Improved Algebraic Fault Analysis: A Case Study on Piccolo and Applications to Other Lightweight Block Ciphers*

Fan ZHANG[1], Xinjie ZHAO[2,3], Shize GUO[3], Tao WANG[2], and Zhijie SHI[1]

[1] University of Connecticut, Storrs, Connecticut, USA
fan.zhang@engineer.uconn.edu, zshi@engr.uconn.edu
[2] Ordnance Engineering College, Shijiazhuang, Hebei, China
zhaoxinjieem@163.com,twangdrsjz@yahoo.com.cn
[3] The Institute of North Electronic Equipment, Beijing, China
tigerone-gsz@vip.sina.com

Abstract. This paper proposes some techniques to improve *algebraic fault analysis* (AFA). First, we show that building the equation set for the decryption of a cipher can accelerate the solving procedure. Second, we propose a method to represent the injected faults with algebraic equations when the accurate fault location is unknown. We take Piccolo as an example to illustrate our AFA and compare it with *differential fault analysis* (DFA). Only one fault injection is required to break Piccolo with the improved AFA. Finally, we extend the proposed AFA to other lightweight block ciphers, such as MIBS, LED, and DES. For the first time, the full secret key of DES can be recovered with only a single fault injection.

Keywords: Algebraic fault analysis, lightweight cipher, Piccolo, DES.

1 Introduction

1.1 Fault Attack

Cryptographic devices perform cryptographic algorithms to achieve various security goals. The operations of the devices are affected by many external factors such as the temperature and the voltage of power supplies. When these factors change, the devices may not function correctly and will produce incorrect outputs [2,3,4]. Adversaries can intentionally introduce errors, also called *faults*, by changing the operation environments of devices and analyze the wrong outputs to recover the secret key. This type of attack is called *fault attack*, which was first proposed by Boneh et al. [7] in 1996 to break RSA-CRT. The fault attack is one type of *implementation attacks*.

* This work was supported in part by the National Natural Science Foundation of China under the grants 60772082, 61173191, 61272491, and US National Science Foundation under the grant CNS-0644188.

Fault attacks on block ciphers are illustrated in Fig. 1. There are two major phases. In the fault injection phase, adversaries inject faults to the selected ***positions***. In the fault analysis phase, adversaries analyze the differences between the correct and faulty outputs to extract the secret key.

– Throughout this paper, the term ***position*** refers to the round where faults are injected. In contrast, the term ***location*** refers to the nibble or byte index for the injected fault in a specified round.

Suppose T is an intermediate state to be injected with faults. g is the operations performed after T. K_T represents the key variables that are used in g. A correct ciphertext can be written as $C = g(T, K_T)$. Suppose T_f denotes the faulty intermediate state of T and f stands for the faults ($f = T + T_f$). A faulty ciphertext can be written as $C^\star = g(T_f, K_T)$. Let h be the key scheduling function: $K_T = h(K)$. Adversaries can build a system of equations for $C, C^\star, K_T, f$. The search space of K can be narrowed down by analyzing the equation system.

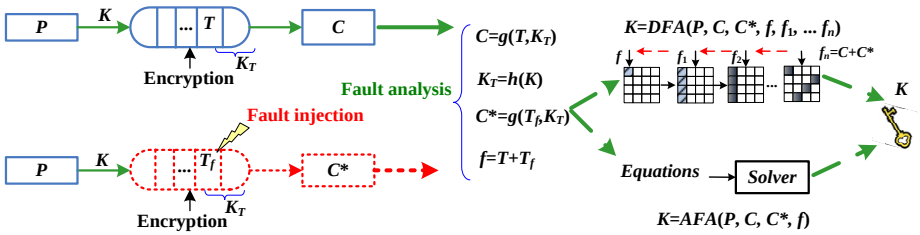


Fig. 1. Overview of fault attacks on block ciphers

1.2 Related Work

Since the proposal of fault attacks, the most widely studied fault analysis technique is *differential fault attack* (DFA), which was used by Biham and Shamir [5] to break DES. After that, DFA has been successfully applied to many other block ciphers [1,20,22,29,30,36,37].

Suppose there are n operations after faults are injected. f_i denote the fault differences at the i-th operation. In the standard DFA, the adversaries collect the faulty ciphertext $C^\star$ to compute f_n, the differences of the correct and faulty ciphertexts. For each of the n operations, the exact locations of the faults have to be determined from f_n. The fault differences f_i are manually deduced. Finally, all the f_i are used to reduce the search space of K. The plaintext P and correct ciphertext C are used to verify the recovered key.

However, there are some difficulties in launching DFAs. (1) DFA relies on the manual calculation of the fault propagation, which is intrinsically hard to be extended to more rounds as the computational complexity increases exponentially. (2) Different approaches of DFAs on the same cipher with the same fault model will still generate different results, which highly depends on how the adversaries

understand the cipher [1,20,22,29,37]. (3) DFA has to be customized for different fault models or different ciphers. It might be difficult when the fault propagation becomes complicated. As a result, it is necessary to find out an automatic, accurate and generic approach for fault attacks.

In eSmart 2010, Courtois et al. [10] proposed a new fault analysis technique known as *algebraic fault attack* (AFA)[4], which combines algebraic cryptanalysis [9] with fault attacks. They showed that if 24 key bits are assumed to be known and two bits in the 13th round are altered, DES can be broken with a single fault injection in 0.01 hour. To launch the full attack in practice, the adversaries have to enumerate the unknown 24 key bits. The maximal time complexity is $2^{17.35}$ hours, which is 10 times faster than the brute force [9].

AFA converts both the cipher and the faults into algebraic equations, and recovers the secret key with automatic tools such as SAT solvers. Unlike DFA, AFA does not require the manual analysis on fault propagations. In COSADE 2011, AFA [27] is used to improve DFA on the stream cipher Trivium [17,18]. The inner state of Trivium can now be recovered using only two fault injections and 420 key stream bits. In Crypto 2011, AFA [8] was used to improve DFA on AES [30] with a customized solver. The secret key of AES can be recovered with only one second if a single byte fault is injected into the 7th round of AES. Recently, AFA [23,40] is used to improve DFA on the lightweight block cipher LED [20,22] in COSADE 2012. The full key of LED is extracted with a single fault injection within three minutes on a PC, which is much more efficient than standard DFAs [20,22].

1.3 Motivations

Recent decades have seen an ever increasing need for efficient cryptography in resource-constrained environments such as smart cards, RFID tags, and IC-printing. This has spurred the study in lightweight cryptography, especially the ultra-lightweight block ciphers, such as mCrypton [25], PRESENT [6], MIBS [19], Piccolo [34] and LED [16]. Due to their compact design, the complexity of algebraic equations representing the lightweight block ciphers is not high. It is also easier to inject faults into devices that adopt such lightweight algorithms because often they are less protected and thus more vulnerable to fault attacks, which is noted in [15]. Therefore, it is important to study the AFAs on the lightweight block ciphers.

Piccolo [34] is a lightweight block ciphers proposed at CHES 2011. There are two versions: Piccolo-80 and Piccolo-128. Piccolo stands for Piccolo-80 if not explicitly specified in this paper. Recently, the security of Piccolo against fault attacks was studied with standard DFAs [21]. Based on a random byte fault model in the penultimate (24th) round, DFA can recover the key of Piccolo-80 with six fault injections. As to Piccolo-128, DFA requires eight fault injections [21]. This paper takes Piccolo as an example to study how to improve DFA with AFA techniques and then extends them to other lightweight block ciphers.

[4] The nature of our *algebraic fault analysis* is the same as the *algebraic fault attack*. So we use the same abbreviation for both.

1.4 Contributions

In this paper, we propose several improvements to DFAs on Piccolo [21] by using AFA. Our contributions are summarized as follows: Firstly, we show that building the equation set for the decryption of a cipher can accelerate the solving procedure. Secondly, we propose a method to represent the injected faults with algebraic equations when the accurate fault location is unknown. We compare our results of AFA on Piccolo with previous DFA work. Only one fault injection is required, which is better than previous work [5,10,23,32,38]. The running time in practice is also affordable. Finally, we extend the proposed AFA to other lightweight block ciphers, such as MIBS, LED, and DES. For the first time, the full secret key of DES can be recovered with only a single fault injection. The advantages of our AFA attacks on Piccolo can be summarized as follows.

1. **Require smaller numbers of fault injections.**
   Assuming the faults are injected into one nibble in the antepenultimate (23rd) round, and the locations and values of the faults are unknown, our AFA requires only one fault injection to recover all the 80 key bits of Piccolo-80, which is less than the six fault injections needed in [21].
2. **Leverage automatic tools.** Unlike previous DFA work, our AFA does not require the manual analysis on Piccolo which limits the number of rounds and propagation paths that can be analyzed. Adversaries only need to build algebraic equations for the cipher and faults.
3. **Extendible.** Our AFA can be extended to work with different fault models such as byte (or word) based fault model and faults located in deep rounds.

1.5 Organization

Section 2 describes the Piccolo algorithm. Section 3 introduces the fault model of this paper. Section 4 describes our AFA on Piccolo. Section 5 presents the experiment results. Section 6 describes the applications of AFA on other lightweight block ciphers and Section 7 concludes the paper.

2 The Description of Piccolo

The full specification of Piccolo can be found in [34]. This section briefly describes the Piccolo algorithm. Table 1 defines the notations used in this paper. Piccolo has the Feistel structure and a block size of 64 bits. There are two variants: Piccolo-80 uses 80-bit keys and 25 rounds, and Piccolo-128 uses 128-bit keys and 31 rounds. Fig. 2(a) shows the encryption of Piccolo.

The operations in one round of Piccolo include two F functions ($F : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$, as shown in Fig. 2(b), an AddRoundKey function (AK) and a RoundPermutation function ($RP : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$). The F function is composed of three operations: SubNibble, MixColumn, and SubNibble. The AK function XORs the output of F with the round key. The RP function groups the 64 bits of a block into eight bytes and permutes the bytes as shown in Fig. 2(c).

Table 1. Notations in the description of Piccolo

Variable	Notations	Variable	Notations
P, C	plaintext, ciphertext	rk_i	the subkey in round $i/2+1$
K	master key	$a\|b$	the concatenation of a and b
r	round number	$a_{(b)}$	there are b bits in a
wk_i	i-th 16-bits of the whitening key	$+$	bitwise XOR

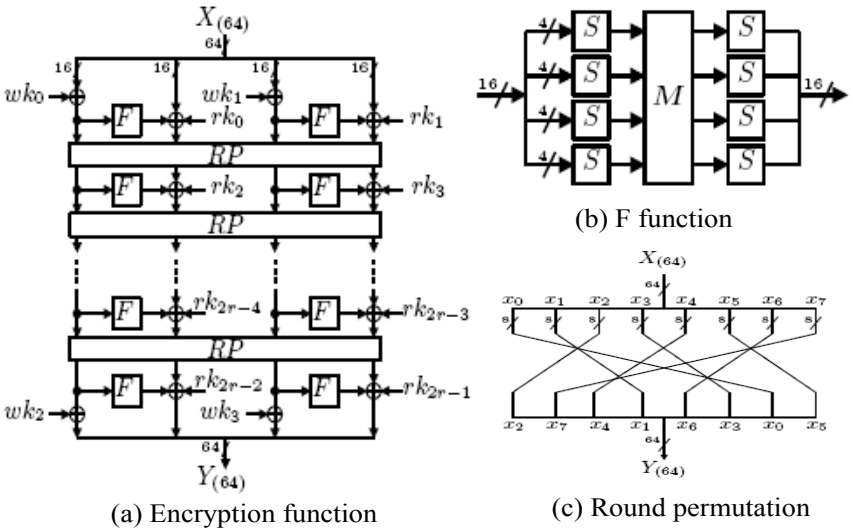


Fig. 2. Specifications of Piccolo

The F function has 16-bit input and output, which are grouped as four nibbles. In the SubNibble operation, each nibble goes through an S-box, which is shown in Table 2. The MixColumn views the four nibbles as a 4×1 vector and multiplies it with a 4×4 matrix M. The structure of M can be found in [34]. The four nibbles generated by MixColumn go through the SubNibble operation again and become the output of the F function.

Table 2. 4-bit (as a hexadecimal digit) bijective S-box in Piccolo

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	E	4	B	2	3	8	0	9	1	A	7	F	6	C	5	D

In Piccolo, prewhitening and postwhitening are done before the first round and after the last round, respectively. In the whitening steps, the 64-bit block

is split into two halves and the left 16 bits of each half are XORed with the prewhitening or postwhitening key. The key scheduling of Piccolo produces four 16-bit whitening keys $wk_i (0 \leq i < 4)$ and two 16-bit round keys for each round. These keys are generated by XORing the master key with 16-bit constants.

3 Fault Model

The fault model assumed for AFA on Piccolo in this paper is described as follows.

- The adversary can choose the plaintext to be encrypted and obtain the corresponding correct and faulty ciphertext.
- The adversary can inject a fault. So one of the nibbles at the input of F functions in the 23rd round is wrong, as shown in Fig. 3. In Section 6, this assumption can be further weaken when extending our AFA to more rounds.
- The adversary knows the fault position but does not know the exact location nor the value of faults. In other words, he can specify which round to inject the faults, but has no control either on which byte or nibble to be altered, nor on the values.

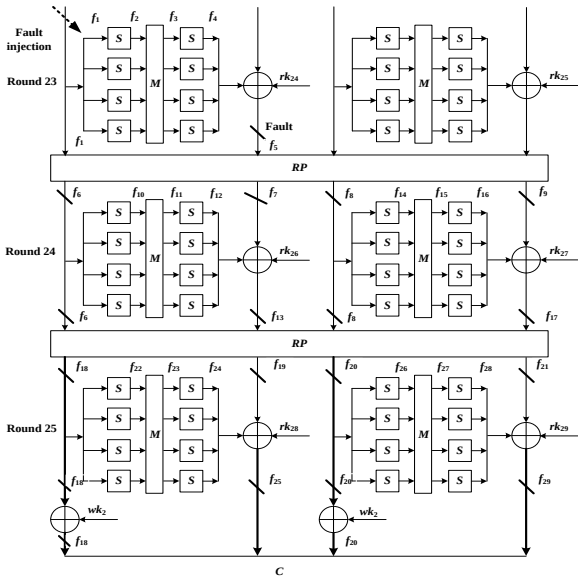


Fig. 3. Fault model of AFA on Piccolo

As shown in Fig. 3, the fault model is difficult to adopt in DFA on Piccolo [21]. This is because most of the nibbles in the ciphertext are faulty. It is difficult to manually deduce the accurate fault location and the fault propagation path in

the last three rounds from the differences observed in the ciphertext. It will become more challenging if a nibble fault is injected into the 22nd round.

4 A Case Study of AFA on Piccolo

4.1 The Framework of AFA

Traditional DFA relies on manual analysis. Its efficiency is limited in scenarios that have very high complexity, for example, when faults are located in deeper rounds of the cipher, or when the exact location of the injected faults in a deep round is unknown. AFA combines fault analysis with algebraic techniques and the analysis can be done with automatic tools. AFA consists of four steps.

Inducing the faults. The number of faults to be injected and their positions are carefully chosen. How to inject a fault can be referred to [2,3,4].

Constructing equations for the cipher. The overall cipher and its key scheduling are represented as equations. The most difficult part in this step for Piccolo is to represent non-linear functions like SubNibble and complicated linear functions like MixColumn. How to build the equation set of cipher suitable for AFA (which is different from traditional algebraic cryptanalysis [9]) is also very important to the efficiency of the attack.

Constructing equations for the faults. After the fault injections, the faults are also represented with algebraic equations. A difficult problem is to represent the faults when their exact location (e.g., the index of the faulty byte or nibble) is not known.

Solving the equation system. The problem of searching for the master key is now transformed into solving the merged equation system. Many automatic tools [12,13,26,35] can be leveraged.

4.2 Constructing Equations for Decryption of Piccolo

The goal of this phase is to represent Piccolo with a large system of low degree boolean equations. Since fault attacks start with analyzing the ciphertext, we will first build the equation set for the decryption of Piccolo.

Suppose $A_i = \{A_{i,1(16)}, A_{i,2(16)}, A_{i,3(16)}, A_{i,4(16)}\}$ is the 64-bit input of round i, $B_i = \{B_{i,1(16)}, B_{i,2(16)}, B_{i,3(16)}, B_{i,4(16)}\}$ is the 64-bit input of the RP function in round i, and $C = \{C_{1(16)}, C_{2(16)}, C_{3(16)}, C_{4(16)}\}$ is the ciphertext. Since the key scheduling of Piccolo is very simple, we will focus on the decryption. Algorithm 1 below lists the full decryption of Piccolo.

From Algorithm 1, the most important yet difficult problem is to construct the equations for SubNibble and MixColumn operations in function F.

The equations for SubNibble. Suppose the input and output of SubNibble are $X = X_{1(4)}|X_{2(4)}|X_{3(4)}|X_{4(4)}$ and $Y = Y_{1(4)}|Y_{2(4)}|Y_{3(4)}|Y_{4(4)}$, respectively. Y can be represented as

$$Y_1 = S(X_1), \quad Y_2 = S(X_2), \quad Y_3 = S(X_3), \quad Y_4 = S(X_4) \tag{1}$$

Algorithm 1 The decryption of Piccolo

$C = \{C_1, C_2, C_3, C_4\}$
$A_{25,1} = C_1 + wk_2, \quad A_{25,2} = F(A_{25,1}) + C_2 + rk_{48}$
$A_{25,3} = C_3 + wk_3, \quad A_{25,4} = F(A_{25,3}) + C_4 + rk_{49}$
$for \;\; i = 24 \;\; to \;\; 2 \;\; do \;\; \{$
  $A_{i+1} = RP(B_i)$
  $A_{i,1} = B_{i,1}, \quad A_{i,2} = F(A_{i,1}) + B_{i,2} + rk_{2i-2}$
  $A_{i,3} = B_{i,3}, \quad A_{i,4} = F(A_{i,3}) + B_{i,4} + rk_{2i-1}$
$\}$
$A_2 = RP(B_1)$
$A_{1,1} = B_{1,1} + wk_0, \quad A_{1,2} = F(A_{1,1}) + B_{1,2} + rk_0$
$A_{1,3} = B_{1,3} + wk_1, \quad A_{1,4} = F(A_{1,3}) + B_{1,4} + rk_1$
$P = \{A_{1,1}, A_{1,2}, A_{1,3}, A_{1,4}\}$

In Eq.(1), $S(\cdot)$ denotes one S-box lookup. Suppose the input and output of one S-box are $x_1|x_2|x_3|x_4$ and $y_1|y_2|y_3|y_4$. The S-box in Piccolo can be represented with the following four equations [24].

$$\begin{aligned} y_1 &= 1 + x_1 + x_2 + x_4 + x_1x_2 \\ y_2 &= 1 + x_1 + x_2 + x_3 + x_2x_3 \\ y_3 &= 1 + x_1 + x_4 + x_1x_2 + x_1x_3 + x_2x_3 + x_3x_4 + x_1x_2x_3 \\ y_4 &= x_1 + x_2 + x_3 + x_1x_3 + x_1x_4 + x_2x_4 + x_3x_4 + x_1x_2x_3 + x_2x_3x_4 \end{aligned} \tag{2}$$

The equations for MixColumn. Suppose the input and output of MixColumn are $X = X_{1(4)}|X_{2(4)}|X_{3(4)}|X_{4(4)}$ and $Y = Y_{1(4)}|Y_{2(4)}|Y_{3(4)}|Y_{4(4)}$, respectively. Y can be represented as

$$\begin{aligned} Y_1 &= 2 \cdot X_1 + 3 \cdot X_2 + 1 \cdot X_3 + 1 \cdot X_4 \\ Y_2 &= 1 \cdot X_1 + 2 \cdot X_2 + 3 \cdot X_3 + 1 \cdot X_4 \\ Y_3 &= 1 \cdot X_1 + 1 \cdot X_2 + 2 \cdot X_3 + 3 \cdot X_4 \\ Y_4 &= 3 \cdot X_1 + 1 \cdot X_2 + 1 \cdot X_3 + 2 \cdot X_4 \end{aligned} \tag{3}$$

where $\cdot$ denotes the multiplication in $GF(2^4)$ with an irreducible polynomial $x^4 + x + 1$.

Suppose the 4-bit input and 4-bit output of a multiplication in $GF(2^4)$ are denoted as $x_1|x_2|x_3|x_4$ and $y_1|y_2|y_3|y_4$. y_i can be represented with x_i, depending on the coefficients from M. Table 3 shows how this is done for three different coefficients in M.

When the F function is substituted in the final equation set, each decryption round needs 544 variables and 928 ANF equations. In addition, 32 variables and ANF equations are needed for round keys, and 64 variables and ANF equations are for the whitening keys.

Table 3. Representing the multiplications for all coefficients in M

Matrix element	y_1	y_2	y_3	y_4
1	x_1	x_2	x_3	x_4
2	x_2	x_3	$x_1 + x_4$	x_1
3	$x_1 + x_2$	$x_2 + x_3$	$x_1 + x_3 + x_4$	$x_1 + x_4$

4.3 Constructing Equations for Faults with Unknown Locations

Previous AFAs assume the location of the injected faults is known [10,23,40]. Inspired by the work in [39], this section proposes a new method to represent the faults when their locations are unknown, which is very important when extending fault attacks to deep rounds.

Suppose the correct input to the F function in the 23rd round is denoted as $X = x_1|x_1|\dots|x_{16}$. The faulty input after the fault injections is denoted as $Y = y_1|y_1|\dots|y_{16}$. The injected fault can be represented as

$$Z = z_1|z_1|\dots|z_{16}, \quad z_i = x_i + y_i, \quad 1 \le i \le 16 \tag{4}$$

Z can be considered as the concatenation of four nibbles $Z_{1(4)}|Z_{2(4)}|Z_{3(4)}|Z_{4(4)}$, where $Z_i = z_{4i-3}|z_{4i-2}|z_{4i-1}|z_{4i}$ $(1 \le i \le 4)$. Four one-bit variables u_i are introduced to represent whether Z_i is faulty or not.

$$u_i = (1 + z_{4i-3}) \wedge (1 + z_{4i-2}) \wedge (1 + z_{4i-1}) \wedge (1 + z_{4i}), \quad 1 \le i \le 4 \tag{5}$$

where u_i is zero if Z_i is faulty. Since there is only one fault injected, only one of u_i $(1 \le i \le 4)$ is zero. The constraint can be represented as

$$(1 + u_1) \vee (1 + u_2) \vee (1 + u_3) \vee (1 + u_4) = 1, u_i \vee u_j = 1, \quad 1 \le i < j \le 4 \tag{6}$$

The injected fault under our assumptions can be fully represented with Equations (4), (5), and (6), which are simple and straightforward.

4.4 Solving the Equation System

Finally, the equation system can be solved for the key variables. In this paper, CryptoMiniSAT, a SAT-based solver, is used. SAT-based solvers are widely studied in previous work [10,22,28,31,39,40]. Many other automatic tools, such as mutantXL algorithm [12,26], and Gröbner basis-based [13] solvers can also be considered. However one major problem for those solvers [12,13,26] is the memory usage when solving large equations systems even if they are sparse. Recently significant improvements have been made to SAT solvers. Therefore we have chosen SAT-based solvers in algebraic cryptanalysis. More specifically, we choose the CryptoMiniSAT v2.9.4 [35] which won the gold prize in the *SAT Race* competition [33] in 2010. The readers can refer to [14,33] for details of how to generate equations and how to feed them to the solvers.

5 Experiment Results

This section presents our experimental results of AFA on Piccolo. The fault injections are simulated in software. Piccolo is implemented with the C language and the source code is modified to induce faults. The CryptoMiniSAT solver is running on a laptop with Intel Core I7-2640M, 2.80 GHZ, 4G memory, Windows XP 64-bit OS. An *instance* refers to one run of our AFA on a set of P, K, C. The instance fails if the solver does not give an output within 48 hours.

5.1 Basic Attack

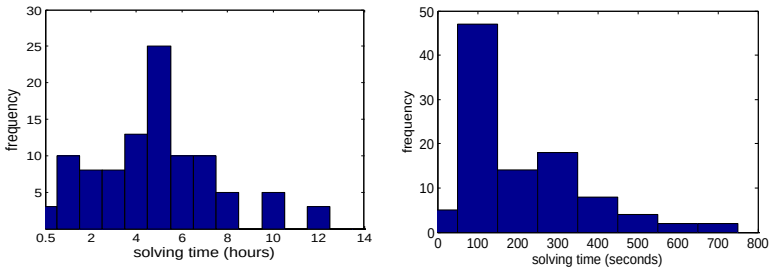
To describe our experiment, we use an example where P=0x3cb19a75b8b476dc, K=0xce076dfb5300f7ea6bf1, and C=0xbc81c26238001c7c. A nibble fault is injected in the 23rd round, generating $C^{\star}$=0xc6f545612c08c9d4. In the attack, we first build the encryption set of full Piccolo encryption for the correct values (both P and C are fed into the equations) and for the last three rounds of the faulty encryption (only C^{*} is fed into the equations). We use the method in Section 4.3 to build the algebraic equations for the faults. The full equation set has 18,317 variables and 30,112 ANF equations. The size of the scripts is about 580K per instance.The solver can not output the solution within 48 hours.

Then, we try to build the algebraic equations for the full Piccolo decryption for the correct values and for the last three rounds of the faulty decryption using the method in Section 4.2. The full equation set has 17,129 variables and 28,016 ANF equations. The size of the scripts is about 553K per instance. The solver can output the correct solution in about five hours. Table 4 lists the output from the CryptoMiniSAT solver, *i.e.*, the binary representation of the extracted key for one instance. Variables No. 2 to No. 81 are the 80 secret key bits of Piccolo-80. A key bit is 1 if the corresponding index is positive, and is 0 otherwise. The recovered key in Table 4 is 0xce076dfb5300f7ea6bf1 in hexadecimal. All the bits in the recovered key are correct.

Table 4. The output of CryptoMiniSAT in one run of AFA on Piccolo

No.	key	No.	key	No.	key	No.	key	No.	key	No.	key	No.	key	No.	key	No.	key	No.	key
2	1	-10	0	-18	0	26	1	-34	0	-42	0	50	1	58	1	-66	0	74	1
3	1	-11	0	19	1	27	1	35	1	-43	0	51	1	59	1	67	1	75	1
-4	0	-12	0	20	1	28	1	-36	0	-44	0	52	1	60	1	68	1	76	1
-5	0	-13	0	-21	0	29	1	37	1	-45	0	53	1	-61	0	-69	0	77	1
6	1	-14	0	22	1	30	1	-38	0	-46	0	-54	0	62	1	70	1	-78	0
7	1	15	1	23	1	-31	0	-39	0	-47	0	55	1	-63	0	-71	0	-79	0
8	1	16	1	-24	0	32	1	40	1	-48	0	56	1	64	1	72	1	-80	0
-9	0	17	1	25	1	33	1	41	1	-49	0	57	1	-65	0	73	1	81	1

To verify the success rate of our attacks, 100 instances are tested with different plaintexts and keys. Ten laptops are used to run the solver in parallel to finish these attacks. Each laptop runs 10 instances. With only one fault injected



(a) with one fault (b) with two faults

Fig. 4. Statistics of the solving time in AFA on Piccolo

in the 23rd round, AFA can recover the correct key in all the 100 instances. The statistics of the solving time is listed in Fig. 4(a). We can see that the execution time seems to follow an exponential distribution (as noted in [31,39]). It varies from 0.5 hour to 12 hours, with an average of 5 hours. The results also show that building the decryption set of ciphers can improve the efficiency of AFA. With two fault injected in the 23rd round, all the 100 instances can recover the 80-bit secret key within 700 seconds on average, as shown in Fig. 4(b).

5.2 Extend to Different Fault Widths

In Section 5.1, the width of the induced faults is 4 bits (one nibble). The width can be increased. We experimented with AFA on Piccolo-80 using a single fault of 8 or 16 bits, and ran each model for 100 instances. The key can be recovered in 6 and 22 hours on average. The maximal time is 15.86 and 28.54 hours respectively. Under the same 8-bit fault model, our attack requires fewer faults than the attack in [21], which requires six fault injections.

5.3 Extend to Different Fault Positions

The position of the injected faults can also be changed. In addition to the 23rd round, an adversary can induce a random fault at the input of the F functions in the 22nd round. Although the full avalanche effect has been achieved in the last four rounds, our AFA can still work if the number of fault injections is increased. Our experiments show that two fault injections can recover the key. We run the attack with 100 random instances. On average, the attack takes 5 and 6 hours under 4-bit and 8-bit fault models, respectively. The maximal time is 15.54 and 17.25 hours respectively.

5.4 Extend to Different Variants of Piccolo

AFA can also be applied to Piccolo-128. In order to extract the master key, the last four round keys have to be known first. This requires multiple fault

injections in two different rounds. Table 5 lists the solving time under different widths and positions of faults for Piccolo-128. Each scenario is tested with 100 random instances. We can see that the fault models of AFA on Piccolo-128 are quite flexible and the adversaries can choose multiple positions for the fault injections. AFA requires three faults in contrast to eight faults needed in [21].

Table 5. Results of AFA on Piccolo-128

Fault model	Number of faults	Time (seconds)
4-bit fault, the 29th round(2) + 26th round (1)	3	78
4-bit fault, the 29th round(2) + 27th round (1)	3	58
4-bit fault, the 30th round(2) + 27th round (1)	3	1800
8-bit fault, the 29th round(2) + 26th round (1)	3	65
8-bit fault, the 29th round(2) + 27th round (1)	3	59
8-bit fault, the 30th round(2) + 27th round (1)	3	1800

6 Applications to Other Lightweight Block Ciphers

Our AFA can be easily extended to break other ciphers such as MIBS, LED, DES and AES. Let n_w denote the fault width and n_c denote the index of the round where faults are injected. Table 6 lists some results of our AFAs on MIBS, LED, DES and AES under different fault models.

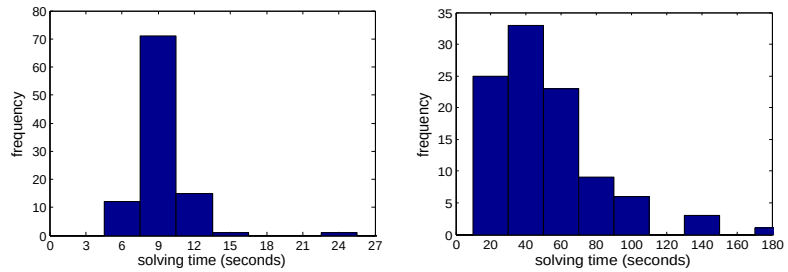
Table 6. Results of AFA on different ciphers

Attack	Block cipher	Fault model	Technique	Faults	Time
[29]	AES-128	n_w=8,n_c=7	DFA	1	2^{32} encryption
[37]	AES-128	n_w=8,n_c=7	DFA	1	50 minutes
[1]	AES-128	n_w=8,n_c=7	DFA	1	5 minutes
[8]	AES-128	n_w=8,n_c=7	AFA	1	1 second
This paper	AES-128	n_w=8,n_c=7	AFA	1	10 hours
[5]	DES	n_w=1,n_c=14,15,16	DFA	3	—
[10]	DES	n_w=2,n_c=14	AFA	2	$2^{13.35}$ hours
[10]	DES	n_w=2,n_c=13	AFA	1	$2^{17.35}$ hours
[32]	DES	n_w=1,n_c=12	DFA	7	–
This paper	DES	n_w=1,n_c=12	AFA	1	10 seconds
[32]	DES	n_w=8,n_c=12	DFA	9	–
This paper	DES	n_w=8,n_c=12	AFA	1	60 seconds
[32]	DES	n_w=1,n_c=11	DFA	11	–
This paper	DES	n_w=1,n_c=11	AFA	1	900 seconds
[38]	MIBS-64	n_w=4,n_c=30	DFA	1	60 seconds
This paper	MIBS-64	n_w=4,n_c=29	AFA	1	1100 seconds
[23]	LED-64	n_w=4,n_c=30	AFA	1	14.67 hours
This paper	LED-64	n_w=4,n_c=30	AFA	1	180 seconds

In the attack, we build the decryption equation set for these ciphers and represent the faults with algebraic equations using the method in Sec 4.3. Finally, these equations are combined and fed into the CryptoMiniSAT solver. We run 100 instances for each attack scenario.

In terms of the number of faults, our experiments have the best results for all the ciphers in Table 6, which demonstrates the advantage of AFA. The efficiency of AFA depends on the algebraic structure of the cipher and fault models. The time needed for solving equations is short for lightweight ciphers such as MIBS, LED, and DES, and longer for block ciphers with more complicated algebraic structures such as AES.

When applied to AES, our SAT-based approach is less efficient than DFA in [1,29,37] and AFA in [8], as shown in Table 6. There are two reasons. The first is that the algebraic structure of AES (especially the 8×8 S-Box) is complicated for the SAT solver. The second is that the solver used is not customized for fault attacks on AES. The attack in [8] is extremely fast because it relies on a customized solver.



(a) DES,n_w=1,n_c=12, n_v=18329,n_a=104073,n_s=2021K

(b) DES,n_w=8,n_c=12, n_v=18350,n_a=105075,n_s=2025K

(c) MIBS,n_w=4,n_c=29, n_v=13505,n_a=20514,n_s=426K

(d) LED,n_w=4,n_c=30, n_v=21131,n_a=35389,n_s=775K

Fig. 5. Statistics of the solving time in AFA on ciphers with a single fault
n_v: the number of variables, n_a: the number of ANF equations, n_s: the script size

Since the first proposal of the DFA on DES [5], we show that the key of DES can be recovered with only a single fault injection for the first time. In

comparison, three faults are required in [5], and 7 to 11 faults are required in [32]. Compared to the first AFA on DES, the work in [10] also requires only a single fault injection in the 13th round. However the condition is that the adversary knows about 24 key bits before the attack. To launch the full attack, he has to enumerate the unknown 24 key bits in practice. The attack requires about $2^{17.35}$ hours, which is still not very practical with a common laptop. Take the following fault model as an example, in which one bit or one byte fault injected to the left part of the DES internal state at the end of the 12-th round. Our AFA can recover the secret key within a few minutes (as shown in Fig. 5(a),5(b)), which is more efficient than that in [10]. In fault attacks on MIBS, our AFA can recover the secret key with an average time of 1100 seconds on average if 4-bit fault is injected into the 29th round (Fig. 5(c)), which is deeper than the fault model in [38]. In fault attacks on LED, our AFA requires much less time than those in previous AFA work in [23] if a 4-bit fault is injected into the 30th round (Fig. 5(d)). The reason is that we built the algebraic equations for the decryption instead of encryption, which significantly accelerates the solving process.

7 Conclusion

This paper proposes an improved *algebraic fault analysis* (AFA) technique, which builds the decryption equation set of the cipher instead of the encryption set and gives a method to represent the faults with algebraic equations even when both the value and location of the faults are unknown. We take Piccolo as an example to verify the proposed AFA and then extend it to some other lightweight block ciphers. The results show that AFA has several advantages in fault attacks on lightweight block ciphers compared with traditional DFA. The improved AFA (1) requires small numbers of fault injections, (2) uses simple and automatic analysis, and (3) is generic and easy to be extended to other models and ciphers. The future work includes:

- Reducing the equation solving time. The equations in AFA can be constructed in many ways that result in different numbers of variables and ANFs. Thus, the construction of equations can affect the performance of solvers like CryptoMiniSAT. Meanwhile, it is also interesting to try other techniques to solve the algebraic equations, such as mutantXL algorithm[12,26], Gröbner basis-based [13] and customized solvers [8].
- Analyzing the complexity of AFA. AFA can be applied to different ciphers with different fault models. Understanding how different factors affect AFA is important for evaluating the efficiency of the attacks.
- Studying attacks and countermeasures in practice. The experiment results in this paper are from simulations. Although they help us understand AFA, it is also important to evaluate the attacks and to design corresponding countermeasures on real devices.

Acknowledgments. The authors would like to thank Ruilin Li and the anonymous referees for helpful discussions and comments.

References

1. S. Ali, D. Mukhopadhyay, M. Tunstall. Differential fault analysis of AES: towards reaching its limits, Journal of Cryptographic Engineering, 2012, DOI 10.1007/s13389-012-0046-y.
2. M. Agoyan, J. Dutertre, D. Naccache, B. Robisson, and A. Tria, When clocks fail: On critical paths and clock faults. In Smart Card Research and Advanced Application, pp. 182-193, 2010.
3. H. Bar-El, H. Choukri,D. Naccache, M. Tunstall, C. Whelan. The Sorcerers Apprentice Guide to Fault Attacks. In IEEE 94, pp. 370-382, 2006.
4. A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault injection attacks on cryptographic devices: Theory, practice and countermeasures. Politecnico di Milano, Milan, Italy, Tech. Rep., 2012.
5. E. Biham, A. Shamir. Differential Fault Analysis of Secret Key Cryptosystem. In CRYPTO 1997, LNCS, vol. 1294, pp. 513-525, 1997.
6. A. Bogdanov, L.R. Knudsen, G. Leander, et al. PRESENT: An Ultra-Lightweight Block Cipher. In CHES 2007, LNCS, vol. 4727, pp. 450-466, 2007.
7. D. Boneh, R.A.DeMillo, R.J.Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In EUROCRYPT 1997, LNCS, vol. 1233, pp. 37-51, 1997.
8. C. Bouillaguet, P. Derbez, and P.-A. Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In CRYPTO 2011, LNCS, vol. 6841, pp. 169-187, 2011.
9. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Asiacrypt 2002, LNCS, vol. 2501, pp. 267-287, 2002.
10. N. Courtois, D. Ware, K. Jackson. Fault-Algebraic Attacks on Inner Rounds of DES. In eSmart 2010, pp. 22-24, 2010.
11. P. Derbez, P.-A. Fouque, and D. Leresteux. Meet-in-the-Middle and Impossible Differential Fault Analysis on AES. In CHES 2011, LNCS, vol. 6917, pp. 274-291, 2011.
12. J. Ding, J. Buchmann, M.S.E. Mohamed, et al. MutantXL algorithm. In Proceedings of the 1st International Conference in Symbolic Computation and Cryptography, pp. 16-22, 2008.
13. J. C. Faugère, Gröbner Bases. Applications in Cryptology. In FSE 2007, Invited Talk, available at: http://fse2007.uni.lu/slides/faugere.pdf.
14. V.B. Gregory. Algebraic Cryptanalysis. Published by Springer, 2009.
15. D. Gu, J. Li, S. Li, Z. Guo and J. Liu, Differential Fault Analysis on Lightweight Blockciphers with Statistical Cryptanalysis Techniques. In FDTC 2012, pp. 27-33, 2012.
16. J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw. The LED Block Cipher. In CHES 2011, LNCS, vol. 6917, pp. 326-341, 2011.
17. M. Hojsik and B. Rudolf., B. Differential fault analysis of Trivium. In FSE 2008, LNCS, vol. 5086, pp. 58-172, 2008.
18. M. Hojsik and B. Rudolf., B. Floating fault analysis of trivium. In INDOCRYPT 2008, pp. 239-250, 2008.
19. M. Izadi, B. Sadeghiyan, S.S. Sadeghian, et al. MIBS: A New Lightweight Block Cipher. In CANS 2009, LNCS, vol. 5888, pp. 334-348, 2009.
20. K. Jeong and C. Lee. Differential Fault Analysis on Block Cipher LED-64. Future Information Technology, Application, and Service, LNEE, vol. 164, pp. 747-755, 2012.
21. K. Jeong. Kitae Jeong. Differential Fault Analysis on Block Cipher Piccolo. Cryptology ePrint Archive, available at http://eprint.iacr.org/2012/399.pdf, 2012.

22. P. Jovanovic, M. Kreuzer, and I. Polian. A Fault Attack on the LED Block Cipher. COSADE 2012, LNCS, vol. 7275, pp. 120-134, 2012.
23. P. Jovanovic, M. Kreuzer and I. Polian, An Algebraic Fault Attack on the LED Block Cipher. Cryptology ePrint Archive. Available: http://eprint.iacr.org/2012/400.pdf, 2012.
24. L.R. Knudsen, C.V. Miolane. Counting equations in algebraic attacks on block ciphers. International Journal of Information Security, vol. 9, No. 2, pp. 127-135, 2010.
25. C. Lim, T. Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors. In WISA 2005, LNCS, vol. 3786, pp. 243-258, 2006.
26. M. Mohamed, W.S.A.E Mohamed, J. Ding, and J. Buchmann. MXL2: Solving Polynomial Equations over GF(2) using an Improved Mutant Strategy. In Proceedings of The Second international Workshop on Post-Quantum Cryptography, LNCS, vol. 5299, pp. 20-215, 2008.
27. M. Mohamed, S. Bulygin and J. Buchmann. Improved Differential Fault Analysis of Trivium. In COSADE 2011, pp. 147-158, 2011.
28. M. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter. Improved Algebraic Side-Channel Attack on AES. Cryptology ePrint Archive. Available: http://eprint.iacr.org/2012/084.pdf, 2011.
29. D. Mukhopadhyay. An Improved Fault Based Attack of the Advanced Encryption Standard. AFRICACRYPT 2009, LNCS, vol. 5580, pp. 421-434, 2009.
30. G. Piret, J.J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In CHES 2003, LNCS, vol. 2779, pp. 77-88, 2003.
31. M. Renauld, F.-X. Standaert. Algebraic Side-Channel Attacks. In INSCRYPT 2009, LNCS, vol. 6151, pp. 393-410, 2009.
32. M. Rivain. Differential Fault Analysis on DES Middle Rounds. In CHES 2009, LNCS, vol. 5747, pp. 457-469, 2009.
33. SAT. Sat Race Competition. http://www.satcompetition.org/.
34. K. Shibutani, T. Isobe, H. Hiwatari, et al. Piccolo: An Ultra-Lightweight Blockcipher. In CHES 2011, LNCS, vol. 6917, pp. 342-357, 2011.
35. M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In SAT 2009, LNCS, vol. 5584, pp. 244-257, 2009.
36. J. Takahashi and T. Fukunaga. Improved Differential Fault Analysis on CLEFIA. In FDTC 2008, pp. 25-34, 2008.
37. M. Tunstall, D. Mukhopadhyay, S. Ali. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In WISTP 2011, LNCS, vol. 6633, pp. 224-233, 2011. Also in Cryptology ePrint Archive, http://eprint.iacr.org/2009/575, 2009.
38. X. Zhao, T. Wang, S. Wang, and Y. Wu. Research on deep differential fault analysis against MIBS. Joumal on Conllnanications, 2010, vol. 31, No. 12, pp. 82-89.
39. X. Zhao, S. Guo, F. Zhang, et al. MDASCA: An Enhanced Algebraic Side-Channel Attack for Error Tolerance and New Leakage Model Exploitation. In COSADE 2012, LNCS, vol. 7275, pp. 231-248, 2012.
40. X. Zhao, S. Guo, F. Zhang, et al. Algebraic Differential Fault Attacks on LED using a Single Fault Injection. Cryptology ePrint Archive, available at http://eprint.iacr.org/2012/347.pdf, 2012.