文章编号:2096-1618(2022)01-0001-07

基于随机故障注入的GOST差分故障攻击方法

蔡承伯[1], 杜之波[1], 吴 震[1], 李圣泉[2], 江 楠[2]
(1.成都信息工程大学网络空间安全学院,四川 成都 610225;2.国电南京自动化股份有限公司 南京华盾电力信息安全测评有限公司,江苏 南京 211106)

摘要:目前针对GOST算法提出的差分故障攻击方法均对故障模型做了假设限定,其要求在特定的时机或位置引入故障。但在实际攻击中,攻击者难以精确控制故障的发生,攻击的普适性较差。此外,国内外已有的攻击实验均为仿真数据,回避了真实攻击中会遇到的错误密文筛选问题。基于此,提出一种基于随机故障注入的GOST差分故障攻击方法。该方法将随机故障注入区间扩大至GOST算法的最后八轮,在注入完成后只需简单筛选错误密文样本,即可利用密钥筛选方法依次恢复出最后八轮子密钥,最终达到恢复算法主密钥的目的。并对无防护GOST智能卡进行了电源故障注入攻击实验,实验结果表明,该方法不仅能在真实实验环境中成功恢复出算法的主密钥,而且扩大了故障诱导的范围,降低了攻击的难度,提高了攻击的灵活性和实用性。

关 键 词:GOST算法;差分故障攻击;随机故障注入;错误密文筛选;密钥筛选方法
中图分类号:TN918 **文献标志码**:A
doi:10.16836/j.cnki.jcuit.2022.01.001

0 引言

1977年美国将DES算法确立为数据加密标准(data encryption standard),此后在其他国家和地区出现了一系列DES替代算法,1989年苏联发布在GOST 28147-89标准中[1]的GOST分组加密算法就是其中之一。2015年俄罗斯联邦制定了GOST34.12-2015加密标准,并于2016年1月1日起生效,该标准包含了两种对称加密算法,其中之一就是GOST,在新标准中又被称为Magma[2]。针对GOST的密码分析工作主要有线性分析[3]、差分分析[3-4]和滑动攻击[5],但这些关于GOST的数学安全性分析都脱离不了"降低代数复杂度"的特定框架,这意味着在现实条件下很难对其构成真正的威胁。

差分故障攻击(differential fault attack, DFA)是一种常见的侧信道攻击技术,在1997年由两位以色列的密码学家Biham和Shamir提出[6]。其核心思想是通过在密码设备运行过程中导入故障值,然后结合正确和错误密文计算出差分值,最后利用密文差分值和相关密钥之间的关系进行密钥破解。利用该方法能够恢复出多种密码算法的密钥,例如AES[7]、SM4[8-10]、FeW[11]、PRESENT[12]等。在GOST的差分故障攻击研究中,文献[13]首次给出GOST的差分故障攻击方法,研究人员选择在倒数第二轮的右半部分导入半字节的随机故障来攻击最后两轮子密钥,注入64次就能成功恢复主密钥。文献[14]提供了两种GOST的差分故障攻击方法,共同点是都采用单字节随机故障模型,其主要区别是攻击最后一轮子密钥时,方法1诱导的故障位置在最后一轮的右半部分,方法2则将故障位置扩展至最后的两轮。仿真实验结果表明在两种模型下恢复主密钥分别需要72和36个故障。上述攻击方法均有两个共同的现实问题:(1)攻击要能成功执行,攻击者必须将故障导入特定的位置,在实际攻击中严格的前提条件导致该方法很难实施,因此针对GOST的差分故障攻击目前尚没有公开发表的实测结果。(2)故障数据都是特定的仿真数据,对错误密文筛选的过程进行回避。实际上,故障注入产生的错误数据并不是全部有效,攻击者需要根据单次攻击的具体情况对错误密文进行筛选。

针对GOST算法的研究现状,提出一种更符合实际攻击的GOST差分故障攻击方法。首先故障诱导时不再局限于某个位置,而是扩大至算法的最后八轮,这不仅极大地降低了故障诱导的难度,还扩大了故障诱导的范围。其次密钥筛选方法让攻击者筛选错误密文时不再需要特定的错误密文,只需在故障分析时简单筛选错误密文即可成功攻击主密钥。

1 GOST算法介绍

1.1 GOST加密算法

GOST分组加密算法是对称加密算法,属于Feistel

收稿日期:2021-06-16
基金项目:"十三五"国家密码发展基金资助项目(MMJJ20180224);四川省重点研发资助项目(2019YFG0096)

网络结构,明文分组采用的是 64 比特,密钥长度采用的是 256 比特,总共迭代 32 轮。在每一轮中,F 函数的输出与轮输入的左半部分进行异或运算成为新的左半部分,然后左右交换成为下一轮的输入,最后一轮左右不交换。整体轮结构如图 1 所示。

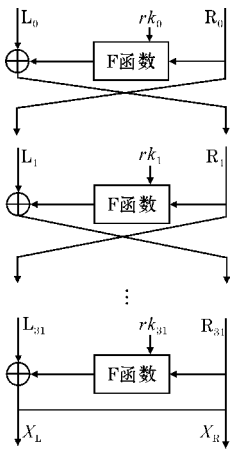

图 1 GOST 轮结构图

1.2 GOST 的 F 函数

F 函数如图 2 所示,由 3 个部分组成。首先右半部分数据与该轮子密钥进行模 2^{32} 加操作,然后将 32 比特结果分为 8 个 4 比特分组,依次进入 8 个 S 盒。最后将 8 个 S 盒的输出重组成 32 比特字,循环左移 11 比特得到输出结果[15]。

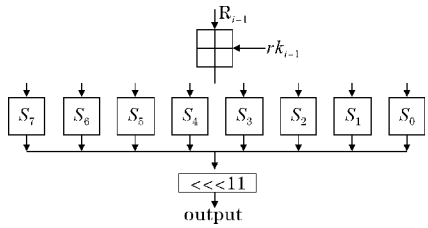

图 2 GOST 的 F 函数

GOST 在 GOST 28147-89 标准中没有特定的 S 盒,但是在最新标准 GOST 34.12-2015 中填补了对 S 盒的定义,并且是固定唯一的。具体见表 1。

1.3 密钥编排

将 256 比特的主密钥 K 分为 8 组 32 比特的子密钥 K_0、K_1、K_2、K_3、K_4、K_5、K_6、K_7,第 i 轮使用的子密钥如表 2 所示。

表 1 GOST 34.12-2015 提供的 S 盒

S 盒	列															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	4	6	2	10	5	11	9	14	8	13	7	0	3	15	1
1	6	8	2	3	9	10	5	12	1	14	4	7	11	13	0	15
2	11	3	5	8	2	15	10	13	14	1	7	4	12	9	6	0
3	12	8	2	1	13	4	15	6	7	0	10	5	3	14	9	11
4	7	15	5	10	8	1	6	13	0	9	3	14	11	4	2	12
5	5	13	15	6	9	2	12	10	11	7	8	1	4	3	14	0
6	8	14	2	5	6	9	1	12	15	4	11	0	13	10	3	7
7	1	7	14	13	0	5	8	3	4	15	10	6	9	12	11	2

表 2 密钥编排

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
子密钥	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
子密钥	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0

2 GOST 算法的 DFA 研究进展

2.1 DFA

一般情况下,故障攻击分为以下 4 个步骤:选择合适的故障模型;针对故障模型选择合适的注入手段进行故障注入;对获得的错误密文样本进行选择性筛选;对筛选后的错误密文样本进行故障分析。在进行故障分析时,如果采用了差分分析法,那么这种故障攻击就称为差分故障攻击。

差分故障攻击主要是在密码芯片运行的过程中进行故障诱导,利用得到的错误密文和正确密文计算出输入差分和输出差分,再根据算法结构建立差分方程,求解差分方程得到轮密钥的候选值集合,通过多次攻击缩小候选值范围,最终得到唯一正确的轮密钥。攻

击原理如图 3 所示。

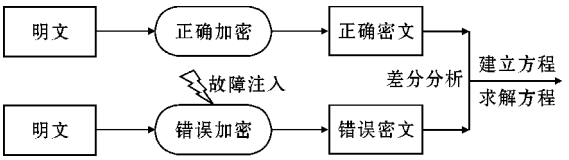

图 3　DFA 原理

2.2　GOST 已有 DFA 方法

2012 年 Jongsung Kim[13] 首次提出了一种 GOST 密码差分故障攻击方法,其攻击原理是利用 GOST 独有的故障传播路径将随机的半字节故障分别注入到 R_{30} 的第 7 个、第 8 个、第 6 个、第 5 个、第 4 个、第 3 个、第 2 个、第 1 个半字节中,一次性攻击最后两轮的子密钥。理论上基于此模型恢复 K_{31} 和 K_{32} 需要 16 个错误密文,恢复主密钥需要 64 个错误密文。该方法具有以下特点:(1)攻击选用的故障模型相对严苛,注入半字节随机故障时,攻击者虽然不控制故障的取值,但是需要控制故障的位置;(2)在实际攻击中利用此方法攻击最后八轮子密钥时,攻击 K_{31} 和 K_{32} 时需要进行一轮故障注入,攻击 K_{29} 和 K_{30} 时需要重新进行一轮故障注入,换言之恢复主密钥总共需要四轮故障注入;(3)在单轮的故障注入后还需要对错误密文进行筛选,并且至少选出 16 个符合要求的错误密文。

2015 年陶智[14] 提出了两种 GOST 密码差分故障攻击方法,攻击采用的均为单字节随机故障模型。方法 1 采用的攻击策略是将随机单字节故障注入到最后一轮 R_{31} 的第 1 个、第 2 个、第 3 个、第 4 个字节中,通过差分分析来恢复最后一轮子密钥的某一字节,最后通过重复攻击来恢复最后一轮子密钥的所有字节。但是这种方法与文献[13]提出的方法一样,存在故障导入精准率太低的问题。因此,方法 2 在方法 1 的基础上提出了一种理论上的改进策略,在攻击最后一轮子密钥时将故障注入的范围扩大为最后两轮,使用该策略在进行差分分析前,要根据密文差分和故障注入位置的关系来判断故障注入的轮数和字节位置。如果不符合要求则需要重新进行故障注入,直到获取的错误密文数量满足攻击成功所需的错误密文数量为止。仿真实验结果表明,使用这两种不同的方法进行攻击分别需要 72 个和 36 个错误密文。

3　基于随机故障注入的 DFA

3.1　符号说明

本文使用的符号说明如表 3 所示。

表 3　符号说明

符号	表达式	含义	位数
P	$P=(L_0,R_0)$	初始明文	64 比特
C	$C=(X_L,X_R)$	正确密文	64 比特
C^*	$C^*=(X_L^*,X_R^*)$	错误密文	64 比特
K	$K=K_0\|\|K_1\|\|K_2\|\|K_3\|\|K_4\|\|K_5\|\|K_6\|\|K_7$	主密钥	256 比特
rk_i	$rk_i=(rk_{i,7},rk_{i,6},rk_{i,5},rk_{i,4},rk_{i,3},rk_{i,2},rk_{i,1},rk_{i,0})$	第 i+1 轮子密钥	32 比特
L_i	$\mathrm{L}_i=(\mathrm{L}_{i,7},\mathrm{L}_{i,6},\mathrm{L}_{i,5},\mathrm{L}_{i,4},\mathrm{L}_{i,3},\mathrm{L}_{i,2},\mathrm{L}_{i,1},\mathrm{L}_{i,0})$	第 i 轮的左半部分	32 比特
R_i	$\mathrm{R}_i=(\mathrm{R}_{i,7},\mathrm{R}_{i,6},\mathrm{R}_{i,5},\mathrm{R}_{i,4},\mathrm{R}_{i,3},\mathrm{R}_{i,2},\mathrm{R}_{i,1},\mathrm{R}_{i,0})$	第 i 轮的右半部分	32 比特
L_i^*	$\mathrm{L}_i^*=(\mathrm{L}_{i,7}^*,\mathrm{L}_{i,6}^*,\mathrm{L}_{i,5}^*,\mathrm{L}_{i,4}^*,\mathrm{L}_{i,3}^*,\mathrm{L}_{i,2}^*,\mathrm{L}_{i,1}^*,\mathrm{L}_{i,0}^*)$	第 i 轮带有故障的左半部分	32 比特
R_i^*	$\mathrm{R}_i^*=(\mathrm{R}_{i,7}^*,\mathrm{R}_{i,6}^*,\mathrm{R}_{i,5}^*,\mathrm{R}_{i,4}^*,\mathrm{R}_{i,3}^*,\mathrm{R}_{i,2}^*,\mathrm{R}_{i,1}^*,\mathrm{R}_{i,0}^*)$	第 i 轮带有故障的右半部分	32 比特
S_{T}	$S_{\mathrm{T}}=(S_{\mathrm{T},7},S_{\mathrm{T},6},S_{\mathrm{T},5},S_{\mathrm{T},4},S_{\mathrm{T},3},S_{\mathrm{T},2},S_{\mathrm{T},1},S_{\mathrm{T},0})$	S 盒正确输入	32 比特
S_{F}	$S_{\mathrm{F}}=(S_{\mathrm{F},7},S_{\mathrm{F},6},S_{\mathrm{F},5},S_{\mathrm{F},4},S_{\mathrm{F},3},S_{\mathrm{F},2},S_{\mathrm{F},1},S_{\mathrm{F},0})$	S 盒错误输入	32 比特
S_{in}	$S_{\mathrm{in}}=(S_{\mathrm{in},7},S_{\mathrm{in},6},S_{\mathrm{in},5},S_{\mathrm{in},4},S_{\mathrm{in},3},S_{\mathrm{in},2},S_{\mathrm{in},1},S_{\mathrm{in},0})$	S 盒输入差分	32 比特
S_{out}	$S_{\mathrm{out}}=(S_{\mathrm{out},7},S_{\mathrm{out},6},S_{\mathrm{out},5},S_{\mathrm{out},4},S_{\mathrm{out},3},S_{\mathrm{out},2},S_{\mathrm{out},1},S_{\mathrm{out},0})$	S 盒输出差分	32 比特
S_k		第 k 个 S 盒	4 比特

3.2　基本过程

基本过程如下:(1)选择一组初始明文,使用密钥 K 进行加密得到正确密文。(2)在主密钥不变的情况下再次对这组明文进行加密,在后八轮任意位置进行随机故障注入,完成后获取一定数量的故障密文,对故

障密文进行简单筛选。随后通过差分分析,结合密钥筛选方法依次恢复出后八轮子密钥 rk_{31}、rk_{30}、rk_{29}、rk_{28}、rk_{27}、rk_{26}、rk_{25}、rk_{24}。(3)结合密钥编排的逆方法恢复出256 位主密钥 K。

3.3 攻击过程详述

步骤 1 采集正确密文信息和泄露的能量曲线。

选择初始明文 $P(\mathrm{L}_0,\mathrm{R}_0)$= FE DC BA 98 76 54 32 10,在密钥为 K 的 GOST 算法下进行加密,得到正确密文 $C(X_\mathrm{L},X_\mathrm{R})$。同时用示波器连接插入智能卡的读卡器,对泄露的能量曲线进行采集,获取最后八轮的运行时间,以便确定导入随机故障的位置,如图 4 所示。

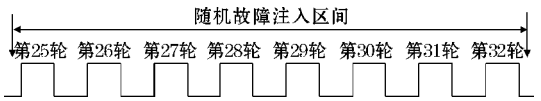

图 4 随机故障注入

步骤 2 执行随机故障注入,获取一定数量的错误密文。

(1)使用密钥 K 再次对明文 P 进行加密,根据步骤 1 获取的能量曲线确定最后八轮的运行时间段,对最后八轮执行随机故障注入,获取一定数量的错误密 $C^*(X_\mathrm{L}^*,X_\mathrm{R}^*)$。

(2)对获取的错误密文进行一个简单筛选。筛选出与正确密文长度一样,但值不一样的错误密文。

步骤 3 攻击第 32 轮。

(1)计算 S 盒的输入差分 S_{in}以及输出差分 S_{out}:

$$S_{\mathrm{in}}=(\mathrm{R}_{31}+rk_{31})\oplus(\mathrm{R}_{31}^*+rk_{31})=(X_\mathrm{R}+rk_{31})\oplus(X_\mathrm{R}^*+rk_{31}) \tag{1}$$

$$S_{\mathrm{out}}=((X_\mathrm{L}\oplus\mathrm{L}_{31})\oplus(X_\mathrm{L}^*\oplus\mathrm{L}_{31}))\ggg 11=(X_\mathrm{L}\oplus X_\mathrm{L}^*)\ggg 11 \tag{2}$$

对应 8 个 S 盒则有:

$$\begin{aligned}&S_k(\mathrm{R}_{31}+rk_{31})\oplus S_k((\mathrm{R}_{31}+rk_{31})\oplus S_{\mathrm{in}})\\&=S_k(X_\mathrm{R}+rk_{31})\oplus S_k((X_\mathrm{R}+rk_{31})\oplus S_{\mathrm{in}})\\&=S_{\mathrm{out},k}\quad k=0,1,2,3,4,5,6,7\end{aligned} \tag{3}$$

(2)建立差分方程。记 S_{out}、R_{31}、X_R、R_{31}^*、X_R^* 从左到右的 4 个字节分别为:$S_{\mathrm{out}}[x]$,$x=0,1,2,3$、$\mathrm{R}_{31}[x]$,$x=0,1,2,3$、$X_\mathrm{R}[x]$,$x=0,1,2,3$、$\mathrm{R}_{31}^*[x]$,$x=0,1,2,3$、$X_\mathrm{R}^*[x]$,$x=0,1,2,3$,则有:

$$S_{\mathrm{out}}[x]=S_{\mathrm{out}}\gg(24-8x)\ \&\ 0\mathrm{xFF},x=0,1,2,3 \tag{4}$$

$$\mathrm{R}_{31}[x]=\mathrm{R}_{31}\gg(24-8x)\ \&\ 0\mathrm{xFF},x=0,1,2,3 \tag{5}$$

$$X_\mathrm{R}[x]=X_\mathrm{R}\gg(24-8x)\ \&\ 0\mathrm{xFF},x=0,1,2,3 \tag{6}$$

$$\mathrm{R}_{31}^*[x]=\mathrm{R}_{31}^*\gg(24-8x)\ \&\ 0\mathrm{xFF},x=0,1,2,3 \tag{7}$$

$$X_\mathrm{R}^*[x]=X_\mathrm{R}^*\gg(24-8x)\ \&\ 0\mathrm{xFF},x=0,1,2,3 \tag{8}$$

S_T 为算法正常运行时 S 盒的正确输入,S_F 为故障注入后 S 盒的错误输入;记 S_T、S_F、S_{in}从左到右对应的 4 个字节分别为 $S_\mathrm{T}[x]$,$x=0,1,2,3$、$S_\mathrm{F}[x]$,$x=0,1,2,3$、$S_{\mathrm{in}}[x]$,$x=0,1,2,3$,则有:

$$\begin{aligned}S_\mathrm{T}[x]&=(S_{(7-2x)}((\mathrm{R}_{31}[x]+k_{\mathrm{byte}})\gg 4\ \&\ 0\mathrm{xF}))\ll 4\ \&\ 0\mathrm{xFF}\\&\oplus S_{(7-(2x+1))}((\mathrm{R}_{31}[x]+k_{\mathrm{byte}})\ \&\ 0\mathrm{xF})\\&=(S_{(7-2x)}((X_\mathrm{R}[x]+k_{\mathrm{byte}})\gg 4\ \&\ 0\mathrm{xF}))\ll 4\ \&\ 0\mathrm{xFF}\\&\oplus S_{(7-(2x+1))}((X_\mathrm{R}[x]+k_{\mathrm{byte}})\ \&\ 0\mathrm{xF}),x=0,1,2,3\end{aligned} \tag{9}$$

$$\begin{aligned}S_\mathrm{F}[x]&=(S_{(7-2x)}((\mathrm{R}_{31}^*[x]+k_{\mathrm{byte}})\gg 4\ \&\ 0\mathrm{xF}))\ll 4\ \&\ 0\mathrm{xFF}\\&\oplus S_{(7-(2x+1))}((\mathrm{R}_{31}^*[x]+k_{\mathrm{byte}})\ \&\ 0\mathrm{xF})\\&=(S_{(7-2x)}((X_\mathrm{R}^*[x]+k_{\mathrm{byte}})\gg 4\ \&\ 0\mathrm{xF}))\ll 4\ \&\ 0\mathrm{xFF}\\&\oplus S_{(7-(2x+1))}((X_\mathrm{R}^*[x]+k_{\mathrm{byte}})\ \&\ 0\mathrm{xF}),x=0,1,2,3\end{aligned} \tag{10}$$

$$S_{\mathrm{in}}[x]=S_\mathrm{T}[x]\oplus S_\mathrm{F}[x],x=0,1,2,3 \tag{11}$$

其中 k_{byte}为 256 个候选字节。

基于随机故障注入模型,S_{in}和 S_{out}应该满足下列式子:

$$S_{\mathrm{in}}[x]=S_{\mathrm{out}}[x],x=0,1,2,3 \tag{12}$$

(3)使用密钥筛选方法,计算出第 32 轮子密钥 rk_{31}。

步骤 4 攻击剩余七轮的子密钥,每轮需要的错误密文均来自于步骤 2。在步骤 3 的基础上对 GOST 最后一轮解密,获得第 31 轮正确的和错误的轮输出,重复步骤 3 的操作,恢复出rk_{30}。以此类推恢复出rk_{29}、rk_{28}、rk_{27}、rk_{26}、rk_{25}、rk_{24}。

步骤 5 根据密钥编排的逆方法,恢复主密钥 K。

3.4 密钥筛选方法

步骤 3(3)提到的密钥筛选方法具体如下:

(1)将 256 个候选字节从小到大放入数组 k_{byte},同时设置一个全为 0 的二维数组 $K[4][256]$,其中 4 代表该轮子密钥从左到右的 4 个字节,256 对应 k_{byte}的 256 个候选字节。

$$k_{\mathrm{byte}}[256]=\{0\mathrm{x00},0\mathrm{x01},0\mathrm{x02},\cdots,0\mathrm{xFF}\} \tag{13}$$

$$K[4][256]=\{\{0,0,\cdots,0,0\},\{0,0,\cdots,0,0\},\{0,0,\cdots,0,0\},\{0,0,\cdots,0,0\}\} \tag{14}$$

(2)如果 S_{out}不为 0,就将 256 个 k_{byte}依次带入等式(9)和式(10),当式(11)成立时,二维数组对应的位置自增 1,不成立则不变。例如分析第一个错误密文时,$x=0$ 对应的 k_{byte}等于 0x00 和 0x03 时使式(11)成立,其余的不成立,则 $K[0][256]=\{1,0,0,1,\cdots,0,0\}$,其他 3 个字节同理。

(3)分析完所有错误密文后,找到二维数组 K 各行最大的值,其对应位置的候选字节即为步骤(4)需要的字节。例如针对第一个字节,$K[0][256]$ =

{5,18,5,1,…,88,…,0,0},最大值 88 位于数组元素编号 121 处,则对应的候选字节为 0x7C,其他 3 个字节同理。

(4)记步骤(3)得到的 4 个字节依次为 $k_{\text{byte},0}$、$k_{\text{byte},1}$、$k_{\text{byte},2}$、$k_{\text{byte},3}$,因为模 2^{32} 加法运算有可能出现进位的情况,所以这里得到的 $k_{\text{byte},0}$、$k_{\text{byte},1}$、$k_{\text{byte},2}$ 不一定是子密钥正确的前 3 个字节,因此需要对这 3 个字节作二次处理,具体如下:

如果 $k_{\text{byte},3}+X_{\text{R}}[3]>0\text{xFF}$,则 $k_{\text{byte},2}=(k_{\text{byte},2}-1)\bmod 2^8$;如果 $k_{\text{byte},2}+X_{\text{R}}[2]>0\text{xFF}$,则 $k_{\text{byte},1}=(k_{\text{byte},1}-1)\bmod 2^8$;如果 $k_{\text{byte},1}+X_{\text{R}}[1]>0\text{xFF}$,则 $k_{\text{byte},0}=(k_{\text{byte},0}-1)\bmod 2^8$。最后将处理后的 4 个字节 $k_{\text{byte},0}$、$k_{\text{byte},1}$、$k_{\text{byte},2}$、$k_{\text{byte},3}$ 重组为 32 比特的字,得到正确的 rk_{31}。

4　实验与分析

4.1　实验环境

所需要的实验环境如表 4 所示。

表 4　实验环境

实验设备	设备参数
PC 机	处理器:Intel(R) Core(TM) i7-8700、内存:32.0 GB
集成开发环境	Eclipse
密码设备	无防护 GOST 智能卡
侧信道分析	Inspector 4.8
驱动设备	Power Tracer
故障注入设备	VC Glitcher
示波器	PicoScope

4.2　原始信息采集

首先设置初始明文为:FE DC BA 98 76 54 32 10,然后采集 GOST 算法智能卡正常工作时产生的能量曲线和输出的正确密文信息,对应 3.3 小节的步骤 1。图 5 是示波器采集到的能量曲线,横轴表示算法的运行时间,纵轴表示运行过程中泄露的功耗。图 6 是采集到的正确明文和密文信息:FE DC BA 98 76 54 32 10 4E E9 01 E5 C2 D8 CA 3D。

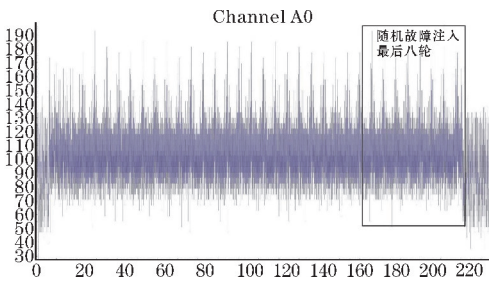

图 5　能量曲线

图 6　正确明文和密文信息

4.3　故障注入

首先通过观察图 5 能量曲线的运行时间轴,以确定 GOST 算法的最后八轮运行区间,在实际攻击中可以选择最后九轮或者最后十轮进行攻击,只要包含最后八轮即可。然后对选中的运行区间实施 1000 次随机故障注入。图 7 是故障注入后 Inspector 采集到的部分数据信息,绿色数据是故障注入失败的返回数据;红色数据是故障注入成功的返回数据;矩形框部分是返回的错误密文,一共有 240 条。最后记录下这 240 条数据,用于后续差分分析。

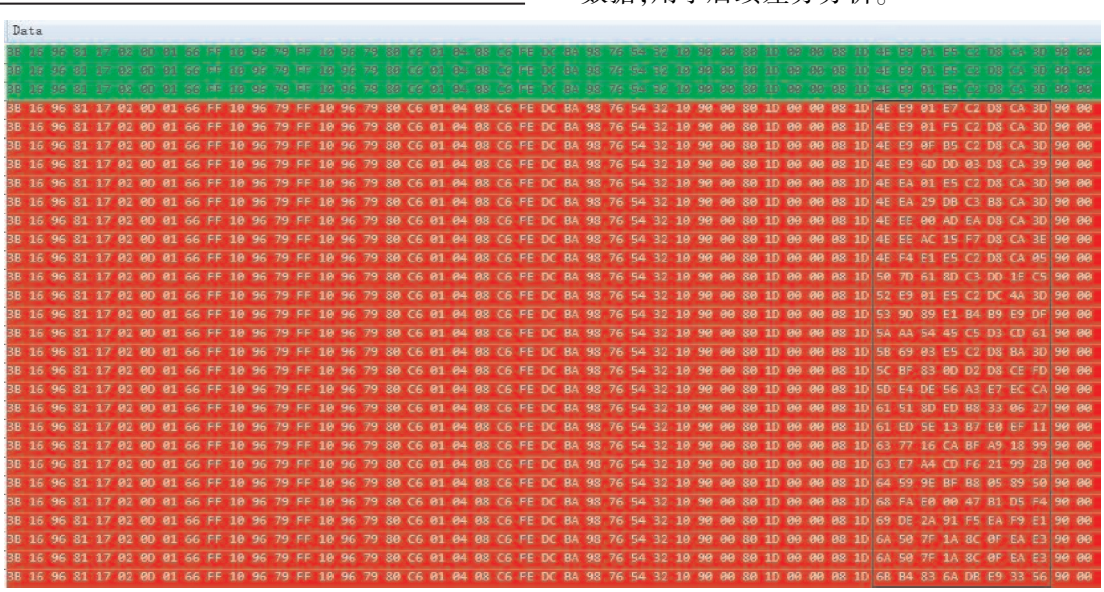

图 7　返回数据信息

4.4 差分分析

使用 3.3 小节步骤 3 和步骤 4 提出的方法对正确密文以及上述获得的 240 密文进行攻击。第 32 轮攻击结果如图 8 所示,在每 4 个字节里面,如果单个字节的置信度最高并且明显高于其他字节,那么这个字节就是正确的候选密钥字节。最终第 32 轮子密钥为:FF EE DD CC。

```
After analyzing 240 ciphertexts
Attack the first byte of round 32
Rank0:(0xFF) Confidence1:(0.1250) Confidence2:(1.0000)
Rank1:(0x7F) Confidence1:(0.0417) Confidence2:(0.3333)
Rank2:(0x09) Confidence1:(0.0375) Confidence2:(0.3000)
Attack the second byte of round 32
Rank0:(0xEE) Confidence1:(0.1470) Confidence2:(1.0000)
Rank1:(0xF1) Confidence1:(0.0547) Confidence2:(0.3824)
Rank2:(0x5E) Confidence1:(0.0500) Confidence2:(0.3529)
Attack the third byte of round 32
Rank0:(0xDD) Confidence1:(0.1208) Confidence2:(1.0000)
Rank1:(0x5D) Confidence1:(0.0833) Confidence2:(0.6897)
Rank2:(0x4D) Confidence1:(0.0750) Confidence2:(0.6207)
Attack the fourth byte of round 32
Rank0:(0xCC) Confidence1:(0.1375) Confidence2:(1.0000)
Rank1:(0x5B) Confidence1:(0.0625) Confidence2:(0.4545)
Rank2:(0x0B) Confidence1:(0.0583) Confidence2:(0.4242)
The key of round 32:(FF EE DD CC)
```

图 8 第 32 轮结果

攻击剩余 7 轮,每一轮使用的错误密文与第 32 轮一致,攻击结果见表 4。

表 4 剩余七轮攻击结果

轮数	轮密钥
第 31 轮	0xBBAA9988
第 30 轮	0x77665544
第 29 轮	0x33221100
第 28 轮	0xF0F1F2F3
第 27 轮	0xF4F5F6F7
第 26 轮	0xF8F9FAFB
第 25 轮	0xFCFDFEFF

4.5 结果验证

使用攻击得到的主密钥 FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 对同一明文进行加密,得到密文 4E E9 01 E5 C2 D8 CA 3D,结果与图 6 采集的正确密文信息一致,表明攻击成功。同时也证明了本文提出的方法在真实实验环境下能够恢复出主密钥,并且相比现有其他方法,扩大了攻击的故障诱导范围,提高了攻击的灵活性和实用性。表 5 对各方法的故障诱导位置作了比较。

表 5 故障诱导位置

方法	第 32 轮	第 31 轮	第 30 轮	第 29 轮	第 28 轮	第 27 轮	第 26 轮	第 25 轮
文献[13]	R_{30}	R_{30}	R_{28}	R_{28}	R_{26}	R_{26}	R_{24}	R_{24}
文献[14]方法一	R_{31}	R_{30}	R_{29}	R_{28}	R_{27}	R_{26}	R_{25}	R_{24}
	L_{30}	L_{29}	L_{28}	L_{27}	L_{26}	L_{25}	L_{24}	L_{23}
文献[14]方法二	R_{30}	R_{29}	R_{28}	R_{27}	R_{26}	R_{25}	R_{24}	R_{23}
	R_{31}	R_{30}	R_{29}	R_{28}	R_{27}	R_{26}	R_{25}	R_{24}
本文方法	最后 8 轮	最后 8 轮	最后 8 轮	最后 8 轮	最后 8 轮	最后 8 轮	最后 8 轮	最后 8 轮

5 结束语

进行差分故障分析时常常是在某假设的故障模型的条件下进行,而故障模型的效率不仅取决于需要的错误密文数量,还取决于在实际攻击中的可用性和可行性。针对现有攻击方法的不足提出基于随机故障注入的 GOST 差分故障攻击方法,将故障诱导的范围扩大到最后八轮,打破了苛刻的理论条件,提高了攻击性和实用性。其次,提出的密钥筛选方法在攻击每一轮子密钥时使用的错误密文都一样,降低了错误密文样本筛选的难度,解决了之前需要根据单轮攻击的具体情况筛选出特定错误密文的难题。本文在一定程度上为同类型密码在实际差分故障攻击中的应用研究提供了新的思路。

参考文献:

[1] GOST 28147-89, Cryptographic Protection for Data Processing Systems[S].

[2] GOST R 34.12-2015, Information technology. Cryptographic data security. Block ciphers[S].

[3] Shorin V V, Jelezniakov V V, Gabidulin E M. Linear and differential cryptanalysis of Russian GOST [J]. Electronic Notes in Discrete Mathematics, 2001, 6:538-547.

[4] Courtois N T. An improved differential attack on full GOST[M]. The new codebreakers. Springer Berlin Heidelberg,2016:282-303.
[5] Ishchukova E, Babenko L, Anikeev M. Fast implementation and cryptanalysis of GOST R34.12-2015 block ciphers[C]. Proceedings of the 9th International Conference on Security of Information and Networks. 2016:104-111.
[6] Biham E, Shamir A. Differential fault analysis of secret key cryptosystems[C]. Annual international cryptology conference. Springer Berlin Heidelberg, 1997:513-525.
[7] Dusart P, Letourneux G, Vivolo O. Differential fault analysis on AES[C]. International Conference on Applied Cryptography and Network Security. Springer Berlin Heidelberg,2003:293-306.
[8] 张蕾,吴文玲. SMS4 密码算法的差分故障攻击[J]. 计算机学报,2006,29(9):1596-1602.
[9] Li Ruilin, Sun Bing, Li Chao, et al. Different Fault Analysis on SMS4 Using a Single Fault[J]. Information Processing Letters,2011,11(4):156-163.
[10] 荣雪芳,吴震,王敏,等. 基于随机故障注入的 SM4 差分故障攻击方法[J]. 计算机工程,2016,42(7):129-133.
[11] 谢敏,李嘉琪,田峰. FeW 的差分故障攻击[J]. 通信学报,2020,41(4):143-149.
[12] 陈伟建,赵思宇,邹瑞杰,等. PRESENT 密码的差分故障攻击[J]. 电子科技大学学报,2019,48(6):865-869.
[13] Kim J. On the security of the block cipher GOST suitable for the protection in U-business services [J]. Personal and ubiquitous computing,2013,17(7):1429-1435.
[14] 陶智. 若干对称密码算法的安全性分析[D]. 上海:东华大学,2015.
[15] 李悦,李玮,曹艳琴,等. 几种轻量级分组密码算法的性能分析[J]. 计算机应用与软件,2016(10):317-320.

Differential Fault Attack Method on GOST based on Random Fault Injection

CAI Chengbo[1], DU Zhibo[1], WU Zhen[1], LI Shengquan[2], JIANG Nan[2]

(1. College of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, China; 2. Nanjing Huadun Power Information Security Evaluation Co., Ltd., Guodian Nanjing Automation Co., Ltd., Nanjing 211106, China)

Abstract: The GOST algorithm is a standard symmetric encryption algorithm of the Russian Federation. At present, the differential fault attack methods proposed for the GOST algorithm all make assumptions on the fault model., which requires the introduction of faults at a specific time or position. However, in actual attacks, it is difficult for the attacker to precisely control the occurrence of the fault, and the universality generality of the attack is poor. In addition, the existing attack experiments at home and abroad are all simulation data, which evades the problem of wrong ciphertext screening in real attacks. In view of the above problems, this paper proposes a GOST differential fault attack method based on random fault injection. This method extendsexpands the random fault injection interval to the last eight rounds of GOST algorithm. After the injection is completed, only the wrong ciphertext samples are simply filtered, and the keys of the last eight rounds can be recovered in turn by using the key screening method, finally achieving the purpose of recovering the master key of the algorithm. At the end of this paper, the experiment of power supply fault injection attack on unprotected GOST smart card is carried out. The experimental results show that this method can not only successfully recover the master key of the algorithm in the real experimental environment, but also expand the scope of fault induction, reduce the difficulty of attack, and improve the flexibility and practicability of attack.

Keywords: GOST algorithm; differential fault attack; random fault injection; wrong ciphertext screening; key screening method