Algebraic Fault Analysis on GOST for Key Recovery and Reverse Engineering

Xinjie Zhao and Shize Guo
The Institute of North Electronic Equipment
Beijing, China
zhaoxinjieem@163.com, nsfgsz@126.com

Fan Zhang
Dept. Information Science & Electrical Engineering
Zhejiang University
Hangzhou, China
fanzhang@zju.edu.cn

Tao Wang
Dept. Information Engineering
Ordnance Engineering College
Shijiazhuang, China
twangdrsjz@yahoo.com.cn

Zhijie Shi and ChuJiao Ma
Dept. Computer Science and Engineering
University of Connecticut
Storrs, USA
{zshi,chum10010}@engr.uconn.edu

Dawu Gu
Dept. Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
dwgu@sjtu.edu.cn

***Abstract*—GOST is a well-known block cipher as the official encryption standard for the Russian Federation. A special feature of GOST is that its eight S-boxes can be secret. However, most of the researches on GOST assume that the design of these S-boxes is known. In this paper, the security of GOST against side-channel attacks is examined with algebraic fault analysis (AFA), which combines the algebraic cryptanalysis with the fault attack. Three AFAs on GOST, which have different attack goals in different scenarios, are investigated. The results show that 8 fault injections are required to recover the secret key when the full design of GOST is known, which is less than 64 fault injections required in previous work. 64 fault injections are required to recover the eight unknown S-boxes assuming the key is known. 270 fault injections are required to recover the key and the eight S-boxes when both are unknown. The results prove that AFA is very effective and keeping some components in a cipher secret cannot guarantee its security against fault attacks.**

***Keywords*-algebraic cryptanalysis; differential fault analysis; GOST; key recovery; reverse engineering.**

I. Introduction

Fault attacks [5] retrieve secret information in a crypto system by injecting computational faults and exploiting incorrect output. The most widely studied method of fault analysis is differential fault analysis (DFA) [4]. DFA derives information about the secret key by examining the differences between the correct and the faulty ciphertexts. Currently, DFA has been successfully applied to many block ciphers [23], [24], [29], [30], [35]. In DFA, adversaries knowing the cipher design need to manually compute the fault propagation path and patterns. Algebraic cryptanalysis [7] transforms ciphers into algebraic equations and recovers the secret key with automatic tools such as SAT solver [38]. Combining algebraic methods with fault attacks is a promising way to improve DFA. Such techniques are called "algebraic fault analysis" (AFA).

In 2010, Courtois et al. [9] first proposed AFA and applied it to DES. They showed that DES can be broken with a single fault injection if two bits in the 13th round are altered. In 2011, AFA [31] was used to improve DFA on the stream cipher Trivium [19], [20]. The inner state of Trivium can now be recovered using only two fault injections and 420 key stream bits. Recently, AFA [25], [42] was used to improve DFA on the lightweight block cipher LED [22], [24]. LED can be broken within three minutes on a PC with a single fault injection, this is more efficient

978-1-4799-6292-1/14 $31.00 © 2014 IEEE
DOI 10.1109/FDTC.2014.13

than standard DFAs [22], [24]. Moreover, the work in [42] showed that AFA can be used to evaluate the reduced key search space for a given fault model. In 2013, AFA [40] was used to improve DFA on Piccolo, DES and only one fault injection is required. From the findings so far, we can see that, as to some ciphers, AFA is more effective than DFA considering the data complexity. Meanwhile, AFA can leverage automatic tools and does not require the manual analysis in DFA, 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. Further investigations on AFA (e.g., AFA for reverse engineering) are important to understand the threats of fault attacks. This is also a motivation of this paper.

This paper studies AFA on the GOST block cipher [32], which is a Soviet and Russian government standard. Its large key size of 256 bits makes it a plausible alternative to AES-256 and Triple-DES. In 2010, GOST was submitted to ISO to become an international standard [16]. Assuming that the S-boxes are public, a lot of work has been done in recent years to analyze the security of GOST with the traditional cryptanalysis techniques [10], [12], [13], [15], [21], [26]. Recently, the security of GOST against fault attacks was studied with standard DFA in 2013 [27]. Based on a random nibble fault model in the right input register of round 25,27,29,31, the work in [27] can recover the key of GOST with 64 fault injections (16 fault injections for each round). The general design of GOST was published in 1994, but some of the crucial elements (such as the choice of eight S-boxes) remained confidential even today. These secret S-boxes can be considered as a secondary key, thus extending the key size to more than 700 bits.

In this paper, we study AFAs on GOST in three scenarios with different assumptions in order to recover the secret key, the contents of the eight S-boxes, or both. Based on a random byte fault model in the right input register of round 24, 26, 28, 30, 8 fault injections (2 faults for each fault position) are required to recover the secret key when the full design of GOST is known, which is less than 64 fault injections required in [27]. The results show that AFA on GOST can succeed under deep fault model and requires smaller numbers of fault injections than DFA on GOST. Based on a random byte fault model in the right input register of round 30, 31, 64 fault injections (32 faults for each fault position) are required to recover the eight unknown S-boxes assuming the key is known. Based on a random byte fault model in the right input register of round 23 to 31, 270 fault injections (30 faults for each fault position) are required to recover the key and the eight S-boxes when both are unknown. For the first time, we show that both the secret key and the eight secret S-boxes of GOST can be recovered by AFA at the same time in one attack.

The rest of this paper is organized as follows. Section II briefly describes the general design of GOST. Section III describes the targeted attack scenarios in this paper. Section IV presents the attack framework and fault models of AFA on GOST and Section V describes the detailed attack procedure. algebraic Section VI provides the experimental results and Section VII concludes the paper.

II. The GOST Block Cipher

This section briefly describes the GOST block cipher. The notations used in this paper are listed as follows.

- L_i: the left half of the data block in the i-th round ($1 \leq i \leq 32$).
- R_i: the right half of the data block in the i-th round($1 \leq i \leq 32$).
- K_i: the round key in the i-th round ($1 \leq i \leq 32$).
- $\boxplus$: addition modulo 2^{32}.
- $<<< n$: left rotation by n bits.

A. Encryption

The general design of GOST [32] was published in 1994, but even today the design of the eight S-boxes remained confidential. One set of S-boxes was published in 1994 as a part of the Russian standard hash function specification GOST R 34.11-94 and used by the Central Bank of the Russian Federation. This specific version of GOST 28147-89 block cipher is the most popular one, and it is commonly called just "the GOST cipher" in the cryptographic literature.

The overall design of GOST is similar to that of DES, which is also a Feistel network using 64-bit blocks. In each round (Fig. 1), the cipher processes the right half of the data block using the function f, XORs the result from f with the left half, and swaps the two halves.

$$
\begin{aligned}
L_{i+1} &= R_i \\
R_{i+1} &= L_i \oplus f(R_i, K_i)
\end{aligned} \tag{1}
$$

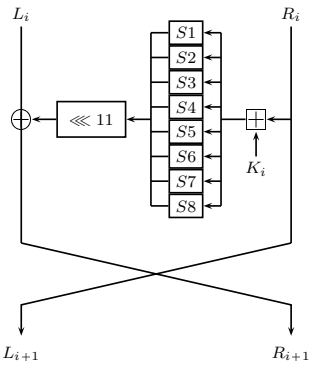


Figure 1. One round of GOST

The f function in GOST consists of three steps.

1) adding round key (AK): the right half of the block, R_i, is added (modulo 2^{32}) with the round key K_i.
2) S-box look-up (SL): the 32-bit output of AK goes through eight different 4×4 S-boxes.
3) rotating bits to the left (RL): the output of SL is rotated to the left by 11 bits.

B. Key Scheduling

The full GOST has 32 rounds and uses 256-bit keys. The key schedule is extremely simple. First, the 256-bit key is divided into eight 32-bit words $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8)$. GOST then uses one of these words as a round key. In the first 24 rounds, the words are used sequentially and repeated three times. In the final 8 rounds (25-32), the words are used in the reverse order.

$$
\begin{aligned}
K_i &= k_{(i-1) \ mod \ 8 \ +1}, \ i \in [1, 24] \\
K_i &= k_{32-i+1}, \ i \in [25, 32]
\end{aligned} \tag{2}
$$

III. Targeted Attack Scenarios

Three attack scenarios are investigated in this paper. The attacks have different goals because of different assumptions.

A. Scenario 1

The complete design of GOST is known and the goal is to recover the 256-bit secret key. In particular, we conduct AFA on GOST with eight S-boxes used by the Central Bank of the Russian Federation.

B. Scenario 2

The general design of GOST is known, but the design of the eight S-boxes is kept secret. The secret key is known to the adversaries and the goal is to recover the secret S-boxes, which is the same as the widely used assumptions in side-channel based reverse engineering [6], [14], [18], [33] and fault based reverse engineering [34].

C. Scenario 3

Both the secret key and the eight S-boxes are unknown. The goal is to recover both of them with AFA, which has not been studied in previous fault attacks.

IV. Overview of Algebraic Fault Analysis (AFA) on GOST

A. Framework

The framework of AFA on GOST is shown in Fig. 2. An AFA attack consists of three steps.

1. Fault injection. Suppose P_m and C_m denote the m-th correct plaintext and ciphertext encrypted with the secret key K, respectively. T_m denotes the correct value of the targeted intermediate state in fault attacks. When encrypting P_m with K, the adversary injects a fault into T_m by changing the power supply voltage , changing the frequency of the external clock, or using laser attack [1], [2], [3]. Then, T_m becomes the faulty value T_m^* and the resulting faulty ciphertext is C_m^*. Note that the correct ciphertext C_m is also available to the adversary. In this paper, the *position* refers to the round where faults are injected and the *location* refers to the location of the faulty byte in the targeted state. The position, location, and value (differences of T_m and T_m^*) of faults depend on the fault model, which will be described in Section V.

2. Fault exploitation. It consists of three substeps.

Substep1: The operations from the faulty state to the ciphertext for both the correct and faulty encryptions are represented with a system of equations. How to represent the nonlinear components of GOST, such as addition modulo 2^{32} and S-box lookup, is the most crucial part in this substep.

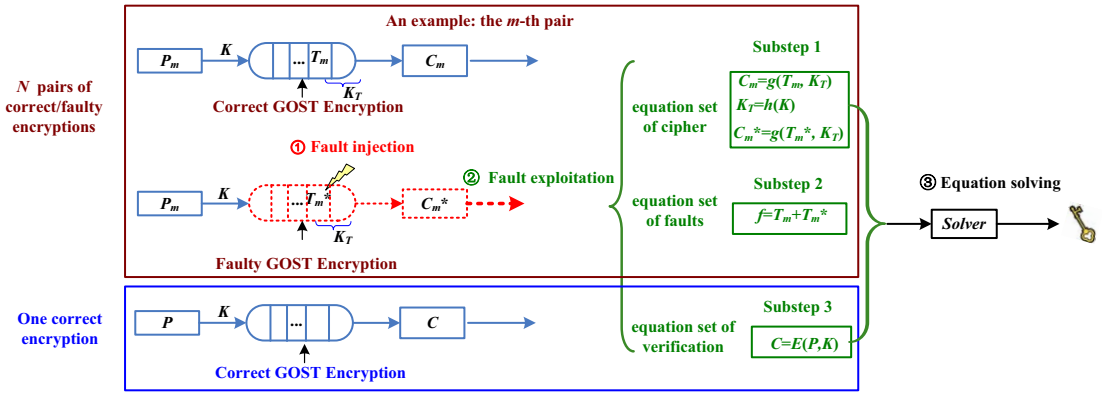


Figure 2. Framework of AFA on GOST

Substep2: Additional algebraic equations can be built to represent the difference of the correct and the faulty intermediate states. In practice, the location and value of the injected faults might not be known. This representation is very important for AFA.

Substep3: A set of new equations that describes a full encryption of GOST given a pair of known plaintext and ciphertext is built to verify the correctness of the recovered key.

Note that the key techniques in Substep1 and Substep3 are identical: representing a GOST round with algebraic equations, which will be discussed in detail in Section V. The difference between the two substeps is the number of rounds that are considered.

3. Equation solving. Finally, equations representing the cipher and the faults are combined into a larger equation system, which can be solved for secret variables by an equation solving tool (e.g., SAT solver). Breaking the cipher is equivalent to solving the equations.

B. Fault Model

Let R_i^j and R_i^{j*} denote the j-th correct and faulty bytes, respectively, in the right half of the data block in the i-th round ($1 \leq i \leq 32, 1 \leq j \leq 4$). $\triangle R_i^j$ denotes the fault difference of R_i^j and R_i^{j*}. Our proposed fault model includes the following assumptions.

1) The attacker has the capability to choose one plaintext P_m to encrypt with an unknown/given secret key K and to obtain the correct ciphertext C_m.
2) There is one random fault in byte R_i^j modifying all later operations depending on this R_i^j when encrypting the same P_m with K, which can be generated by injecting one byte fault into R_i^j or L_{i-1}^j. The attacker can obtain the faulty ciphertext C_m^*. It is called a *fault injection*.
3) The attacker can repeat the *fault injection* many times when encrypting N different plaintexts with the fixed key K. N depends on the attack and can be used to evaluate the data complexity of the attacks.
4) In a *fault injection*, fault is only injected into one round. The fault position i (the round number) is known, but both the fault location (j) and the fault difference ($\triangle R_i^j$) are not known.

V. Procedure of AFA on GOST

A. Converting GOST into Algebraic Equations

In this section, we first describe how to represent the nonlinear functions ($\boxplus$ and S-box) and the linear function (left rotation), then give the equation set for partial and full GOST.

1) Representing AK: Suppose $X = (x_1, x_2, ..., x_{32})$ and $Y = (y_1, y_2, ..., y_{32})$ are the two 32-bit input of the $\boxplus$ function. $Z = (z_1, z_2, ..., z_{32})$ is the output of the $\boxplus$ function and $Z = X \boxplus Y$. We employ the techniques in [11] to represent Z, as shown in Eq.(3). Only 31 extra binary variables t_i ($1 \leq i \leq 31$) are introduced for the carry bits in $\boxplus$.

$$
\begin{aligned}
z_{32} &= x_{32} + y_{32} \\
t_{31} &= x_{32}y_{32} \\
z_{31} &= x_{31} + y_{31} + t_{31} \\
t_{30} &= x_{31}y_{31} + x_{31}t_{31} + y_{31}t_{31} \\
z_{30} &= x_{30} + y_{30} + t_{30} \\
t_{29} &= x_{30}y_{30} + x_{30}t_{30} + y_{30}t_{30} \\
z_{29} &= x_{29} + y_{29} + t_{29} \\
t_{28} &= x_{29}y_{29} + x_{29}t_{29} + y_{29}t_{29} \\
&\dots \\
z_2 &= x_2 + y_2 + t_2 \\
t_1 &= x_2y_2 + x_2t_2 + y_2t_2 \\
z_1 &= x_1 + y_1 + t_1
\end{aligned} \tag{3}
$$

2) Representing SL: Suppose (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) are the input and output of a 4-bit S-box. We build the representation of the S-box in two different cases.

Case 1: The S-box is public. We employ the techniques in [28] to represent every S-box output bit with high-degree equations from the four S-box input bits. Four equations can be generated for one S-box of GOST. For other techniques to representing one S-box with many more cubics than four equations, please refer to [8]. Take the S-box $S1$ in the Central Bank of the Russian Federation [32] as an example. $S1$ can be denoted as:

$$
\begin{aligned}
y_1 &= x_2 + x_3 + x_4 + x_1x_2 + x_1x_3 + \\
&\quad x_2x_4 + x_1x_2x_4 + x_2x_3x_4 \\
y_2 &= 1 + x_3 + x_4 + x_3x_4 + x_1x_2x_3 + \\
&\quad x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 \\
y_3 &= x_1 + x_4 + x_1x_3 + x_1x_4 + x_2x_4 + \\
&\quad x_1x_2x_4 + x_2x_3x_4 \\
y_4 &= x_2 + x_3 + x_1x_4 + x_2x_4 + x_3x_4 + \\
&\quad x_1x_2x_3 + x_1x_3x_4
\end{aligned} \tag{4}
$$

Case 2: The S-box is secret. $S1$ can be denoted as in Eq.(5). 64 extra binary variables a_i $(1 \leq i \leq 64)$ are introduced as the secret parameters for each S-box.

$$
\begin{aligned}
y_1 &= a_1 + a_2x_1 + a_3x_2 + a_4x_3 + a_5x_4 + \\
&\quad a_6x_1x_2 + a_7x_1x_3 + a_8x_1x_4 + \\
&\quad a_9x_2x_3 + a_{10}x_2x_4 + a_{11}x_3x_4 + \\
&\quad a_{12}x_1x_2x_3 + a_{13}x_1x_2x_4 + a_{14}x_1x_3x_4 + \\
&\quad a_{15}x_2x_3x_4 + a_{16}x_1x_2x_3x_4 \\
y_2 &= a_{17} + a_{18}x_1 + a_{19}x_2 + a_{20}x_3 + a_{21}x_4 + \\
&\quad a_{22}x_1x_2 + a_{23}x_1x_3 + a_{24}x_1x_4 + \\
&\quad a_{25}x_2x_3 + a_{26}x_2x_4 + a_{27}x_3x_4 + \\
&\quad a_{28}x_1x_2x_3 + a_{29}x_1x_2x_4 + a_{30}x_1x_3x_4 + \\
&\quad a_{31}x_2x_3x_4 + a_{32}x_1x_2x_3x_4 \\
y_3 &= a_{33} + a_{34}x_1 + a_{35}x_2 + a_{36}x_3 + a_{37}x_4 + \\
&\quad a_{38}x_1x_2 + a_{39}x_1x_3 + a_{40}x_1x_4 + \\
&\quad a_{41}x_2x_3 + a_{42}x_2x_4 + a_{43}x_3x_4 + \\
&\quad a_{44}x_1x_2x_3 + a_{45}x_1x_2x_4 + a_{46}x_1x_3x_4 + \\
&\quad a_{47}x_2x_3x_4 + a_{48}x_1x_2x_3x_4 \\
y_4 &= a_{49} + a_{50}x_1 + a_{51}x_2 + a_{52}x_3 + a_{53}x_4 + \\
&\quad a_{54}x_1x_2 + a_{55}x_1x_3 + a_{56}x_1x_4 + \\
&\quad a_{57}x_2x_3 + a_{58}x_2x_4 + a_{59}x_3x_4 + \\
&\quad a_{60}x_1x_2x_3 + a_{61}x_1x_2x_4 + a_{62}x_1x_3x_4 + \\
&\quad a_{63}x_2x_3x_4 + a_{64}x_1x_2x_3x_4
\end{aligned} \tag{5}
$$

3) Representing RL: Suppose $X = (x_1, x_2, ..., x_{32})$ and $Y = (y_1, y_2, ..., y_{32})$ are the 32-bit input and output of the left rotation function in GOST. y_i can be denoted as

$$
y_i = x_{((i+9) \ mod \ 32)+1} \tag{6}
$$

4) Building the equation set of GOST: Combining the representation of AK, SL, RL, the equation sets for partial GOST can be built to represent the last few rounds of the correct and faulty encryptions starting from the position of fault injection. Similarly, the equation set for full GOST can be built from the known plaintext and ciphertext to verify the recovered key. Since fault attacks start with analyzing the ciphertext, we will first build the equation set for GOST decryption. This can accelerate the speed of solving the algebraic equations, which is also noted in [42]. Building the equation set for r decryption rounds $(1 \leq r \leq 32)$ of GOST is illustrated in Algorithm 1.

Algorithm 1. Building the equation set for r rounds decryption of GOST

1: $C \longleftarrow [c_1, c_2, ..., c_{64}]$
2: $L_{33} \longleftarrow [c_1, c_2, ..., c_{32}]$
3: $R_{33} \longleftarrow [c_{33}, c_{34}, ..., c_{64}]$
4: $L_{32} \longleftarrow L_{33} \oplus RL(SL(RL(R_{33}, K_{32})))$
5: $R_{32} \longleftarrow R_{33}$
6: **for** i =31 to $32 - r$ $(i > 0)$ **do**
7: $L_i \longleftarrow R_{i+1}$
8: $R_i \longleftarrow L_{i+1} \oplus RL(SL(RL(R_{i+1}, K_i)))$
9:**end for**

Using Algorithm 1, each round of GOST can be represented with 429 variables and 777 CNF equations when the S-boxes are known, 949 variables and 2353 CNF equations when the S-boxes are unknown.

B. Converting Faults into Algebraic Equations

When a fault is injected into R_i, the standard DFAs [23], [24], [29], [30], [35] have to deduce the accurate location of the faulty byte in R_i by analyzing the fault propagation pattern. And all previous AFAs [9], [25], [42] assumed the fault location is known. Inspired by the work in [41], this section describes a new method to convert faults into algebraic equations when both the fault location and fault value are unknown.

Suppose the correct input to the right register of the i-th round of GOST is denoted as $X = (x_1, x_2, \ldots, x_{32})$. The faulty input after the fault injection is denoted as $Y = (y_1, y_2, \ldots, y_{15})$. Let Z denote the fault difference of X and Y. Z can be computed by

$$Z = (z_1, z_2, \ldots, z_{32}), z_i = x_i \oplus y_i, 1 \leq i \leq 32 \quad (7)$$

Z can be considered as the concatenation of four bytes $Z_1||Z_2||Z_3||Z_4$, where $Z_i = (z_{8i-7}, z_{8i-6}, \ldots, z_{8i})$ $(1 \leq i \leq 4)$. Four one-bit variables u_i are introduced to represent whether Z_i is faulty or not.

$$u_i = (1 \oplus z_{8i-7}) \wedge (1 \oplus z_{8i-6}) \wedge (\ldots) \wedge (1 \oplus z_{8i}) \quad (8)$$

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

$$\begin{aligned} &(1 - u_1) \vee (1 - u_2) \vee (1 - u_3) \vee (1 - u_4) = 1, \\ &u_i \vee u_j = 1, \quad 1 \leq i < j \leq 4 \end{aligned} \quad (9)$$

The injected fault can be represented with the Eq(7)(8)(9) whose representations are quite simple and straightforward. Note that the technique of this section is very important for fault attacks with multiple fault injections when the accurate fault location is difficult to be deduced from the ciphertext differences.

C. Solving the Equations

The problem of searching for the master key is now transformed into solving the merged equation system. Many automatic tools, such as Gröbner basis-based [17] and SAT-based solver [38], can be leveraged. We choose CryptoMiniSAT [38] as the solver in our analysis. The readers can refer to [37] for details of how to generate equations and how to feed them to the solver.

VI. Experiment results

To prove the feasibility of AFA on GOST, we conducted extensive experiments in all three scenarios described in Section III.

A. Experiment Setup

How to inject faults has been discussed in [1], [2], [3] and is out of the scope of this paper. In our experiment, we simulated the fault injection with computer software and used CryptoMiniSat v2.9.4 [38] to solve the combined algebraic equations. The PC that runs CryptoMiniSAT has the following configuration: Intel Core I7-2640M, 2.80 GHZ, and 4G bytes memory. The operating system is 64-bit Windows 7. In this section, an *instance* refers to one run of our AFA on a set of correct plaintexts, correct and faulty ciphertexts and a fixed key.

In the attack, several different parameters are considered, which is partially referred from [39].

- N: the number of fault injections, which is used to evaluate the data complexity of AFA on GOST.
- $V(N)$: the number of variables in representing the equations of GOST and N faults.
- $A(N)$: the number of ANF equations in representing GOST and N faults.
- $\upsilon(N)$: the size of the generated scripts in representing the equations of GOST and N faults.
- $t(N)$: the time complexity (seconds) required in solving the algebraic equations of AFA for a given value of N.

Table I
$N = 8$ PAIRS OF CORRECT AND FAULTY CIPHERTEXTS

m	P_m	C_m	C_m^*	i
1	0x287d14432765f03d	0x25ce9927f052a780	0x619989d703b98b08	24
2	0x8b100cea6ce4ded1	0x9a4a57d15eb00228	0x71aae71e4822279a	24
3	0x7efda7963d01c081	0xff240d1e54c7f45b	0x2d5f4ea945cb0615	26
4	0xdaba6447681dc833	0x17e7d15d13393b0b	0x99d864f232a2b556	26
5	0xff1dd1c07d6050d6	0x8af39ce68950b222	0x7af69ce2f968b202	28
6	0x9c09c3f2bc9bfe26	0x58cee80c516767b7	0x7d720e6389185d9f	28
7	0x520e7f84154eccab	0xfbaf17fd782e58f6	0xcb5be7ab784842de	30
8	0x65b3a38cbbf5fe79	0x024fe23051aef23f	0x5a3c44b0f1ae9608	30

- τ: the threshold of the time complexity (seconds) required in a successful AFA.
- $\phi(N, \tau)$: the success rate of the key recovery in AFA for a given value of N and τ.
- $\lambda(N)$: the entropy of the secret key in AFA on GOST for a given value of N under Scenario 1. In this scenario, the plaintexts are unknown and the ciphertexts are known.

B. Results of AFA on GOST in Scenario 1

In this scenario, a fault is injected in a random byte in R_i, $i \in \{24, 26, 28, 30\}$, when N random plaintexts are encrypted with a fixed secret key. The position of the injected fault in this section is more deep than the work in [27] (random nibble fault injected in the right input register of round 25,27,29,31). We believe that under our deep fault model, more subkey are involved in the fault propagation path per fault injection and the full master key can be recovered with less number of fault injections. The algebraic equations for the last $33-i$ rounds of the correct and faulty encryptions, and for one full correct encryption are built as described in Section V.A. It should be noted that the S-boxes are known in this scenario and the equations are built as the Case 1 in Section V.A. The differences of R_i and R_i^* are converted into algebraic equations with the method in Section V.B.

In the attack, we assume $4n$ random faults are injected into R_i, $i \in \{24, 26, 28, 30\}$ of GOST (n faults for each value of i, $N = 4n$). Table I lists 8 pairs of correct and faulty ciphertexts encrypted with key $K =$ 0xd4d0ed83f9e0580bf473e25ddda03fb93623ff14144fe3cb3dfd6f6c0ddee894. A correct plaintext/ciphertext pair used for verification in this Table is $P =$ 0xa11d39c1d41da871 and $C =$ 0xee349c1fcdf71607.

In the solution file generated by CryptoMiniSAT, variables No. 2 to No. 257 are the 256-bit secret key of GOST. A key bit is 1 if the corresponding index is positive, and is 0 otherwise. The recovered key in AFA is the same as the correct key K. We conducted attacks on GOST with different values of n, $n = 1, 2, 3, 4$ ($N = 4, 8, 12, 16$). For each case, we generated different random plaintext/key and ran 100 instances.

In the attack, when N=16, $V(N) = 52,801$, $A(N) = 102,385$, $\upsilon(N) = 1999$KB, 100% of the instances can be solved within one minute, $\phi(16, 60) = 100\%$. When N=12, $V(N) = 42,857$, $A(N) = 83,037$, $\upsilon(N) = 1613$KB, $\phi(12, 600) = 80\%$, $\phi(12, 3600) = 95\%$, $\phi(12, 7200) = 100\%$, which means that 80%,95%,100% of the instances can be solved within ten minutes, one hour and two hours respectively. When N=8, $V(N) = 32,913$, $A(N) = 63,689$, $\upsilon(N) = 1226$KB, $\phi(8, 3600) = 85\%$, $\phi(8, 7200) = 95\%$, $\phi(8, 14400) = 100\%$. When N=4, $V(N) = 22,969$, $A(N) = 44,341$, $\upsilon(N) = 840$KB, all the instances can not be solved within 48 hours. The results show that AFA on GOST can succeed under deep fault model and requires smaller numbers of fault injections than DFA on GOST in [27].

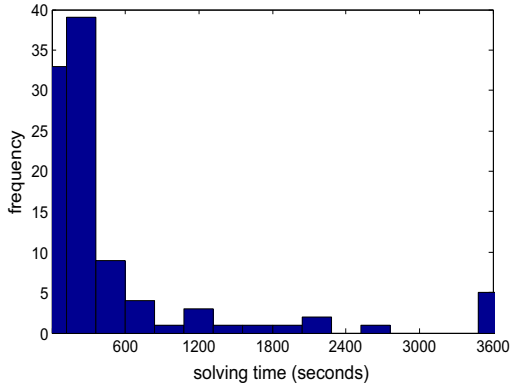
The distribution of equation solving time of N=8 and N=12 is shown in Fig. 3. we can see that the equation solving time of CryptoMiniSAT seems to follow an exponential distribution (as noted in [36], [41], [42]).

Since v2.9.4, the CryptoMiniSAT solver can support multiple solutions outputting. We are interested in using this feature to evaluate the reduced search space of the secret key in AFA. Under this scenario, the value of the plaintext is not required to fed into the solver.

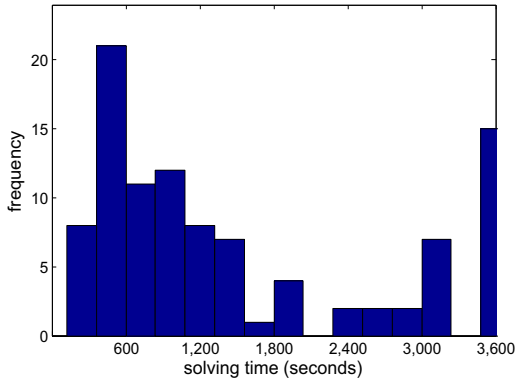
We conducted 100 random instances of AFA on GOST considering four cases, $N = 4, 8, 12, 16$. For

Table II
64-BIT SECRET PARAMETERS FOR THE EIGHT RECOVERED S-BOXES OF GOST (IN HEXADECIMAL)

S-box	$a_1, a_2, \ldots, a_{64}$	S-box	$a_1, a_2, \ldots, a_{64}$
$S1$	0x3e4a983e4b4a3174	$S5$	0x0ab8873cec12349e
$S2$	0xf478c97494c208a6	$S6$	0x1dceda3679486e34
$S3$	0x6986bf52669eec3c	$S7$	0xf5c8eb982aead2b2
$S4$	0x5802b282ac52f22e	$S8$	0x5c4a3b560eba85b6



(a) N=12, $V(N) = 42,857$, $A(N) = 83,037$, $\upsilon(N) =$ 1613KB



(b) N=8, $V(N) = 32,913$, $A(N) = 63,689$, $\upsilon(N) =$ 1226KB

Figure 3. Equation solving time in AFA on GOST in Scenario 1

$N = 4$, the solver can not output all the solutions within 48 hours. For $N = 8, 12, 16$, the average $\lambda(N)$ can be calculated easily and $\lambda(16) = 0.2$, $\lambda(12) = 12.2$, $\lambda(8) = 16.7$, which means that key search space of GOST can be reduced to $2^{0.2}$,$2^{12.2}$,$2^{16.7}$ after analyzing 16, 12, 8 fault injections, respectively. The reduced entropy of the secret key is larger with the decrease of N. For large value of $\lambda(N)$ (e.g, $N = 8$), if the plaintext are fed into the solver (as shown in above), the solver has to verify the correctness of the $2^{\lambda(N)}$ possible solutions, thus the solving time is much longer.

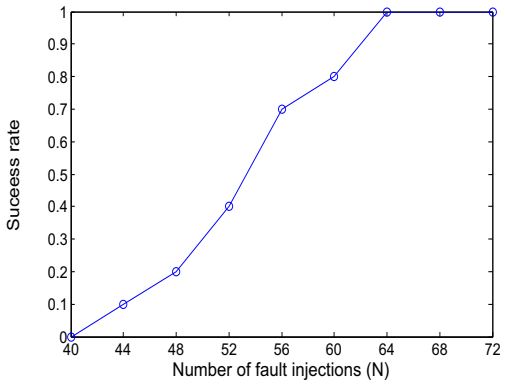
C. Results of AFA on GOST in Scenario 2

In this scenario, a fault is injected in a random byte in R_i, $i \in \{30, 31\}$, when N random plaintexts are encrypted with a fixed secret key. The method of building the algebraic equations are almost the same as in Section VI.B except that the S-boxes are unknown in this scenario and the S-box equations are built as in Case 2 in Section V.A. Recall that the 256-bit key is known and the goal is to recover the 512-bit in the secret S-boxes.

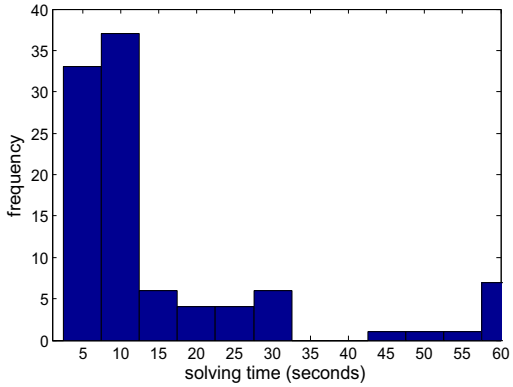
We firstly implemented GOST with the eight S-boxes used by the Central Bank [32], but assumed that the contents of these S-boxes are unknown to us. The experiment results show that if 32 random faults are injected into R_{30} and R_{31} (64 faults in total, $V(N) = 183,553$, $A(N) = 467,969$, $\upsilon(N) = 9024$KB), the 512-bit secret parameters for the eight S-boxes can be recovered with on average one minute. We can see that the number of variables and ANF equations required in this scenario is much bigger than that in scenario 2. This is mainly caused by the difficulty in representing unknown S-boxes. Table II lists the output from CryptoMiniSAT, which shows the hexadecimal representation of the coefficients in Eq. (5) for eight S-boxes.

For each S-box, we substitute the values of a_1 to a_{64} in Table II into Eq.(5). As to the first S-box $S1$, we can obtain equations that are the same as Eq.(4). Then we substitute the values of the S-box input to the extracted four equations in Eq.(4) and recover the S-box. The recovered eight S-boxes are the same as the released S-boxes in [32].

To show the easy and hard cases of our AFA, we conducted attacks under different number of faults. For different values of N, we run 100 random instances and calculate the solving time, the success rate. Fig. 4(a)



(a) Success rate (different N)



(b) Solving time (N=64, $V(N) = 183,553$, $A(N) = 467,969$, $\upsilon(N) = 9024$KB)

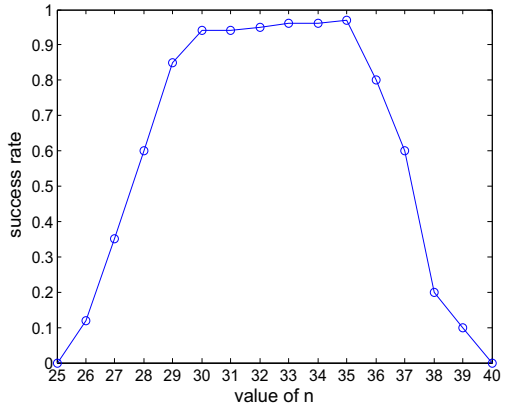
Figure 4. Results of AFA on GOST in Scenario 2

shows the success rate for different N values when the time for solving equations is limited to under one hour. $\phi(N, \tau)$ increase with the value of N. When N=64, $V(N) = 183,553$, $A(N) = 467,969$, $\upsilon(N) = 9024$KB, $\phi(64, 3600) = 100\%$. The time complexity of AFA on GOST with N=64 is shown in Fig. 4(b). We can see that 93% of the instances can be solved by CryptoMiniSAT within one minute.

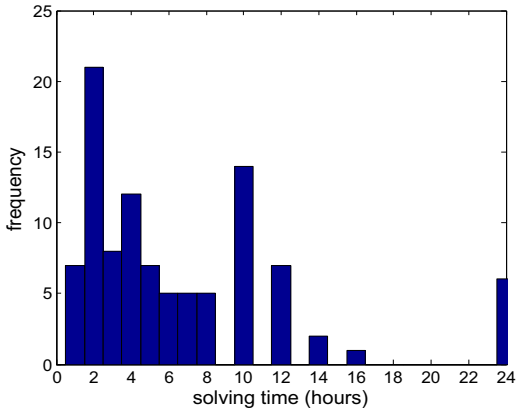
Besides the eight S-boxes used by the Central Bank [32], we also performed several attacks on GOST implemented with other groups of eight unknown S-boxes. Both the data and time complexities required in the attacks are close, which shows the strong applicability of the AFA technique.

D. Results of AFA on GOST in Scenario 3

In this scenario, we assume $9n$ random faults are injected into R_i, $i \in \{23, 24, 25, 26, 27, 28, 29, 30, 31\}$ of GOST (n faults for each value of i) when N random plaintexts are encrypted with a fixed secret key, $N = 9n$. The method of building the algebraic equations is almost the same as in Section VI.C except that the 256-bit key is unknown. The goal is to recover the 512-bit for the eight S-boxes and the 256-bit key (768-bit in total).



(a) Success rate $\phi(N, \tau)$, $N = 9n$



(b) Solving time (N=270, $V(N) = 1,543,797$, $A(N) = 3,974,183$, $\upsilon(N) = 83,700$KB)

Figure 5. Results of AFA on GOST in Scenario 3

We also conducted attacks under different number of faults. We set τ=86400 seconds, which is equal to 24 hours. For different values of N, we run 100 random instances and calculate the solving time, the success rate. Fig. 5(a) shows the success rate for different N. When $n \leq 25$, all of the attacks failed and the success rate $\phi(N, \tau)$ is 0%. When $25 < n \leq 35$, $\phi(N, \tau)$ is increased with n, and the average solving time is decreased. When $n > 35$, the CryptoMiniSAT solver might output exceptions and the $\phi(N, \tau)$ is even

decreased. This maybe caused by the problem of large memory requirement in SAT solver. When $n > 35$, the script size of the full equation set is larger than 100 MB (including more than five million ANF equations and two million variables). The time complexity of AFA on GOST with n=30 is shown in Fig. 5(b). The average solving time when n=30 is 6.39 hours, which is larger than 3.40 hours when n=35.

VII. Conclusion

This paper evaluates the security of GOST against the algebraic fault analysis (AFA). We propose a technique to represent the secret S-boxes and the injected faults with algebraic equations. The attacks are studied in three scenarios with different conditions and goals. The results show that AFA can be used for reverse engineering and keeping some components in a cipher secret cannot guarantee its security. Both the key and the S-boxes can be recovered by AFA with small numbers of faults. To the best of our knowledge, this is the first fault attacks that can recover both the secret key and the secret S-boxes at the same time. Future work on the proposed AFA includes evaluating the complexity of AFA, optimizing the equation solving time, and investigating the attacks on real devices.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under the grants 60772082, 61173191, 61272491, 61202386, 61309021, the Major State Basic Research Development Program (973 Plan) of China under Grant 2013CB338004, the US National Science Foundation under the grant CNS-0644188.

References

[1] 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.

[2] 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.

[3] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault injection attacks on cryptographic devices: Theory, practice and countermeasures. Italy, Tech. Rep., 2012.

[4] E. Biham, A. Shamir. Differential Fault Analysis of Secret Key Cryptosystem. In CRYPTO 1997, LNCS, vol. 1294, pp. 513-525, 1997.

[5] 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.

[6] C. Clavier. An improved scare cryptanalysis against a secret a3/a8 gsm algorithm. In ICISS 2007, LNCS, vol. 4812, pp. 143-155, 2007.

[7] N. T. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Asiacrypt 2002, LNCS, vol. 2501, pp. 267-287, 2002.

[8] N. T. Courtois, G. V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In IMA 2011, LNCS, vol. 4887, pp. 152-169, 2007.

[9] N. T. Courtois, D. Ware, K. Jackson. Fault-Algebraic Attacks on Inner Rounds of DES. In eSmart 2010, pp. 22-24, 2010.

[10] N. T. Courtois. Algebraic Complexity Reduction and Cryptanalysis of GOST, In Cryptology ePrint Archive, Report 2011/626. March 2013, available at http://eprint.iacr.org/2011/626.pdf.

[11] N. T. Courtois, D. Hulme and T. Mourouzis. Solving Circuit Optimisation Problems in Cryptography and Cryptanalysis, In (informal) proceedings of SHARCS 2012 workshop, pp. 179-191, available at http://2012.sharcs.org/record.pdf, March 2012.

[12] N. Courtois. An Improved Diffrential Attack on Full GOST, In Cryptology ePrint Archive, Report 2012/138. March 2011, available at http://eprint.iacr.org/2012/138.pdf.

[13] N. Courtois. Security Evaluation of GOST 28147-89 in View of International Standardisation. Cryptologia, 2012, 36(1): 2-13.

[14] R. Daudigny, H. Ledig, F. Muller, and F. Valette. Scare of the des. In ACNS 2005, LNCS, vol. 3531, pp. 393-406, 2005.

[15] I. Dinur, O. Dunkelman, and A. Shamir. Improved Attacks on Full GOST. In FSE, LNCS, vol. 7549, pp. 9-28, 2012.

[16] V. Dolmatov. RFC 5830: GOST 28147-89 encryption, decryption and MAC algorithms, IETF (March 2010), ISSN: 2070-1721, available at http://tools.ietf.org/html/rfc5830.

[17] J.-C. Faug ère, Gröbner Bases. Applications in Cryptology. In Invited Talk of FSE 2007, available at http://fse2007.uni.lu/slides/faugere.pdf.

[18] S. Guilley, L. Sauvage, J. Micolod, et al. Defeating any secret cryptography with scare attacks. In LATINCRYPT 2010, LNCS, vol. 6212 pp. 273-293, 2010.

[19] M. Hojsik and B. Rudolf., B. Differential fault analysis of Trivium. In FSE 2008, LNCS, vol. 5086, pp. 58-172, 2008.

[20] M. Hojsik and B. Rudolf., B. Floating fault analysis of trivium. In INDOCRYPT 2008, pp. 239-250, 2008.

[21] T. Isobe: A Single-Key Attack on the Full GOST Block Cipher. Journal of Cryptology, February 2012: 1-18.

[22] 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.

[23] K. Jeong. Kitae Jeong. Differential Fault Analysis on Block Cipher Piccolo. Cryptology ePrint Archive, available at http://eprint.iacr.org/2012/399.pdf, 2012.

[24] P. Jovanovic, M. Kreuzer, and I. Polian. A Fault Attack on the LED Block Cipher. COSADE 2012, LNCS, vol. 7275, pp. 120-134, 2012.

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

[26] O. Kara and F. Karako. Fixed Points of Special Type and Cryptanalysis of Full GOST, In CANS 2012, 11th International Conference on Cryptology and Network Security, Darmstadt, Germany, December 12-14, 2012.

[27] J. Kim. On the security of the block cipher GOST suitable for the protection in U-business services. Personal and Ubiquitous Computing, vol. 17, No. 7, pp. 1429-1435, 2010.

[28] 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.

[29] W. Li, D. Gu, J. Li. Differential fault analysis on the ARIA algorithm. Information Sciences. 2008, 10(178): 3727-3737.

[30] W. Li, D. Gu. Differential fault analysis on the contracting UFN structure, with application to SMS4 and MacGuffin. Journal of Systems and Software, 2009, 82: 346-354.

[31] M. Mohamed, S. Bulygin and J. Buchmann. Improved Differential Fault Analysis of Trivium. In COSADE 2011, pp. 147-158, 2011.

[32] National Bureau of Standards. Federal Information Processing Standard- Cryptographic Protection - Cryptographic Algorithm. GOST 28147-89, 1989.

[33] R. Novak. Side-channel based reverse engineering of secret algorithms. In ERK 2003, pp. 445-448, 2003.

[34] M. S. Pedro, M. Soos, and S. Guilley. FIRE: Fault Injection for Reverse Engineering. In WISTP 2011, LNCS, Vol. 6633, pp. 280-293, 2011.

[35] 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.

[36] M. Renauld, F.-X. Standaert. Algebraic Side-Channel Attacks. In INSCRYPT 2009, LNCS, vol. 6151, pp. 393-410, 2009.

[37] SAT. SAT Race Competition. Available at http://www.satcompetition.org/.

[38] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In SAT 2009, LNCS, vol. 5584, pp. 244-257, 2009.

[39] F-X Standaert, T. G. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In EUROCRYPT 2009, LNCS, vol. 5479, pp. 443-461, 2009.

[40] F. Zhang, X.J. Zhao, S.Z. Guo, Tao Wang and Z.J. Shi. Improved Algebraic Fault Analysis: A Case Study on Piccolo and Applications to Other Lightweight Block Ciphers. In COSADE 2013, LNCS, vol. 7864, pp. 62-79, 2013.

[41] X. J. Zhao, F. Zhang , S. Z. Guo, 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.

[42] X. J. Zhao, S. Z. Guo, F. Zhang, et al. Improving and Evaluating Differential Fault Analysis on LEDwith Algebraic Techniques. In FDTC 2013, pp. 41-51, 2013.