Algebraic Fault Analysis on GOST for Key Recovery and Reverse Engineering

FDTC 2014
Fault Diagnosis and Tolerance in Cryptography

Xinjie Zhao, Shize Guo, Fan Zhang, Tao Wang, Zhijie Shi, Chujiao Ma and Dawu Gu

The Institute of North Electronic Equipment, Beijing, China
Ordnance Engineering College, Shijiazhuang , China
Zhengjiang University, Hangzhou, China
University of Connecticut, Storrs, USA
Shanghai Jiao Tong University , Shanghai , China

Outline

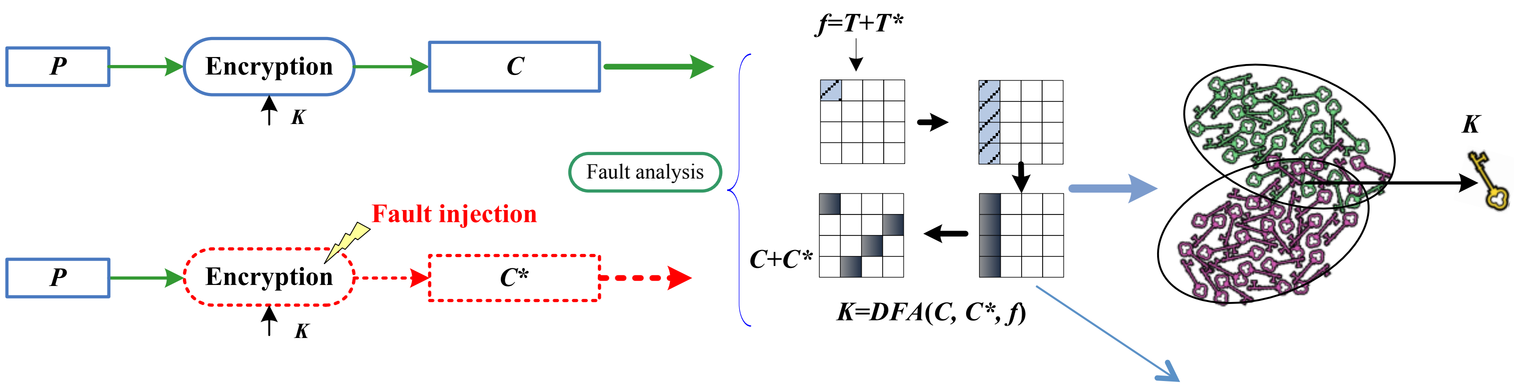
- **Motivation?** Algebraic Fault Analysis
- **Target?** GOST and Attack Scenarios
- **Technique?** AFA on GOST
- **Results?** Key Recovery and Reverse Engineering
- **Summary?** Conclusion of Our Work

Traditional Fault Analysis

FA (Fault Attack) first proposed by Boneh et al in 1996.

– Received faulty output, guess the fault, find the secret.

- DFA (Differential Fault Analysis) proposed by Biham and Shamir in 1997.
  – Used to break public-key ciphers (ECC), block ciphers (AES, ARIA, Camellia and CLEFIA) and stream ciphers (RC4, Trivium).

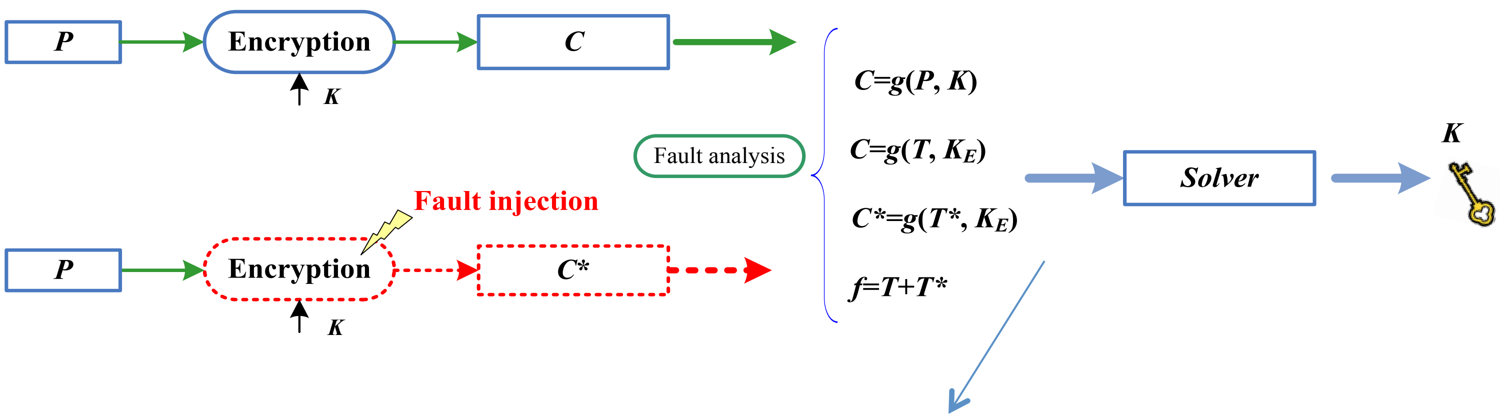


Framework of DFA

Manually fault analysis;
Maximal efficiency unknown？

Algebraic Fault Analysis

- AFA (Algebraic Fault Analysis) proposed by Courtois in 2010.
  - Algebraic cryptanalysis with fault attack.



Compared with DFA:

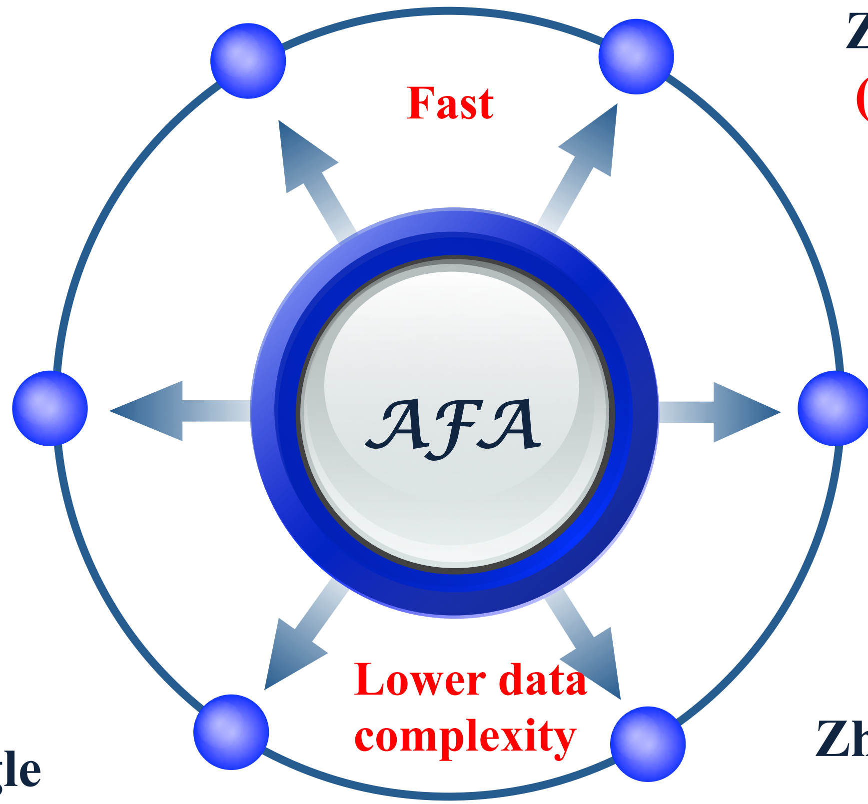
- Algebraic analysis are generic and automatic
- Solvers (automatic) allow easier and simpler analysis
- Fault information allows optimization

State-of-the-art AFA

eSmart 2010
Courtois: DES, single fault, $2^{17.35}$ hours

COSADE 2011
Mohamed: Trivium, less faults

ePrint 2012/400
Jovanovic: LED, single fault, 14.67 hours.



COSADE 2013
Zhang: Piccolo, DES (10 seconds), MIBS, single fault

FDTC 2013
Zhao: LED, single fault, 1-3 minutes, evaluating DFA

CACR 2013
Zhao: LBlock, single fault

Our Motivations?

- Current AFA
  - Key recovery when the design of cipher is known
  - Evaluating the reduced key search space of DFA
- Our work
  - Can AFA work when partial design of cipher is unknown?
  - Can AFA be used for reverse engineering besides key recovery?

Outline

- **Motivation?** Algebraic Fault Analysis
- **Target? GOST and Attack Scenarios**
- **Technique?** AFA on GOST
- **Results?** Key Recovery and Reverse Engineering
- **Summary?** Conclusion of Our Work

Overview of GOST

- A Soviet and Russian government standard symmetric key block cipher.
  - 64-bit block cipher
  - 256 bit keys
  - 32 rounds
  - Feistel structure
  - 8 S-Boxes
  - modulo 2^{32} nonlinear part
  - Simple key schedule

Overview of GOST

- processes the right half of the block using function f, XORs the result from f with the left half, and swaps the two halves.
- key schedule is simple, divide 256-bit key into 8 pieces, using one piece per round

the contents of 8 S-Boxes

might be secret

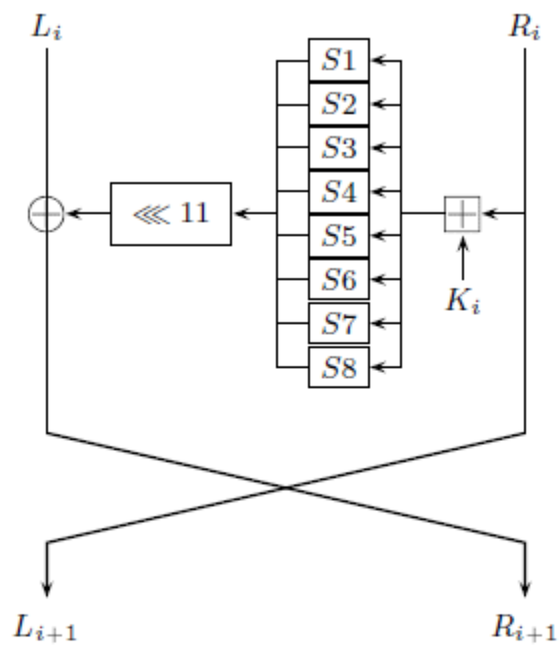


Figure 1. One round of GOST

Attack Scenarios

single byte fault injection on the right half of GOST

- **Scenario 1:** known complete GOST design, key recovery?

- **Scenario 2:** 8 S-Boxes secret, known secret key, AFA technique, reverse engineering of S-Boxes?

- **Scenario 3:** 8 S-Boxes secret, unknown secret key, AFA technique, both key recovery and reverse engineering?

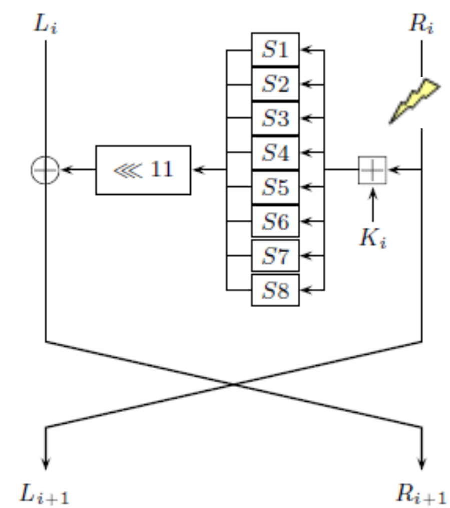


Figure 1. One round of GOST

Outline

- **Motivation?** Algebraic Fault Analysis
- **Target?** GOST and Attack Scenarios
- **Technique? AFA on GOST**
- **Results?** Key Recovery and Reverse Engineering
- **Summary?** Conclusion of Our Work

AFA on GOST

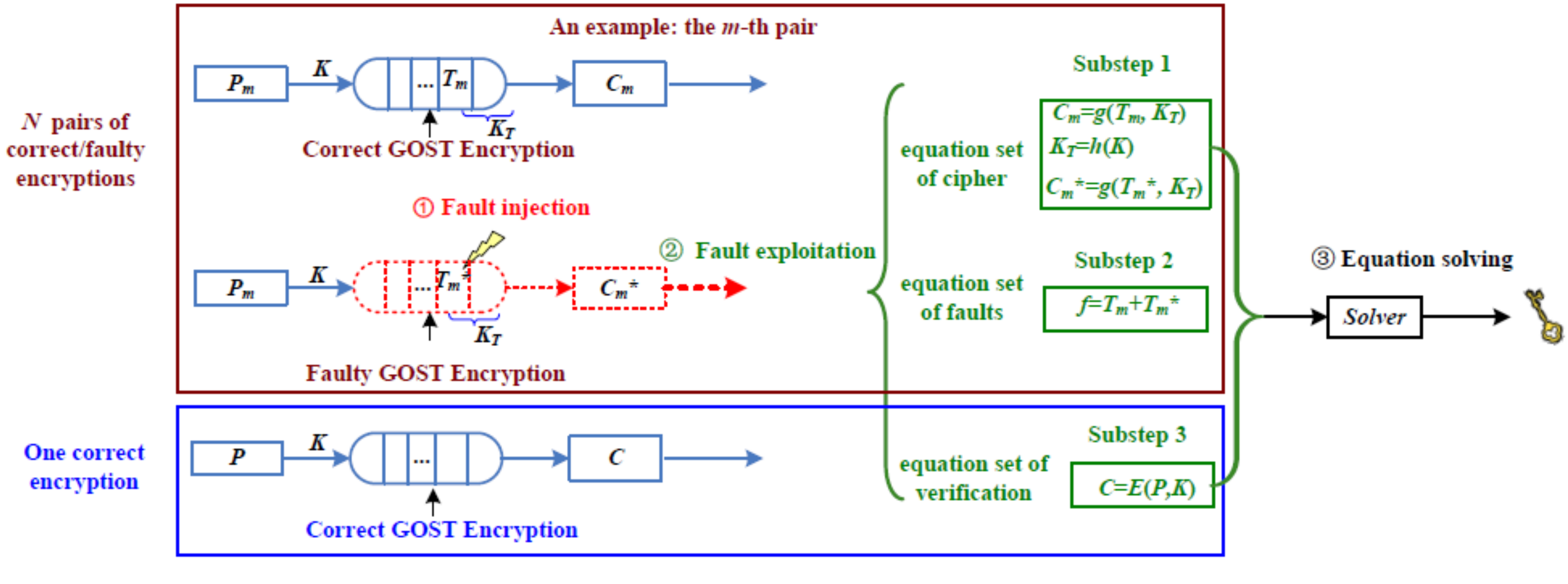


Figure 2. Framework of AFA on GOST

- one full correct GOST equation set
- the last few GOST rounds equation set since the fault injections for N pairs of correct and faulty encryptions

Step 1: GOST Equation Set

- Represent AK (Adding modulo 2^{32})

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

Step 1: GOST Equation Set

- Represent SL (S-Box lookup)

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

Public S-Box

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

Secret S-Box

64 variables a_i are introduced

Step 1: GOST Equation Set

- Represent RL (Rotating bits to left)

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

- Represent GOST decryption can accelerate speed of AFA)

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

```
C ⟵ [c_1, c_2, ..., c_64]
L_33 ⟵ [c_1, c_2, ..., c_32]
R_33 ⟵ [c_33, c_34, ..., c_64]
    L_32 ⟵ L_33 ⊕ RL(SL(RL(R_33, K_32)))
    R_32 ⟵ R_33
for i =31 to 32 − r (i > 0) do
    L_i ⟵ R_{i+1}
    R_i ⟵ L_{i+1} ⊕ RL(SL(RL(R_{i+1}, K_i)))
end for
```

Step 2: Fault Equation Set

- Suppose Z denote the injected fault difference
  - Z can be considered as the concatenation of four bytes

$$Z_1||Z_2||Z_3||Z_4, \quad Z_i = (z_{8i-7}, z_{8i-6}, \ldots, z_{8i})\ (1 \leq i \leq 4).$$

  - Four one-bit u_i are used to represent whether Z_i is faulty (u_i=0) or not

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

  - Only one byte fault is injected, only one u_i=0

$$(1 - u_1) \vee (1 - u_2) \vee (1 - u_3) \vee (1 - u_4) = 1,$$
$$u_i \vee u_j = 1, \quad 1 \leq i < j \leq 4$$

Step 3: Solver

- Combine the equation set of GOST with injected fault and use solver to recover the secret key.
- CryptoMiniSAT v2.9.4, support multiple solution output
- 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.

Outline

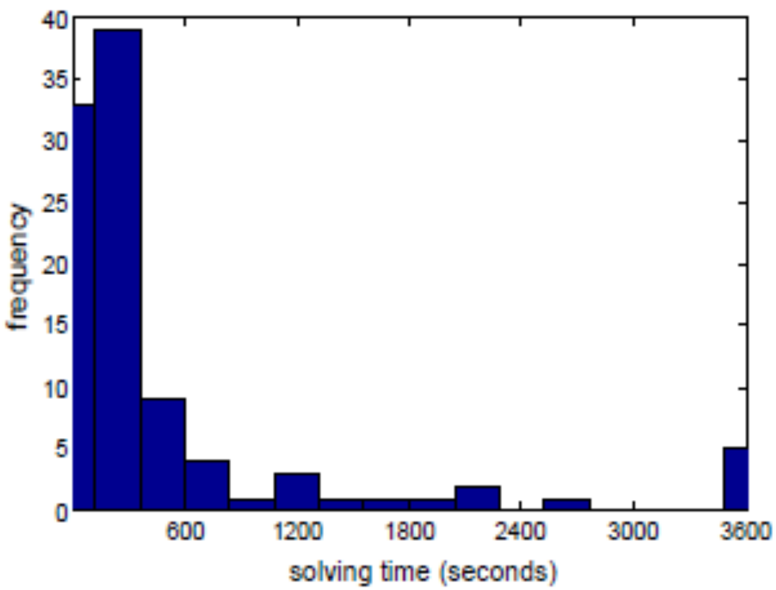
- **Motivation?** Algebraic Fault Analysis
- **Target?** GOST and Attack Scenarios
- **Technique?** AFA on GOST
- **Results?** Key Recovery and Reverse Engineering
- **Summary?** Conclusion of Our Work

Experiment Parameters

N	**the number of fault injections**
$V(N)$	the number of variables in equation set
$A(N)$	the number of ANF equations in equation set
$\upsilon(N)$	the size of the generated scripts
$t(N)$	the time complexity (seconds) required in solver
τ	threshold of the time complexity (seconds) in a successful AFA
$\varphi(N,\tau)$	the success rate
$\lambda(N)$	the entropy of the secret key in Scenario 1

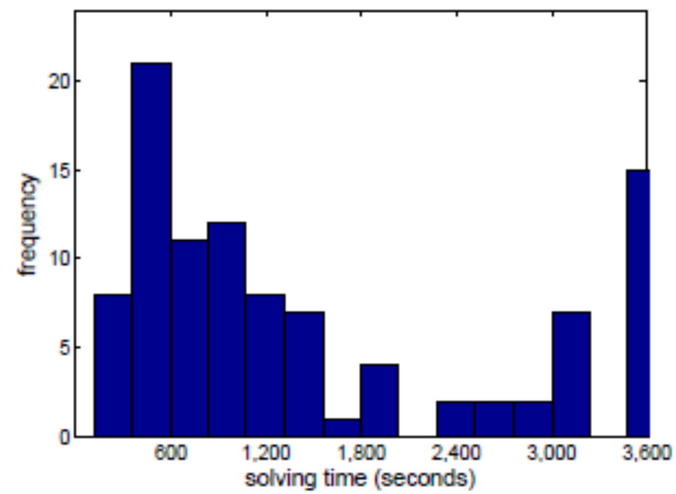
Results of Scenario 1

$4n$ random faults are injected into R_i, i ={24, 26, 28, 30} of GOST (n faults for each i, $N = 4n$).



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

$\lambda(N)=2^{12.2}$



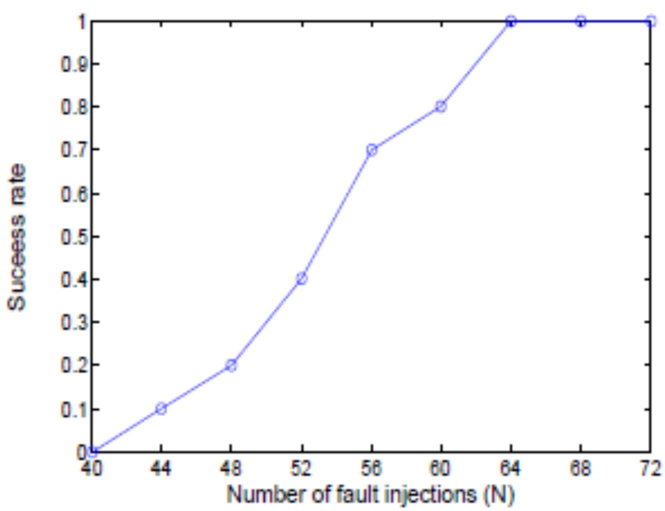
(b) N=8, $V(N) = 32,913$, $A(N) = 63,689$, $v(N) =$ 1226KB

$\lambda(N)=2^{16.7}$

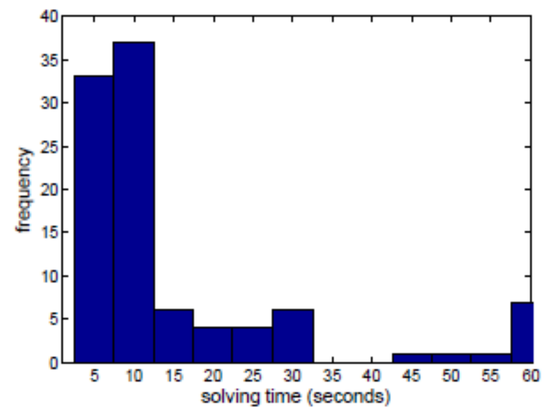
N=8 faults are required to recover the master key, which is less than 64 in [Kim10].

Results of Scenario 2

2*n* random faults are injected into R_i, i ={30, 31} of GOST (*n* faults for each *i*, $N = 2n$).



(a) Success rate (different N)



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

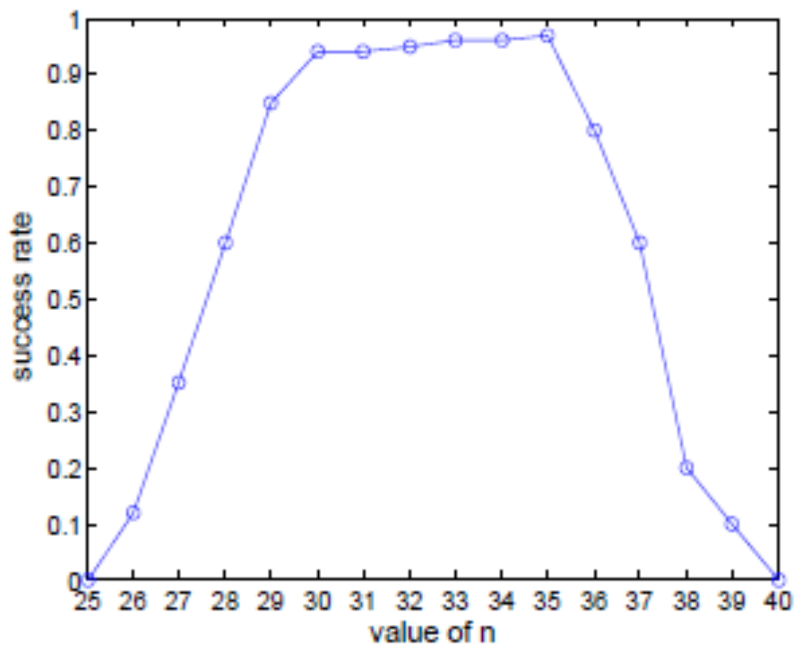
64 faults to recover the 8 S-Boxes

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

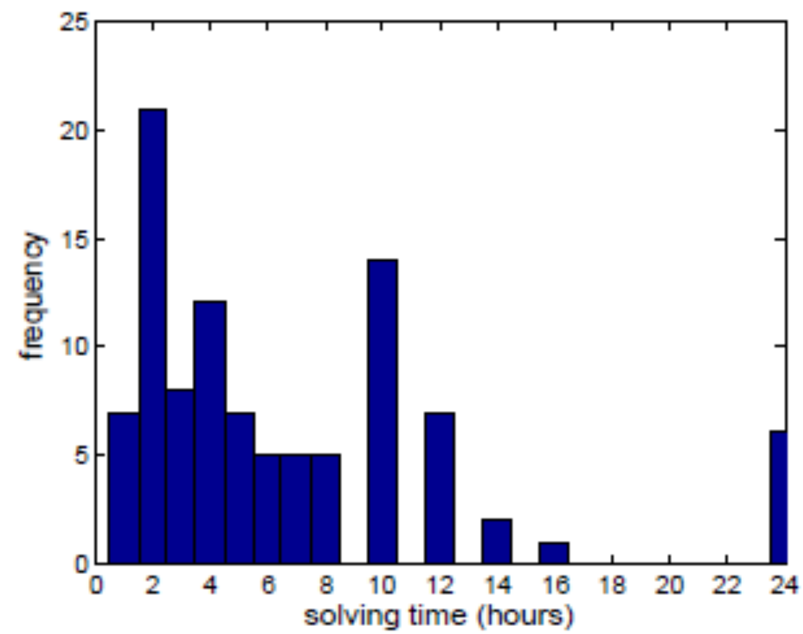
S-box	$a_1, a_2, \ldots, a_{64}$	S-box	$a_1, a_2, \ldots, a_{64}$
$S1$	0x3e4a983e4b4a3174	$S5$	0x0ab8873cec12349e
$S2$	0xf478c97494c208a6	$S6$	0x1dceda3679486e34
$S3$	0x6986bf52669eec3c	$S7$	0xf5c8eb982aead2b2
$S4$	0x5802b282ac52f22e	$S8$	0x5c4a3b560eba85b6

Results of Scenario 3

9*n* random faults are injected into R_i, *i* ={23,24,25,26,27, 28,29,30,31} of GOST (*n* faults for each *i*, *N* = 9*n*).



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



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

270 faults for the recovery of both of the key and 8 S-Boxes

Outline

- **Motivation?** Algebraic Fault Analysis
- **Target?** GOST and Attack Scenarios
- **Technique?** AFA on GOST
- **Results?** Key Recovery and Reverse Engineering
- **Summary? Conclusion of Our Work**

Conclusion of Our Work

Make a comprehensive study of AFA on GOST

- **AFA is Efficient**: when the whole design of GOST is known, the key recovery requires only 8 fault injection, less than 64 in previous DFA work.
- **AFA is Powerful:** can be used for reverse engineering, even both the key and S-Boxes are secret.
- **AFA is Automatic**: no need to analyze the fault propagation.
- **AFA is Generic**: apply to different attack scenarios.
- **One lesson:** keeping some components in a cipher secret cannot guarantee its security.

Thanks!

Q & A

Email：zhaoxinjieem@163.com

fanzhang@zju.edu.cn