the master key. In this special case, we first try to solve for the secret key directly under Mode A. When $t_{out} = 2$ hours, SR is only 18% for most instances, which indicates that it is difficult for CryptoMiniSAT to find the solution. To overcome this, we guess an 8-bit value of the master key and feed this value into the solver. The attack stops when the solver finds out a satisfiable solution for one key guess. Since there are 256 possible values, we can conduct at least 1, at most 256 (on average 128) guesses for each instance. When more guessed key variables are fed into the solver, CryptoMiniSAT can either find a satisfiable solution or output "unsatisfiable". The statistics of the solving time of 100 AFA instances are listed in Figure 4. The master key can be recovered within 1997 seconds on average and $SR = 92\%$ when $t_{out} = 2$ hours. *To the best of our knowledge, this is the first time LBlock has been attacked with only one injection under bit-based fault model.*

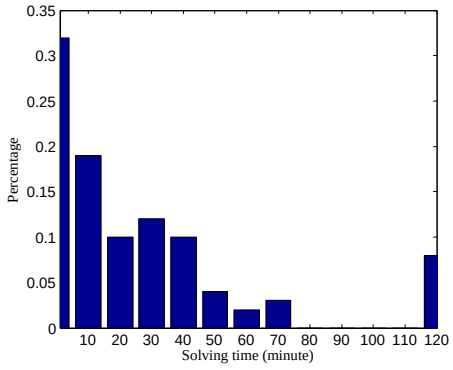


Figure 4: Distribution of solving time with one injection to R_{26} under bit-based fault model (Mode A)

To interpret the results in Table 4, we evaluate $\phi(K)$ for one fault injection ($N = 1$) under Mode B. Let ψ denote the number of the faulty nibbles in the ciphertext for one injection. Let $\bar{\psi}$ denote the average of ψ where 10000 random instances are collected. Results of ψ, $\bar{\psi}$ and $\phi(K)$ are listed in Table 4.

From Table 4, we can see that when $28 \leq r \leq 30$, only a few nibbles in the ciphertext become faulty. Since $r = 26$, all 16 nibbles in the ciphertext are faulty. When $(r, \psi) = (26, 16)$, our best result of AFA shows that $\phi(K)$ can be reduced to 17.3. Note that in Table 4, $N = 1$. When $(r, N) = (30, 1)$, $\phi(K)$ can be reduced to about 70.2, which means that 9.8 key bits can be recovered with a single injection. Then, when $(r, N) = (30, 10)$, $\phi(K)$ can be reduced to a smaller value. This can also explain why CryptoMiniSAT can output the correct solution within a few seconds for $(r, N) = (30, 10)$ in Table 3. In particular, when $(r, N) = (26, 1)$, $\phi(K)$ can be reduced to about 17.3 in Table 4. It explains why CryptoMiniSAT can find the secret key within affordable time under Mode A in Table 3.

Table 4: ψ, $\bar{\psi}$ and $\phi(K)$ under bit-based fault model

r	ψ	$\bar{\psi}$	Best $\phi(K)$
30	5	5	70.2
29	8	8	61.6
28	$11 \leq \psi \leq 12$	11.81	50.4
27	$12 \leq \psi \leq 15$	14.57	32.6
26	$10 \leq \psi \leq 16$	15.21	17.3
25	$9 \leq \psi \leq 16$	14.99	18.5
24	$9 \leq \psi \leq 16$	14.99	≤ 24
23	$8 \leq \psi \leq 16$	15.00	≤ 40

4.4 Case Study 2: Nibble-based Fault Model

In [21], [22], the adversary has to build the distinguishers manually and deduce the fault position. Specific algorithms must be customized for each fault position. We conduct AFA under nibble-based fault model as in [21], [22]. However, with our framework, the solver can automatically deduce the fault position and solve for the key. The workload for customizations can be saved.

We extend the faults into deeper rounds and calculate $\phi(K)$ for a given amount of fault injections. The comparison with previous work [21], [22] under Mode B is shown in Table 5. Under the same fault model, our AFA can use less injections. For example, when $r = 30$, we can reduce N from 7 to 5 as compared to [21].

Table 5: Comparison with previous work under nibble-based fault model

r	[21]		[22]		this paper	
	N	$\phi(K)$	N	$\phi(K)$	N	$\phi(K)$
30	7	30	-	-	5	23
29	5	25	-	-	5	13
28	-	-	-	-	3	7.56
27	-	-	-	-	2	4.6
26	-	-	-	-	2	0.1
25	-	-	24	0	3	0
24	-	-	32	0	5	0

Our AFA can further reduce $\phi(K)$. In [21], $\phi(K)$ is 30 and 25 when $(r, N) = (30, 7)$ and $(29, 5)$ respectively. As for $(r, N) = (29, 5)$, $\phi(K)$ is 13 in our AFA, compared to 25 in [21]. The estimation on $\phi(K)$ in [21] might not be accurate. This is because the manual analysis may miss some faulty states in the propagation path, while the solver fully utilizes all the faults along all paths. Each faulty state can contribute his own entropy to reducing $\phi(K)$. As a result, our AFA can achieve better efficiency.

Significant enhancements are achieved when the injections are to R_{24} or R_{25}. In Table 5, our attack requires only 3 and 5 injections, compared with 24 injections for R_{25} and 32 injections for R_{24} in [22], respectively.

4.5 Case Study 3: Byte-based Fault Model

Previous fault attacks on LBlock [21], [22], [20] are mainly under bit-based or nibble-based model. As aforementioned in Section 3, LBlock usually adopts the size-optimized or speed-optimized implementation on 8-bit microcontrollers. For speed-optimized implementation, the fault width is one byte. Under byte-based fault model, the fault propagation becomes more complicated.

We are concentrating on challenging AFA on LBlock under byte-based fault model. We implement the speed-optimized version of LBlock. One single byte fault is injected into the input of the big S-Box. The results under Mode B are listed in Table 6 where our AFA can still reduce $\phi(K)$ to a smaller value. For example, when $(r, N) = (26, 2)$, $\phi(K)$ can be further reduced to 0.

Table 6: AFA under random byte-based fault model

r	t is unknown		t is known	
	N	$\phi(K)$	N	$\phi(K)$
30	5	20.8	5	16
29	3	20.4	3	13.5
28	3	15.6	2	12.4
27	-	-	2	2.3
26	-	-	2	0
25	-	-	3	0

4.6 Comparisons with Previous Work

Compared with previous fault attacks on LBlock [20], [21], [22], [23], our work demonstrates that the data complexity of previous work is not optimal and AFA can work at much deeper rounds. Meanwhile, under different fault models, AFA can automatically evaluate the remaining key search space. For the first time, only one fault injection is required to recover the master key. To the best of our knowledge, this is the best result for fault attacks on LBlock in terms of data complexity.

5 Application to LBlock: Fault Injection to Key Scheduling (Scenario 2)

5.1 Fault Model

In this scenario, a state register for round keys is altered due to the injected fault. The fault will be propagated to the remaining rounds of key scheduling. This case is equivalent to injecting multiple faults simultaneously into multiple rounds. The manual analysis is difficult due to the complexity. In contrast, the automatic analysis by CryptoMiniSAT is expected to be much more efficient. This is because the more equations that are generated, the more entropies are utilized in the same problem solving.

In this model $\mathsf{F}(X_r^{\mathrm{ks}}, \lambda, w, t, f)$, a fault is injected into the left 32-bit of the 80-bit key register L in the r-th round key scheduling ($\lambda = 32$), as shown in Algorithm 6. The round key $K_r, K_{r+1}, \ldots, K_{32}$ are faulty. We consider three cases for the fault width ($w = 1, 4, 8$) and the location t is known.

Algorithm 6: Fault Injection to Key Scheduling

$r_{max} = 32$;
$L = K$;
$K_1 = \text{Left32}(L)$;
for $rc = 1$; $rc < r_{max}$; rc++ **do**
  $f \rightsquigarrow L$ <<< 29 ;
  $[l_{79}\|l_{78}\|l_{77}\|l_{76}] = s_9[l_{79}\|l_{78}\|l_{77}\|l_{76}]$;
  $[l_{75}\|l_{74}\|l_{73}\|l_{72}] = s_8[l_{75}\|l_{74}\|l_{73}\|l_{72}]$;
  $[k_{50}\|k_{49}\|k_{48}\|k_{47}\|k_{46}] \oplus [rc]$;
  $K_{rc+1} = \text{Left32}(L)$;
end

5.2 AFA Procedure

The detailed procedure is depicted by Algorithm 7 where there are only two slight differences with Algorithm 5. In Line #3, the adversary has to build the equation set for the faulty key scheduling (R_r to R_{31}). In Line #4, he has to build the equation set for the faulty encryption (R_r to R_{32}) using the faulty round keys.

Algorithm 7: The AFA Procedure of Scenario 2

input : N, r, w, b_t
output: t_{sol} in Mode A, $\phi(K)$ in Mode B

RandomPT($\mathbb{P}$) ;
$\mathbb{K}$=KS (K, L) ;
for $rc = 1$; $rc < r_{max}$; rc++ **do**
  GenKSRdES(rc, K_{rc+1}) ; // #1
end
for $i = 0$; $i < N$; i++ **do**
  C_i=Enc(P_i, K) ;
  for $rc = r - 1$; $rc < r_{max}$; rc++ **do**
    GenEnRdES($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #2
  end
  GenInputES(C_i) ;
  $\mathbb{K}^*$=InjectFault(KS(K, L)) ;
  for $rc = r$; $rc < r_{max}$; rc++ **do**
    GenKSRdES(rc, K^*_{rc+1}) ; // #3
  end
  C_i^*=Enc($P_i, \mathbb{K}^*$) ;
  for $rc = r$; $rc < r_{max}$; rc++ **do**
    GenEnRdES($X_{rc+1}, X_{rc}, K^*_{rc+1}$) ; // #4
  end
  GenInputES(C_i^*) ;
  GenFaultyES($f = L + L^*$) ; // #5
end
RandomPT(P_v) ;
C_v=Enc(P_v, K) ;
for $rc = 0$; $rc < r_{max}$; rc++ **do**
  GenEnRdES($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #6
end
GenInputES(P_v, C_v) ; // #7
$(T_{sol}, \phi(K))$ = RunAFA() ; // #8

5.3 Case Study 1: Bit-based Fault Model

First, we evaluate $\phi(K)$ for different r under bit-based fault model under Mode B. ψ, and $\bar{\psi}$ are collected

from 10000 instances with single fault injection. $\phi(K)$ is calculated from 100 full AFA attacks. Results of ψ, $\bar{\psi}$ and $\phi(K)$ are listed in Table 7.

Table 7: $\bar{\psi}$ and $\phi(K)$ under bit-based fault model

r	ψ	$\bar{\psi}$	Best $\phi(K)$
30	$2 \leq \psi \leq 3$	2.10	74
29	$3 \leq \psi \leq 4$	3.40	71
28	$4 \leq \psi \leq 8$	6.17	62
27	$5 \leq \psi \leq 12$	9.52	42
26	$8 \leq \psi \leq 15$	12.89	28
25	$9 \leq \psi \leq 16$	14.83	20
24	$9 \leq \psi \leq 16$	15.09	16
23	$10 \leq \psi \leq 16$	15.00	≤ 30
22	$10 \leq \psi \leq 16$	15.00	≤ 42

From Table 7, we can see that when $27 \leq r \leq 30$, only a few nibbles in the ciphertext become faulty. Since $r = 25$, all 16 nibbles in the ciphertext are faulty. When $(r, \psi) = (24, 16)$, our best result of AFA shows that $\phi(K)$ can be reduced to 16.

It is interesting to see that if $r \geq 23$, $\phi(K)$ increases when r decreases. For instance, $\phi(K)$ changes from 16 to less than 30 if the injection changes from R_{24} to R_{23} in key scheduling. Meanwhile, $\bar{\psi}$ is approximately 15 for $r = 23$, which is even slightly smaller than $\bar{\psi} = 15.09$ for $r = 24$. The reason behind is the overlap of the faults in the last few rounds.

Note that Table 7 can be used to determine the optimal round position for the injection and estimate the total number of injections that is required. From Table 7, we can deduce that it is into R_{24} where we should inject a bit-based fault in order to minimize $\phi(K)$.

In our attack, when $r = 24, 25, 26$, two single-bit fault injections ($N = 2$) can reduce $\phi(K)$ to 0 under Mode B. In particular, we also conducted AFA with only one single-bit fault injection under Mode A for $r = 24, 25$. As in Section 4.3, we guess an 8-bit value of the master key and feed this value into the solver. The results show that the full key can be recovered within two hours where SR is about 85%.

5.4 Case Study 2: Nibble-based Fault Model

The results under nibble-based fault model are shown in Table 8, where 10000 random instances are collected. We can see that the fault propagation is faster under this model than under bit-based model. For example, for the same $r = 27$, $\bar{\psi} = 10.09$ in Table 8 while $\bar{\psi} = 9.52$ in Table 7. Note that $\phi(K) \leq 40$ when $23 \leq r \leq 27$. Our best results show that two fault injections can recover the master key of LBlock when $24 \leq r \leq 26$. Similarly, it is in R_{25} where we should inject a nibble-based fault in order to minimize $\phi(K)$, which could be used as an empirical parameter to guide the physical injections if possible.

5.5 Case Study 3: Byte-based Fault Model

The results under byte-based fault model are shown in Table 9. We can observe that the fault propagation under byte-based model is very fast. $\bar{\psi}$ is close to 4 when $r = 30$. $\phi(K) \leq 40$ when $23 \leq r \leq 28$. Our best results show that when $24 \leq r \leq 28$, two fault injections can recover the full key of LBlock.

Table 8: $\bar{\psi}$ and $\phi(K)$ under nibble-based fault model

r	ψ	$\bar{\psi}$	Best $\phi(K)$
30	$2 \leq \psi \leq 3$	2.11	75
29	$3 \leq \psi \leq 5$	3.60	66
28	$3 \leq \psi \leq 10$	6.70	62
27	$4 \leq \psi \leq 13$	10.09	38
26	$5 \leq \psi \leq 15$	13.24	32
25	$8 \leq \psi \leq 16$	14.92	20
24	$10 \leq \psi \leq 16$	15.06	≤ 32
23	$10 \leq \psi \leq 16$	15.00	≤ 40
22	$10 \leq \psi \leq 16$	15.00	≤ 60

Table 9: $\phi(K)$ and ψ under byte-based fault model

r	ψ	$\bar{\psi}$	Best $\phi(K)$
30	$2 \leq \psi \leq 5$	3.98	75
29	$3 \leq \psi \leq 9$	3.65	60
28	$4 \leq \psi \leq 14$	10.45	39
27	$6 \leq \psi \leq 15$	13.32	35
26	$8 \leq \psi \leq 16$	14.92	28
25	$8 \leq \psi \leq 16$	15.02	26
24	$10 \leq \psi \leq 16$	15.01	16
23	$10 \leq \psi \leq 16$	15.00	≤ 45
22	$10 \leq \psi \leq 16$	15.00	≤ 65

6 Application to LBlock: Fault Injection for Round Modification (Scenario 3)

6.1 Fault Model

During a typical implementation, *round number*, denoted by r_{max}, is the total number of rounds to be executed. *round counter*, denoted by rc, is a variable that specifies which round it is executing. In this section, we evaluate the security of LBlock against *round modification attack* (RMA). RMA can induce the misbehavior of round operations by fault injections. A fault could be injected either into r_{max} or rc. The new values are denoted by r'_{max} or rc'. The change in the execution of LBlock can facilitate subsequent cryptanalysis.

In LBlock, there are 31 rounds in key scheduling. The round keys generated from key scheduling will be further utilized in the 32-round encryption. Two round counters are actually used for key scheduling and encryption. $r_{max} = 32$ before the fault injection. Due to page limitation, we mainly discuss the scenario when a fault is injected to modify the round during encryptions.

In this model $\mathsf{F}(X^{\text{en}}_{rc}, \lambda, w, t, f)$, a fault is injected into X^{en}_{rc} in encryption. As in previous RMA work [11], [12], we assume that both the fault value f and the fault location t are known. $\lambda = w = 8$. We consider two cases for the fault position, as shown in Algorithm 8.

6.2 AFA Procedure

The detailed procedure is depicted by Algorithm 9, where there are only three slight differences with Algorithm 5. Line #3 and Line #4 show how the adversary

Algorithm 8: Fault Injection to r_{max} or rc

1 $P = X_1 \| X_0$;
2 $f \rightsquigarrow r_{max} = 32$;
3 **for** $f \rightsquigarrow rc = 0$ *to* $r_{max} - 1$ **do**
4 $\quad X_{rc+2} = F(X_{rc+1}, K_{rc+1}) + (X_{rc} <<< 8)$;
5 **end**
6 $C = X_{32} \| X_{33}$;

can build the equation set for the faulty encryption (R_r to R_{31}) if the fault is injected into r_{max} or rc (determined by b) respectively. Line #4 in Algorithm 5 is discarded here.

Algorithm 9: The AFA Procedure of Scenario 3

input : N, b, r, rc', r'_{max}
output: t_{sol} in Mode A, $\phi(K)$ in Mode B

1 RandomPT($\mathbb{P}$) ;
2 $\mathbb{K}$=KS (K, L) ;
3 **for** $rc = 1$; $rc < r_{max}$; rc++ **do**
4 $\quad$ GenKSRdES(rc, K_{rc+1}) ; // #1
5 **end**
6 **for** $i = 0$; $i < N$; i++ **do**
7 $\quad C_i$=Enc ($P_i, \mathbb{K}$) ;
8 $\quad$ **for** $rc = 0$; $rc < r_{max}$; rc++ **do**
9 $\quad\quad$ GenEnRdES($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #2
10 $\quad$ **end**
11 $\quad$ GenInputES(C_i) ;
12 $\quad$ **switch** b **do**
13 $\quad\quad$ **case** *0*
14 $\quad\quad\quad C_i^*$=InjectFault(r'_{max}, Enc($P_i, \mathbb{K}$)) // #3
$\quad\quad\quad$ **for** $rc = 0$; $rc < r'_{max}$; rc++ **do**
15 $\quad\quad\quad\quad$ GenEnRdES($X_{rc+1}, X_{rc}, K_{rc+1}$)
16 $\quad\quad\quad$ **end**
17 $\quad\quad\quad$ GenInputES(C_i^*) ;
18 $\quad\quad$ **end**
19 $\quad\quad$ **case** *1*
20 $\quad\quad\quad C_i^*$=InjectFault(r,rc',Enc($P_i, \mathbb{K}$)) // #4
$\quad\quad\quad b_{tag} = 0$;
21 $\quad\quad\quad$ **for** $rc = 0$; $rc < r_{max}$; rc++ **do**
22 $\quad\quad\quad\quad$ GenEnRdES($X_{rc+1}, X_{rc}, K_{rc+1}$);
23 $\quad\quad\quad\quad$ **if** $rc = r - 2$ *and* $b_{tag} = 0$ **then**
24 $\quad\quad\quad\quad\quad b_{tag}++$;
25 $\quad\quad\quad\quad\quad rc = rc'$;
26 $\quad\quad\quad\quad\quad$ **if** $rc > 31$ **then**
27 $\quad\quad\quad\quad\quad\quad$ break;
28 $\quad\quad\quad\quad\quad$ **end**
29 $\quad\quad\quad\quad$ **end**
30 $\quad\quad\quad$ **end**
31 $\quad\quad\quad$ GenInputES(C_i^*) ;
32 $\quad\quad$ **end**
33 $\quad$ **endsw**
34 **end**
35 RandomPT(P_v) ;
36 C_v=Enc ($P_v, \mathbb{K}$) ;
37 **for** $rc = 0$; $rc < r_{max}$; rc++ **do**
38 $\quad$ GenEnRdES($X_{rc+1}, X_{rc}, K_{rc+1}$) ; // #5
39 **end**
40 GenInputES(P_v,C_v) ; // #6
41 $(T_{sol}, \phi(K))$ = RunAFA() ; // #7

6.3 Case Study 1: Injecting Faults to Modify r_{max}

In this case, a fault is injected into r_{max} in Line 2 of Algorithm 8. r_{max} could be accessed at the beginning of each instance where the fault may cause an increase or decrease in the total number of rounds.

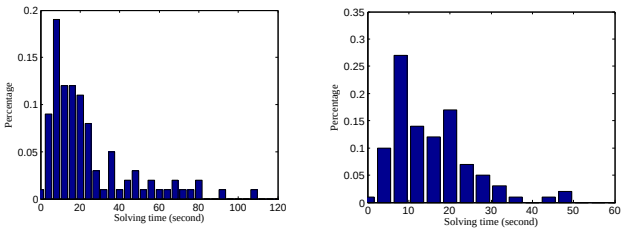
6.3.1 *Case 1:* $r'_{max} \geq 32$

In this case, LBlock will proceed ($r'_{max} - 32$) additional rounds after the normal encryption. These extra rounds use invalid values of round keys (for instance, four 0xcc bytes observed from physical experiments) which are known to the adversary. This case does not provide the adversary with any useful information.

6.3.2 *Case 2:* $r'_{max} < 32$

In this case, LBlock will only proceed with the first r'_{max} rounds and skip the remaining ($32 - r'_{max}$) rounds. As for the adversary, the key recovery is a reduced ($32 - r'_{max}$) round cryptanalysis. We are interested in the cases $r'_{max} = 28$ or 29 which are difficult for previous work.

We first run 100 random AFA instances under Mode A. Time statistics for $r'_{max} = 28$ and $r'_{max} = 29$ are shown in Figure 5. The solver can output the correct solution within one minute for $r'_{max} = 28$ and two minutes for $r'_{max} = 29$. Under Mode B, we also run 100 random AFA instances and calculate $\phi(K)$ for $r'_{max} = 28$ and 29. The results show that $\phi(K)$ can be reduced to $16 \sim 17$ which could be done with a brute force. This can also explain why the solver can recover the master key within a limited time under Mode A.



(a) $r'_{max} = 28$ (b) $r'_{max} = 29$

Figure 5: Distribution of solving time for AFA when modifying r_{max}

Meanwhile, we conduct AFA on LBlock for $r'_{max} = 3$ or 4. Under unknown plaintext scenario, since the key recovery is equivalent to analyzing the ($32 - r'_{max}$) round LBlock, it is difficult for the solver to recover the secret key within limited time. However, under known plaintext/ciphertext scenario, it can be converted into the algebraic analysis of a reduced r'_{max} round LBlock. Under Mode A, the solver can always solve the problem within one minute.

6.4 Case Study 2: Injecting Faults to Modify rc

In this case, a fault is injected to rc in Line 3 of Algorithm 8 at the beginning of R_r, the r-th round. Depending on the instant value of rc and the faulty value rc', various changes may occur during encryption, such as adding, reducing or even repetitively executing several rounds.

6.4.1 Case 1: $rc' < rc < r_{max}$

In this case, $(rc - rc')$ intermediate encryption rounds can be repeated. We illustrate a simple case where $rc = 30$ and $rc' = 29$. The sequence of rounds during encryption is shown as below.

$$R_1, R_2, \cdots, R_{29}, R_{30}, R_{30}, R_{31}, R_{32} \tag{13}$$

We can see that R_{30} is repeated twice. During the key recovery, two types of equation sets are built: those for $R_1, \cdots, R_{29}, R_{30}, R_{31}, R_{32}$ with a correct ciphertext, and those for $R_1, \cdots, R_{29}, R_{30}, R_{30}, R_{31}, R_{32}$ with a faulty ciphertext.

Under known ciphertext scenario, we conduct 100 AFA instances. The results show that under Mode A, the solver can finish in two minutes with 100% success rate; under Mode B, $\phi(K)$ can be reduced to $16 \sim 17$.

6.4.2 Case 2: $rc < rc' < r_{max}$

In this case, $(rc' - rc)$ intermediate encryption rounds can be skipped. We investigate the case when $rc = 29$ and $rc' = 31$. The sequence of those rounds during encryption is shown as below. R_{30} and R_{31} are skipped. The total number of rounds actually executed is 30.

$$R_1, R_2, \cdots, R_{29}, R_{32} \tag{14}$$

Then the key recovery is converted into the algebraic analysis with two equation sets: one for $R_1, R_2, \cdots, R_{29}, R_{30}, R_{31}, R_{32}$ with a correct ciphertext, and one for $R_1, R_2, \cdots, R_{29}, R_{32}$ with a faulty ciphertext. Results achieved are similar to the ones in Case 1. One fault injection is enough to recover the master key of LBlock within two minutes.

6.4.3 Case 3: $rc < r_{max} < rc'$

In this case, $(33 - rc)$ intermediate encryption rounds can be skipped. One more example can be given for $rc = 30$ and $rc' = 35$. The sequence is $R_1, R_2, \cdots, R_{29}$. Note that R_{30}, R_{31}, R_{32} are skipped. This case is equivalent to our Case Study 1 when $r_{max} = 29$. The result is similar to Case 1. One fault injection is enough to recover the full key within one minute.

It should be noted that AFA can also be used to recover the master key when a fault is injected to modify the round during key scheduling. Since only the number of rounds in key scheduling has been modified and that in the encryption is always 32, the equation sets to be built are slightly different from those in this section. Our experiment results show that, if a single fault could be injected into either r_{max} or rc in key scheduling of LBlock, $\phi(K)$ can also be reduced to $16 \sim 17$.

7 Extensions to Other Block Ciphers

The work on LBlock demonstrated the generic feature of our framework on lightweight block ciphers which typically have simple structures but many iterative rounds. That is why we chose lightweight block ciphers such as LBlock as an appropriate starting point to check how to represent the internal structure with equations and how AFA is affected by the number of rounds or other factors such as fault models. In fact the work in this paper can be extended to other well known block ciphers. Some new and interesting results are achieved.

7.1 Application to DES

DES is a block cipher that uses a 56-bit master key and operates on 64-bit blocks. It has 16 rounds preceded and followed by two bit-permutation layers IP and IP^{-1}. The round transformation F follows a Feistel scheme. The 64-bit block is split into two 32-bit parts L and R. F is defined as $F_{K_r} = (R, L \oplus f_{K_r}(R))$. The function f first applies an expansion layer E that expands the 32 input bits into 48 output bits by duplicating 16 of them. The 48-bit round key K_r is then introduced by bitwise addition. Afterward the block is split into eight 6-bit blocks, each entering into a different S-Box S_i with a 4-bit output. Finally, the 32 bits from the eight S-Boxes are permuted by a bit-permutation P which yields the 32-bit block.

For simplicity reasons, we assume that a single-bit fault is injected into the left part of the DES state at the end of one round, as in the previous work [3], [25], [8]. The fault model can be described as $\mathsf{F}(X_r^{\text{en}}, \lambda = 32, w = 1, t, f)$. In practice, the single-bit fault can be injected by high precision techniques such as lasers when both the location t and the value f are known. We conduct both AFA on DES under Mode A and Mode B.

Under Mode A, for each r in the range of [1,16], we fix the number of fault injection $N = 1$ and randomly choose the fault location t in the range of [0,31]. When $1 \leq r \leq 10$, or $13 \leq r \leq 16$, 10 simulations were conducted for each round and the solver could not output the solution within 24 hours. When $(r, N) = (12, 1)$, the solver can recover the secret key within one hour. When $(r, N) = (11, 1)$, the solver can succeed in one minute.

When $(r, N) = (11, 1)$ or $(r, N) = (12, 1)$, we observe a very interesting experimental result. The solving time is different when t varies. The statistical results are shown in Figure 6 and 7 where 640 AFAs are conducted and each value of t is tested for 20 times. When $t \in \{0, 3, 4, 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31\}$, the solving time is smaller. We guess that this might be caused by different entropy of the remaining key search space and then conduct the attack under Mode B. The statistical results are shown in Figure 8 and 9 which are consistent to those in Figure 6 and 7. When $t \in \{0, 3, 4, 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31\}$, $\phi(K)$ is smaller than in other locations. When $(r, N) = (12, 1)$, $\phi(K) \approx 5$. When $(r, N) = (11, 1)$, $\phi(K) \approx 0$. The reason behind lies on the fact that, since the expansion layer E duplicates some bits, the single bit fault on those locations propagates to two S-Boxes instead of one.

Compared with the first DFA work on the last three rounds of DES [3], [25] and the middle rounds of

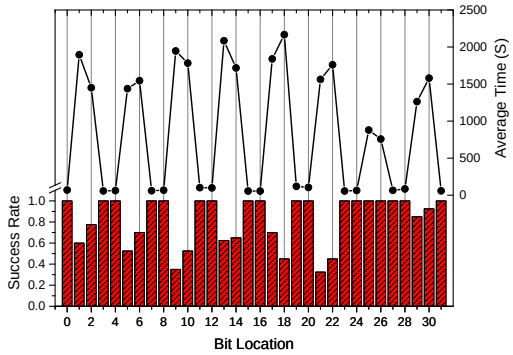


Figure 6: The average solving time and the success rate on different bit locations for DES,$(r, N) = (11, 1)$

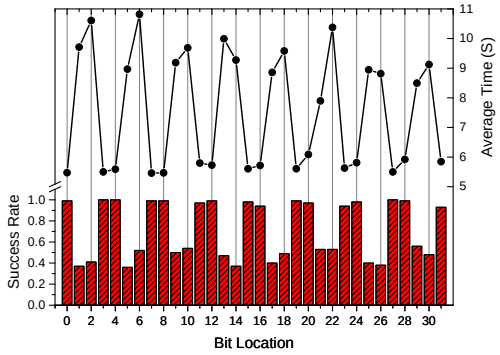


Figure 7: The average solving time and the success rate on different bit locations for DES,$(r, N) = (12, 1)$

DES [25], our AFA demands less data complexity and only one fault injection is required. Compared with the first AFA work on DES [4], the time complexity of our work is optimal and our best results show that the master key of DES can be recovered within a few seconds. Contrary to the recent AFA on DES [8], we evaluate the remaining key entropy of faults attacks on DES and find out that the key recovery efficiency is not the same for the various bits of fault locations.

7.2 Application to PRESENT

PRESENT is a 31-round block cipher with an SPN structure. The block size is 64 bits. Each round consists of three major operations. The first one is addRoundKey (AK) where the 64-bit input is XORed with the round key. The second one is sBoxlayer (SL) where 16 identical 4×4 S-Boxes are used in parallel. The third operation is the pLayer (PL) where the 64-bit input is permuted according to a table P. PRESENT has two versions. In this paper, we mainly focus on PRESENT-80. Its key scheduling uses simple bit rotation, S-Box lookup and round-counter XOR operations. More details can be found in [14].

Extensive AFAs are conducted on PRESENT under different fault models. We assume that the input of AK layer in the 29th or 28th round is injected with faults.

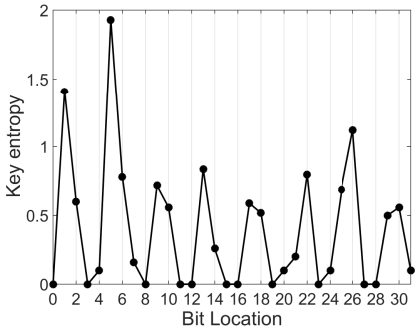


Figure 8: $\phi(K)$ with different w for DES, $(r, N) = (11, 1)$

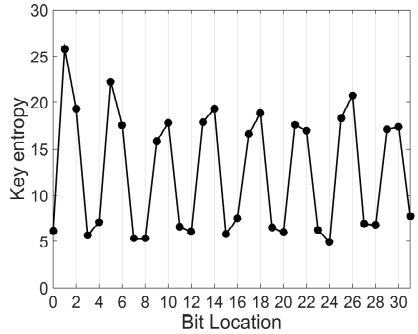


Figure 9: $\phi(K)$ with different w for DES, $(r, N) = (12, 1)$

The fault width w can be 1, 4, 8, 16, 32. 100 instances are performed for each w. The statistical results of AFA on PRESENT under Mode B are shown in Figure 10 and 11. Considering the fault position, the injection in the 28th round is more efficient than in the 29th round. The average value of $\phi(K)$ is much smaller. Considering the fault width, the single-bit fault model is the optimal one and the word-based fault model is the worst. Our best results show that when $w = 1$ and $(r, N) = (28, 1)$, for some attack instances, one fault injection can reduce $\phi(K)$ to less than 30 with 35% probabilities. For most instances, two injections can recover K within three minutes.

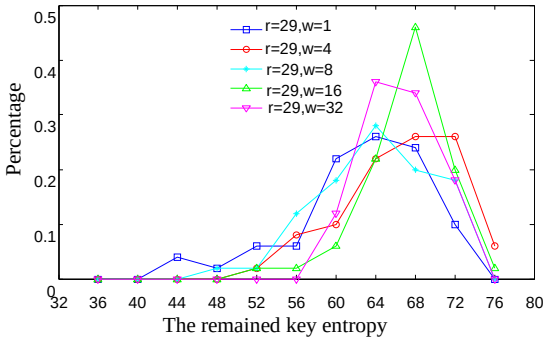


Figure 10: $\phi(K)$ with different t for PRESENT, $(r, N) = (29, 1)$

Previous fault attacks on PRESENT [26], [27] are DFAs

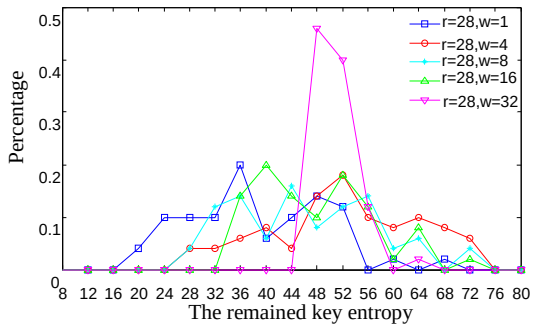


Figure 11: $\phi(K)$ with different t for PRESENT, $(r, N) = (28, 1)$

to the 29th round ($r = 29$) under nibble-based fault model. Their results show that 8 fault injections can reduce $\phi(K)$ to 14.7. As to faults injected into a deep round, e.g., $r = 28$, the fault propagation paths get overlapped and the techniques in [26], [27] are difficult to work. The AFA in this paper is very generic and the solver can automatically analyze all the faults along the propagation path. Only one to two injections are required for key recovery.

7.3 Application to Twofish

Twofish is a 128-bit Feistel structure block cipher, which was one of the five AES finalists [15]. In this paper we only consider Twofish with a key length of 128 bits. The plaintext is split into four 32-bit words and XORed with four words of the whitening key (one rotated by 1 bit towards the left) and followed by 16 rounds. In each round, two most significant input words (one rotated by 8 bits towards the left) are fed into the F function. F has a g function followed by Pseudo-Hadamard Transform (PHT) and key word addition (modulo 2^{32}). The g function consists of four byte-wide key-dependent S-Boxes followed by linear mixing operation with the 4×4 MDS matrix. The two output words (one rotated by 1 bit towards the right) of the F function are then XORed with the two least significant words of the round input. More details can be found in [15].

Some features of Twofish are different from traditional Feistel block ciphers (e.g., DES), which makes DFA difficult to work. The first one is the PHT operation. Due to modulo addition in PHT, it is impossible to obtain a clear differential characteristic for DFA. The second one is round key addition. Unlike other Feistel ciphers, the round key addition in Twofish is not to the input of S-Box. Even if the attacker retrieves the input-output difference pair of an S-Box, he cannot retrieve the key. The third one is key dependent S-Boxes. Each S-Box uses two bytes that are associated with the key instead of one byte in other block ciphers.

There is only one DFA work on Twofish [28]. As in [28], we assume that one single byte fault is induced into the last round input of Twofish. Then in ciphertext, one byte in the left 64 bits and four bytes in the right become faulty. For the key scheduling, we only build the equation set of generating the key dependent S-Boxes, eight words of the whitening key and the two words of the last round key (2375KB script size, 26600 variables and 100863 CNF equations). For the encryption, we only build the equation set of the last round (each round requires 9608 variables and 33704 CNF equations). 100 AFA executions were distributed on ten computers with the same configuration. Our results show that under Mode A, 280 fault injections (about 280MB script size) can recover the secret key of Twofish in 24 hours with 95% probabilities. Compared with the 320 fault injections and 8 hours offline analysis in [28], our AFA requires less fault injections at the cost of time complexity. In our attack, with 320 fault injections, the solving time remains the same as the one with 280 fault injections. We infer that it was caused by the increase in the script size.

8 Conclusion and Future Work

This paper proposes a generic framework for algebraic fault analysis on block ciphers. The framework could be used to analyze the efficiency of different fault attacks, to compare different scenarios, and to evaluate the factors that may determine the solving time and the success rate.

First, we highlight a conceptual overview of the framework. The important levels and roles are clarified, and four functional parts and three workflow stages are depicted. Second, we select LBlock as a start point to illustrate how our framework can work on a specific cipher, especially a lightweight one. To demonstrate the flexibility of the framework, three scenarios are exploited, which include injecting a fault to encryption, to key scheduling, or to modify the rounds. Third, to demonstrate the generic feature of our framework, more fault attacks are conducted on different block ciphers which are well known and have some typical structures.

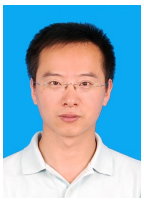
Future work can be derived in different directions. One possible area is to further improve the efficiency of the framework. The current version still meets some difficulties in AFA on deep round of extremely complicated ciphers such as Twofish. With an enhanced solver, more compact equation builders and other advanced techniques, the AFA framework might work with more rounds of Twofish.

Acknowledgment

This work was supported in part by the Major State Basic Research Development Program (973 Plan) of China under Grant 2013CB338004, the National Natural Science Foundation of China under the grants 61173191,61272491,61202386,61309021,61472357,61571063, the Zhejiang University Fundamental Research Funds for the Central Universities under the grant 2015QNA5005, the Science and Technology on Communication Security Laboratory under the grant 9140C110602150C11053 and the European Commission through the ERC project 280141 (acronym CRASH).

REFERENCES

[1] H. B.-E. Hamid, H. Choukri, D. N. M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," in *FDTC*, 2004.
[2] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *EUROCRYPT 1997*, pp. 37–51.
[3] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *CRYPTO 1997*, pp. 513–525.
[4] T. Courtois, D. Ware, and K. Jackson, "Fault algebraic attacks on inner rounds of DES," in *eSmart 2010*.
[5] N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *ASIACRYPT 2002*, pp. 267–287.
[6] X. Zhao, S. Guo, F. Zhang, T. Wang, Z. Shi, and K. Ji, "Algebraic differential fault attacks on LED using a single fault injection," in *IACR Cryptology ePrint Archive*, 2012.
[7] P. Jovanovic, M. Kreuzer, and I. Polian, "An algebraic fault attack on the LED block cipher," in *IACR Cryptology ePrint Archive*, 2012.
[8] F. Zhang, X. Zhao, S. Guo, T. Wang, and Z. Shi, "Improved algebraic fault analysis: A case study on piccolo and applications to other lightweight block ciphers," in *COSADE 2014*, pp. 62–79.
[9] X. Zhao, S. Guo, F. Zhang, Z. Shi, C. Ma, and T. Wang, "Improving and evaluating differential fault analysis on LED with algebraic techniques," in *FDTC*, 2013, pp. 41–51.
[10] W. Wu and L. Zhang, "LBlock: a lightweight block cipher," in *ACNS*, 2011, pp. 327–344.
[11] H. Choukri and M. Tunstall, "Round reduction using faults," in *FDTC*, 2015, pp. 13–24.
[12] A. Dehbaoui, A. P. Mirbaha, N. Moro, J. M. Dutertre, and A. Tria, "Electromagnetic glitch on the AES round counter," in *COSADE 2013*, pp. 17–31.
[13] NIST, "Data encryption standard," in *Federal Information Processing Standards Publications*, May 1977.
[14] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, *PRESENT: An ultra-lightweight block cipher*. Springer, 2007.
[15] B. Schneier and J. Kelsey, "Twofish: A 128-bit block cipher," http://www.schneier.com/paper-twofish-paper.pdf.
[16] X. Zhao, S. Guo, F. Zhang, T. Wang, Z. Shi, and D. G. Chujiao Ma, "Algebraic fault analysis on GOST for key recovery and reverse engineering," in *FDTC*, 2014, pp. 29–39.
[17] F.-X. Standaert, T. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *EUROCRYPT*, 2009, pp. 443–461.
[18] M. Cazorla, K. Marquet, and M. Minier, "Survey and benchmark of lightweight block ciphers for wireless sensor networks," in *SECRYPT*, 2013, pp. 543–548.
[19] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Grobschadl, and A. Biryukov, "Triathlon of lightweight block ciphers for the internet of things." *IACR Cryptology ePrint Archive, 2015/209*.
[20] L. Zhao, T. Nishide, and K. Sakurai, "Differential fault analysis of full LBlock," in *COSADE*, 2012, pp. 135–150.
[21] K. Jeong, C. Lee, and J. I. Lim, "Improved differential fault analysis on lightweight block cipher LBlock for wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 151, pp. 1–9, 2013.
[22] H. Chen and L. Fan, "Integral based fault attack on LBlock," in *ICISC*, 2014, pp. 227–240.
[23] W. Li, J. Zhao., X. Zhao, and J. Zhu, "Algebraic fault analysis on LBlock under nibble-based fault model," in *IMCCC*, 2013, pp. 1525–1529.
[24] L. Knudsen and C. Miolane, "Counting equations in algebraic attacks on block ciphers," *International Journal of Information Security*, vol. 9, no. 2, pp. 127–135, 2010.
[25] M. Rivain, "Differential fault analysis on DES middle rounds," in *CHES*, 2009, pp. 457–469.
[26] J. Li and D. Gu, "Differential fault analysis on PRESENT," in *CHINACRYPT*, 2009, pp. 3–13.
[27] X. Zhao, S. Guo, and F. Zhang, "Fault-propagate pattern based dfa on PRESENT and PRINTcipher," *Wuhan University Journal of Natural Sciences*, vol. 17, no. 6, pp. 485–493, 2012.
[28] S. Ali and D. Mukhopadhyay, "Differential fault analysis of Twofish," in *Inscrypt*, 2013, pp. 10–28.

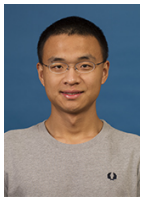


Fan Zhang was born in 1978. He received his Ph.D. degree in Department of Computer Science and Engineering from University of Connecticut in 2012. He is currently working in the College of Information Science and Electrical Engineering, Zhejiang University. He is also with the Science and Technology on Communication Security Laboratory. His research interests include side channel analysis and fault analysis in cryptography, cyber security, computer architecture and sensor network.

Shize Guo was born in 1964. He received his Ph.D. degree in Harbin Institute of Technology in 1989 and his M.S. and B.S. degrees from Ordnance Engineering College, China, in 1991 and 1988, respectively. He is currently a researcher in Institute of North Electronic Equipment and also a Professor in Beijing University of Post and Telecommunications. His main research interest includes information technology and information security.

Xinjie Zhao received his Ph.D., M.S. and B.S. degrees in Ordnance Engineering College in 2012, 2009 and 2006, respectively. He is currently working in Institute of North Electronic Equipment. His main research interest includes side channel analysis, fault analysis and combined analysis in cryptography. He won the best paper award in COSADE 2012 and the outstanding doctoral dissertation award in Hebei province.

Tao Wang was born in 1964. He received his Ph.D. degree in computer application from Institute of Computing Technology Chinese Academy of Sciences in 1996 and masters degree in computer application from Ordnance Engineering College in 1990. He is currently a Professor in Ordnance Engineering College. His research interests include information security and cryptography.



Jian Yang is currently a Ph.D. student of Computer Science and Engineering at University of Notre Dame. He received his Bachelor degree of Engineering from College of Information Science and Electrical Engineering, Zhejiang University. His research interests focus on hardware security and mobile computing.

Francois-Xavier Standaert was born in Brussels, Belgium in 1978. He received the Electrical Engineering degree and PhD degree from the Universite catholique de Louvain, respectively in June 2001 and June 2004. His research interests include digital electronics, FPGAs and cryptographic hardware, low power implementations for constrained environments, the design and cryptanalysis of symmetric cryptographic primitives, physical security issues in general and side-channel analysis in particular.

Dawu Gu is a full professor at Shanghai Jiao Tong University in Computer Science and Engineering Department. He received from Xidian university of China his B.S. degree in applied mathematics in 1992, M.S. in 1995, and Ph.D. degree in 1998 both in cryptography. His current research interests include cryptography, side channel attack, and software security. He leads the Laboratory of Cryptology and Computer Security (LoCCS) at SJTU.