文章编号：2096-1618(2022)01-0008-08

针对轻量级分组密码算法 PRESENT 的随机差分故障攻击

黄湘蜀， 王 敏， 杜之波， 吴 震， 王 燚
(成都信息工程大学网络空间安全学院，四川 成都 610225)

摘要：轻量级分组密码算法 PRESENT 采用了 SPN 网络结构，具有实现面积小、功耗低等特点，因此广泛使用于资源受限的环境中。针对 PRESENT 算法，设计多字节故障模型，在 PRESENT 算法的第 30、29 轮的任意位置分别进行随机故障注入，注入的字节数不固定。利用 PRESNET 算法的故障传播路径，构建输出差分和可能输入值之间的关系，通过提出的并行 S 盒分析方法得到正确输入，进而得到正确的轮子密钥。最后通过分析密钥编排方案，只需要两轮正确的轮子密钥即可推导出初始的80 bits主密钥。实验结果表明，与现有的针对 PRESENT 算法的故障攻击相比，利用提出的故障模型，可以将攻击复杂度由 2^{31} 降低到 2^{18}，并且轮密钥攻击平均时长由20000 ms降低到1000 ms。与此同时，提出的方法将单字节、固定位置故障模型改进为多字节、任意位置的故障模型，更加符合实际的攻击情况，降低了对故障注入设备的要求，提高方法的实用性。

关 键 词：PRESENT 算法；多字节故障模型；随机故障注入；故障传播路径；并行 S 盒分析
中图分类号：TP309 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2022.01.002

0 引言

轻量级分组密码算法凭借其结构简单、功耗低、实现面积小、适应性广泛等特点，已经成为物联网应用的关键组成部分。目前在物联网应用中，已经有许多轻量级分组密码算法，如 MIBS[1]、TWINE[2]、HIGHT[3]、GIFT[4]。这些算法都具有加解密一致、结构简单、易于实现等特点。而 PRESENT 算法是 A. Bogdanov 等[5]提出的一种轻量级分组密码算法，其具有轻量级分组密码普遍特点，同样适合在物联网相关应用中使用。当前针对 PRESENT 算法的分析方法主要有积分攻击[5]、差分代数攻击[6]、多重差分链攻击[7]以及差分故障攻击[8-9]等方法，文献[5]通过分析 PRESENT 算法一系列中间状态的和的概率进行攻击，攻击复杂度为 2^{20}。文献[6]将差分攻击和代数攻击相结合，建立并且简化代数方程组，攻击复杂度为 $2^{64.58}$。文献[7]利用多重差分链对 PRESENT 算法进行攻击，攻击复杂度为 2^{81}。差分故障攻击[10]作为旁路攻击[11]的一种，最早在 1996 年由 Boneh 提出，是一种将差分分析[12]和故障攻击[13]相结合的技术。攻击原理是在加密过程中的某一时间段进行故障注入，制造数据差分，利用输出的故障密文和故障动作，结合差分方程得到轮密钥的可能值。该方法已经应用于 CLEFIA[14]、SM4[15]等分组密码上，对分组密码算法构成了严重的威胁。文献[8]采用的是面向字节的随机故障注入模型，在 PRESENT 算法的加密过程中引入单字节的故障，通过差分故障特性识别需要的故障密文，并由识别得到的故障密文恢复轮密钥。文献[9]建立单字节的随机故障模型，利用 PRESENT 算法 P 置换层的故障传播特性，通过单字节扩展的方式恢复轮子密钥，攻击复杂度为 2^{31}。现有的两种方法虽然都在理论上进行了证实，但在实际操作中攻击者难以控制故障注入的位置以及产生故障的字节数，所以当前针对 PRESENT 算法的故障注入攻击仍然有一定的局限性。

针对上述问题，为提高故障攻击方法在实际攻击中的可行性以及效率，降低故障注入的难度，对现有的单字节差分故障模型进行改进，利用固定的故障传播路径，建立多字节的故障注入模型。通过电压注入的方式在第 30、29 轮任意位置进行随机故障注入，并由输出差分与 S 盒输入值之间的关系，筛选出正确的 S 盒输入。设计的故障模型，可将攻击复杂度降低 41.9%，攻击平均时间由20000 ms降低到1000 ms。

1 PRESENT 算法

1.1 符号说明

K：表示密钥；

收稿日期：2021-06-30
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)。

k_i:表示当前轮密钥的第 i 比特位;

$p(i)$:第 i 比特位的置换运算;

$S[\ ]$:表示 S 替换;

S^i:表示第 i 轮 S 盒输入;

P^i:表示第 i 轮 P 盒输入;

C^i:表示第 i 轮 P 盒输出;

Nc^i:表示经过第 i 轮 P 置换后故障变化到的比特位置;

Ns^i:表示第 i 轮故障所在的 S 盒;

K_j^i:表示第 i 轮的 j 比特的主密钥;

f:表示输入差分;

f':表示输出差分;

S_{in}:S 盒的输入值;

$F(f')$:表示输出差分为 f' 时,输入差分的候选集合。

1.2　PRESENT 算法迭代过程

PRESENT 算法的明文和密文长度都为 64 比特,主密钥长度为 80 比特或者 128 比特,文中仅讨论 80 比特的情况。该算法共有 31 轮迭代过程,每一轮的子密钥长度为 64 比特,如图 1 所示,在第 31 轮结束后,再进行一次轮密钥加的白化操作。

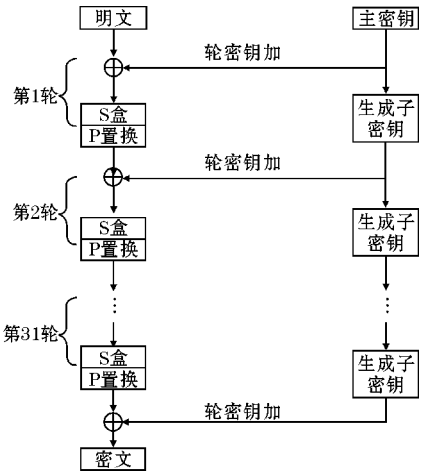

图 1　PRESENT 算法加密流程

算法主要是由“轮密钥加”“S 盒置换”“P 盒置换”3 部分组成。“轮密钥加”为按位进行异或操作。“S 盒置换”为按照 S 盒置换表进行置换,每个 S 盒输入和输出都是 4 比特(为方便阐述,文中将 4bits 视作一个字节),由 16 个 S 盒组成,一共64 比特,S 盒置换如表 1 所示。

表 1　S 盒置换表

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

“P 盒置换”相当于移位操作,每比特位置按式(1)进行置换,其中 i 为比特位。

$$p(i)=\begin{cases}16\cdot i \bmod 63 & 0\leqslant i\leqslant 62\\ 63 & i=63\end{cases} \tag{1}$$

1.3　PRESENT 密钥扩展

以 80 比特的主密钥为例来阐述 PRESENT 算法的密钥扩展方法。主密钥首先存储在存储器中,表示为 $K=k_{79}k_{78}\cdots k_0$,在进行轮密钥加操作时按位取当前密钥寄存器的高 64 比特,表示为 $K=k_{79}k_{78}\cdots k_{17}k_{16}$。提取完子密钥后按如下方式进行扩展。首先,将 80 比特密钥循环左移 61 位,之后依次从左到右取 4 比特进行 S 盒置换,最后将 $k_{19}k_{18}k_{17}k_{16}k_{15}$ 与轮计数器 r 做异或运算,并且将异或后产生的结果取代原来的 5 比特数据,如式(2)所示。

$$\begin{cases}[k_{79}k_{78}\cdots k_1k_0]=[k_{18}k_{17}\cdots k_{20}k_{19}]\\ [k_{79}k_{78}k_{76}k_{77}]=S[k_{79}k_{78}k_{76}k_{77}]\\ [k_{19}k_{18}k_{17}k_{16}k_{15}]=[k_{19}k_{18}k_{17}k_{16}k_{15}]\oplus r\end{cases} \tag{2}$$

2　PRESENT 分组密码故障分析

2.1　PRESENT 算法故障传播路径

PRESENT 算法属于轻量级分组密码算法的一种,算法中的 S 盒和 P 置换是重要的组成成分,在故障攻击中攻击者可以根据 S 盒的非线性将 S 盒的输入缩小范围,通过对 P 置换的分析得到故障的传播路径。图 2 是最后两轮加密过程。

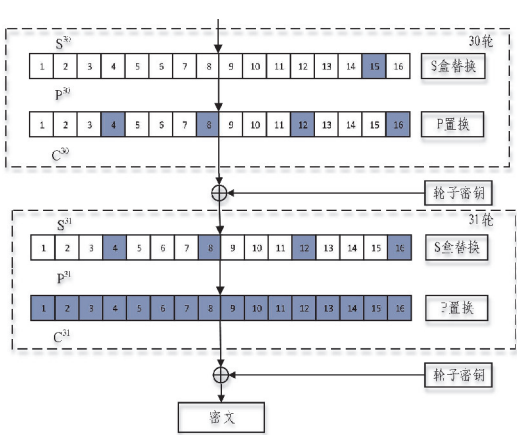

图 2　PRESENT 算法最后两轮加密过程

图 2 中,首先以单字节故障进行说明,多字节故障传播类似。假设在 S^{30} 第 15 个字节处产生了单字节的故障,由于一个字节表示4bits,即最多 4 比特位的数据产生了变化。此时经过 P 置换后,该故障数据会传播到 S^{31} 的 4 个不同的 S 盒上,这 4 个 S 盒都会受到故障数据的影响。若再经过一次 P 置换运算,那么所有字节都会受到故障数据的影响。表 2 是故障传播路径表,其中 i $(0\leqslant i\leqslant 15)$ 表示在第 30 轮注入单字节故障的位置。

表 2 故障传播路径表

i	Nc^{30}	Ns^{31}	Nc^{31}
0	0,16,31,48	0,4,8,12	0,16,32,48,4,20,36,52,8,24,40,56,12,28,44,60
1	1,17,33,49	0,4,8,12	0,16,32,48,4,20,36,52,8,24,40,56,12,28,44,60
2	2,18,34,50	0,4,8,12	0,16,32,48,4,20,36,52,8,24,40,56,12,28,44,60
3	3,19,35,51	0,4,8,12	0,16,32,48,4,20,36,52,8,24,40,56,12,28,44,60
4	4,20,36,52	1,5,9,13	1,17,33,49,5,21,37,53,9,25,41,57,13,29,45,61
5	5,21,37,53	1,5,9,13	1,17,33,49,5,21,37,53,9,25,41,57,13,29,45,61
6	6,22,38,54	1,5,9,13	1,17,33,49,5,21,37,53,9,25,41,57,13,29,45,61
7	7,23,39,55	1,5,9,13	1,17,33,49,5,21,37,53,9,25,41,57,13,29,45,61
8	8,24,40,56	2,6,10,14	2,18,34,50,6,22,38,54,10,26,42,58,14,30,46,62
9	9,25,41,57	2,6,10,14	2,18,34,50,6,22,38,54,10,26,42,58,14,30,46,62
10	10,26,42,58	2,6,10,14	2,18,34,50,6,22,38,54,10,26,42,58,14,30,46,62
11	11,27,43,59	2,6,10,14	2,18,34,50,6,22,38,54,10,26,42,58,14,30,46,62
12	12,28,44,60	3,7,11,15	3,19,35,51,7,23,39,55,11,27,43,59,15,31,47,63
13	13,29,45,61	3,7,11,15	3,19,35,51,7,23,39,55,11,27,43,59,15,31,47,63
14	14,30,46,62	3,7,11,15	3,19,35,51,7,23,39,55,11,27,43,59,15,31,47,63
15	15,31,47,63	3,7,11,15	3,19,35,51,7,23,39,55,11,27,43,59,15,31,47,63

由表 2 可知,当在第 30 轮产生一个单字节故障时,至多会影响第 31 轮的 4 个 S 盒的输入,因此影响至多 16 比特位的 S 盒输出。同理可得,当产生故障字节数为 $i(0\leqslant i\leqslant 15)$ 个时,则会影响下一轮 16 个 S 盒的输入,并且影响 64 比特位的 S 盒输出。利用上述故障传播路径,将有利于进行差分故障分析。

2.2 PRESENT 算法 S 盒差分特征

在差分故障攻击中,最为核心的步骤就是根据差分来计算每个 S 盒可能的输入,为

$$f'=S[S_{\text{in}}\oplus f]\oplus S[S_{\text{in}}] \tag{3}$$

攻击者可以通过已知输出差分 f 构建 S 盒输入差分查询表,如表 3 所示。通过差分查询表来寻找可能的输出差分 f,最后通过式(3)求得 S 盒的输入值 S_{in}。相比于以往的通过穷举的方式寻找 S_{in},利用查表方式能够快速地找到 f 并且求得 S_{in},减少针对 S 盒输入值的穷举量,缩短攻击时间。

表 3 差分 S 盒特征

$f(f'(0001)_2)$	$f(f'(0010)_2)$	$f(f'(0100)_2)$	$f(f'(1000)_2)$
$(0011)_2$,$(0101)_2$	$(0110)_2$,$(0111)_2$	$(0011)_2$,$(0101)_2$	$(0110)_2$,$(0111)_2$
$(0111)_2$,$(1011)_2$	$(1001)_2$,$(1010)_2$	$(1001)_2$,$(1011)_2$	$(1001)_2$,$(1010)_2$
$(1101)_2$,$(1111)_2$	$(1100)_2$,$(1101)_2$	$(1101)_2$,$(1111)_2$	$(1011)_2$,$(1100)_2$
	$(1110)_2$		$(1110)_2$

表 3 仅仅给出了每个 S 盒输出差分只有 1 位为 1 的情况,然而在实际情况中每个 S 盒的输出差分可能有 4 个位为 1,需要根据实际的输出差分来找对应的输入差分。

3 PRESENT 算法的随机差分故障攻击

利用 PRESENT 算法的故障传播特性,可设计如下方案对其实施随机差分故障攻击,具体过程如下:

步骤一 首先随机选取一组明文,将明文在密钥 K 的情况下进行加密,获取对应的密文 C,并且利用示波器采集加密过程中产生的能量信号并转化为能量曲线。将采集到的曲线进行预处理,并把预处理后的曲线结合 PRESENT 算法的流程进行分析,进而确定算法的第 30 轮和第 29 轮的运行时间段,方便后续的故障注入。

步骤二 在智能卡运行期间进行电压故障注入获得错误密文。

(1)根据步骤一采集到的曲线,确定第 30 轮、29 轮的位置,并且在这两轮进行故障注入,进而得到错误密文 C^*。

(2)将正确密文和故障密文进行异或得到差分密文 diff_C。

$$\mathrm{diff_C} = \mathrm{C} \oplus \mathrm{C}^* \tag{4}$$

(3)由故障传播路径,可以根据得到的差分密文 $\mathrm{diff_C}$,通过 P 盒的逆运算 P^{-1} 反推出第 31 轮的 16 个 S 盒的输出差分 diff_{P31}。

$$\mathrm{diff}_{P31} = P^{-1}(\mathrm{diff_C}) \tag{5}$$

(4)若攻击者对每一个满足条件的输入差分都进行分析,会得到 S 盒输入集合 $M\{S_{in1}, S_{in2} \cdots S_{inn}\}$,由于 S 盒的输入为 4 位,因此 M 集合的最大下标为 16。表 4 是输出差分固定的情况下,输入差分和 S 盒输入之间的关系表。

表 4　PRESENT 算法 S 盒输入特征表

f'		固定输出差分下的输入差分和 S 盒输入之间的关系							
1	f	3	5	7	B	D	F		
	S_{in}	9,10	3,6	0,7,8,15	5,14	1,12	2,4,11,13		
2	f	6	7	9	A	C	D	E	
	S_{in}	8,14	3,4	0,9	5,15	1,13	6,7,10,11	2,12	
3	f	1	2	3	8	A	D	E	
	S_{in}	12,13,14,15	4,6	1,2	3,11	0,11	5,8	7,9	
4	f	3	5	9	B	D	F		
	S_{in}	4,7	8,13	3,5,10,12	0,11	2,15	1,6,9,14		
5	f	2	4	A	C				
	S_{in}	12,13,14,15	0,1,4,5	2,3,8,9	6,7,10,11				
6	f	2	3	4	6	7	9	E	
	S_{in}	9,11	12,13,14,15	3,7	0,6	2,5	1,8	4,10	
7	f	1	3	4	8	C	D	E	
	S_{in}	6,7,10,11	0,3	8,12	5,13	2,14	4,9	1,15	
8	f	6	7	9	A	B	C	E	
	S_{in}	1,7	10,13	6,15	4,14	2,3,8,9	0,12	5,11	
9	f	1	4	5	8	B	C	E	
	S_{in}	0,1,4,5	9,13	11,14	2,10	7,12	3,15	6,8	
A	f	2	3	4	5	A	B	C	D
	S_{in}	0,2	5,6	11,15	9,12	7,13	1,10	4,8	3,14
B	f	3	5	6	8	B	D		
	S_{in}	8,11	2,7	3,5,10,12	1,6,9,14	4,15	0,13		
C	f	2	4	5	6	7	9	E	
	S_{in}	8,10	2,6	0,1,4,5	9,15	11,12	7,14	3,13	
D	f	1	2	5	8	A	B	E	
	S_{in}	2,3,8,9	5,7	10,15	4,12	1,11	6,13	0,14	
E	f	2	4	9	A	C	F		
	S_{in}	1,3	10,14	2,4,11,3	6,12	5,9	0,7,8,15		
F	f	6	7	8	F				
	S_{in}	2,4,11,13	1,6,9,14	0,7,8,15	3,5,10,12				

由表 4 可知,当输出差分固定时,如果对单个 S 盒进行分析,那么集合 M 中会有 16 个不同的值,显然这样的攻击是无效的。针对这一问题,提出如下的解决方法:

①由表 2 可知,若将连续相邻的 4 个 S 盒分为 1 组,则 1 组 S 盒经过 1 轮 P 置换后会影响相同的 4 个 S 盒,因此可以采用并行处理的方式同时处理 4 个 S 盒,减少攻击所需要的时间。

②由上述条件,攻击者对 1 组即 4 个 S 盒同时进行差分分析,根据 4 个 S 盒已知的输出差分(若其中的某些 S 盒的输出差分为 0 则可忽略这些 S 盒),找到候选输入差分交集 G。假设有两类输出差分分别为 1、4,则对应的输入差分候选集为 $F(1)=\{3,5,7,B,D,F\}$,$F(4)=\{3,5,9,B,D,F\}$,交集 $G=\{3,B,D,F\}$。结合表 4、表 5 给出了 $F(1)$ 与其他输入差分候选集 $F(n)$ $(2 \leqslant n \leqslant 15)$ 交集的情况,通过寻找输入差分的交集能够减少 S_{in} 的搜索范围。

表5 局部 S 盒输入特征表

$F(n)$	固定的 $F(1)$ 与 $F(n)$ 的交集和 S 盒输入之间的关系					
2	G S_{in}	7 0,7,8,15	D 1,12			
3	G S_{in}	3 9,10	D 1,12			
4	G S_{in}	3 9,10	5 3,6	B 5,14	D 1,12	F 2,4,11,13
5	G S_{in}					
6	G S_{in}	3 9,10	7 0,7,8,15			
7	G S_{in}	3 9,10	D 1,12			
8	G S_{in}	7 0,7,8,15	B 5,14			
9	G S_{in}	5 3,6	B 5,14			
A	G S_{in}	3 9,10	5 3,6	D 1,12		
B	G S_{in}	3 9,10	5 3,6	B 5,14	D 1,12	
C	G S_{in}	5 3,6	7 0,7,8,15			
D	G S_{in}	5 3,6	B 5,14			
E	G S_{in}	F 2,4,11,13				
F	G S_{in}	7 0,7,8,15	F 2,4,11,13			

③由②可知,已经对 S 盒的输入进行了筛选,若筛选出来的输入差分包含了正确的输入差分值,那么一定可以得到 S 盒的正确输入,式(6)为 S 盒正确输入的概率计算公式。

$$\text{Correct}=(1-\frac{n_2}{n_1})\cdot\left[\frac{\text{sum}_1}{n_1}\right]\cdot\left(\frac{\text{sum}_2}{15}\right)^{-1}\cdot 100\% \qquad (6)$$

其中:n_1 代表遍历不同输出差分组合的数目,n_2 代表 $F(x_1)\cap F(x_2)=\varnothing$ 的输出差分组合数目,sum_1 代表不同输出差分组合下的相同输入差分的数量之和,sum_2 代表不同输出差分下所有的输入差分数量之和。由式(6)可以得到表6,其中 No-Key $=\frac{n_2}{n_1}\cdot 100\%$。

表6 S 盒正确输入概率表 单位:%

Species	1	2	3	4
Correct	100	40.1	10.8	1.5
Fault	0	56.1	53.8	25.8
No-Key	0	3.8	35.4	72.7

表6 中 Species 代表 4 个 S 盒中正确输入差分的种类,Correct 代表当前 S 盒正确输入出现的概率。Fault 代表出现错误输入的概率,No-Key 代表不会得到输入的概率,由表6 可知若 4 个 S 盒中只有一类正确的输入差分,那么正确的输入差分一定包含在 G 中,则一定会出现正确输入。若有两类正确的输入差分,则会有40.1% 的概率出现正确的输入。此时虽然得到错误输入的概率为56.1%,但是在这56.1% 概率中包含了 15 种错误的输入,因此每种错误输入的概率为3.74%,远低于正确输入的概率。同理三类正确输入差分的正确输入概率为10.8%,四类为1.5%。

(5)根据上述内容,对 G 集合中的候选输入差分 diff_{sin} 进行遍历,由式(7)可以找出所有候选的 S 盒输入值 S_{in}。

$$\text{diff}_{p31}=S[S_{\text{in}}\oplus\text{diff}_{\text{sin}}]\oplus S[S_{\text{in}}] \qquad (7)$$

(6)由计算得到的 S_{in},根据式(8)可以恢复出该 S 盒对应的 4 比特位密钥候选值。

$$K_{64}^{31}(4\cdot i+3)\parallel K_{64}^{31}(4\cdot i+2)\parallel K_{64}^{31}(4\cdot i+1)\parallel$$
$$K_{64}^{31}(4\cdot i)=S[S_{in}^{31}]\oplus C_p^{31}(4\cdot i+3)\parallel$$

$$C_p^{31}(4\cdot i+2)\parallel C_p^{31}(4\cdot i+1)\parallel C_p^{31}(4\cdot i) \tag{8}$$

式(8)中的 i 表示第 i 个 S 盒,$K_{64}^{31}(4\cdot i+m)(0\leqslant m\leqslant 3)$ 表示第 31 轮当前 S 盒的 4 比特密钥值,$C_p^{31}(4\cdot i+m)$ 表示当前 S 盒的 4 比特位经过 P 置换后对应的密文值,由此可以得到 4 个候选密钥位。同理,按照式(7)对其他的 S 盒进行相同的运算,即可恢最后一轮的 64 个候选密钥位。通过对多条故障密文进行相同的处理,经过密钥筛选后即可恢复最后一轮的轮子密钥。

(7)候选输入筛选。首先设置一个长度为 16 的数组 key[16][16],并且初值都 0,key[16][16] = {{0,0…0},{0,0…0}…{0,0…0}},数组 key 中的行代表第 0 ~ 16 个 S 盒,列下标表示 16 个可能的输入值。将式(8)计算出的候选输入,按照列下标匹配的方式存入到数组 key 中,并且对应位的数值加 1。通过分析多条故障数据,找到二维数组每一行的最大值,最大值的下标就是当前 S 盒的正确输入,从而恢复出第 31 轮的 64 位轮子密钥 K_{64}^{31}。

步骤三　根据上述恢复出的第 31 轮的轮子密钥,逆推可以得到第 30 轮的密文,重复(1)至(7)的步骤恢复出第 30 轮的轮子密钥 K_{64}^{30}。利用 K_{64}^{30}、K_{64}^{31},通过密钥编排算法恢复出主密钥 K。

步骤四　主密钥密钥恢复。每一轮的主密钥长度为 80bits,记为 $k=[k_{79}k_{78}\cdots k_0]$,轮计数器 r 用二进制表示$(r_4r_3r_2r_1r_0)_2$,由密钥扩展算法可知,K^{30} 经过左移 61 比特位后,此时第 31 轮 80 比特的最后 16 比特位为 $k'=k_{34}k_{33}k_{32}\cdots k_{20}k_{19}$,而该 16 比特位在 K^{30} 的前 64 比特位中,由攻击出的第 30 轮 64 比特密钥 K^{30},可以获得 k'的具体值。经分析经过密钥扩展算法 k'只有一位会产生变化即 $k_{34}=k_{34}\oplus r_0$。根据已经恢复 k^{31} 的 64 比特密钥以及上述推出的 k',可以推出最后一轮的 80 比特主密钥 K_{80}^{31}。

$$K_{80}^{31}=K_{64}^{31}\parallel K_{34}K_{33}K_{32}\cdots K_{20}K_{19} \tag{9}$$

得到最后一轮 80 比特主密钥后根据(10)式推导出 K_{80}^{30}。

$$\begin{cases}[k_{19}k_{18}k_{17}k_{16}k_{15}]=[k_{19}k_{18}k_{17}k_{16}k_{15}]\oplus r\\ [k_{79}k_{78}k_{76}k_{77}]=S^{-1}[k_{79}k_{78}k_{76}k_{77}]\\ [k_{79}k_{78}\cdots k_1k_0]=[k_{60}k_{59}\cdots k_{62}k_{61}]\end{cases} \tag{10}$$

式(10)中第一步为当前主密钥的 $k_{19}k_{18}k_{17}k_{16}k_{15}$ 与轮计数器进行异或,之后进行 S 盒的逆运算,最后右移 61 比特从而得到第 30 轮的主密钥 K_{80}^{30}。同理,可以推出初始的 80 比特主密钥。通过上述对密钥扩展算法的缺陷分析,攻击者不用通过穷举最后16 bits的方式也可恢复主密钥,提高了主密钥恢复效率。

4　攻击实验与分析

使用智能卡、VC Glitcher、Inspector、示波器等设备进行正确密文和故障密文的采集。选择初始明文为 0xc 0x7 0x3 0x9 0xd 0x7 0xe 0xa 0xf 0xa 0xe 0x4 0xe 0xd 0xa 0x3,初始密钥为 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf 0xf,整个实验过程包括信息采集、故障注入、故障分析、复杂度分析以及实验结果分析。

4.1　数据采集

首先采集智能卡在运行时的能量曲线并且记录正确的密文,图 3 为示波器上采集到的能量曲线。

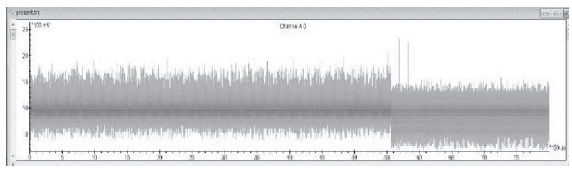

图 3　能量曲线

图 4 为采集到的明文和密文信息,C7 39 D7 EA FA E4 ED A3 为采集到明文信息,35 75 04 0B 4F D5 85 BC 为采集到的正确密文信息,90 00 代表标识位表示智能卡返回数据成功,其余数据代表对智能卡下发的指令。

图 4　正确密文信息

4.2　故障注入

对采集到的能量曲线首先进行信号处理,确定 PRESENT 算法第 29 轮、第 30 轮的时间。由图 3 可知 5000 ~ 5400 μs为第 29 轮、第 30 轮的时间。利用 VC Glitcher 在 PRESENT 算法的第 29 轮、第 30 轮分别进行随机故障注入,故障注入的具体位置和故障注入的字节数未知。图 5 是故障注入后返回的部分故障密文,其中黑框部分为产生的故障密文。

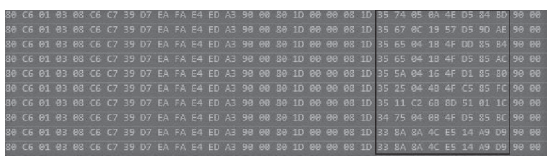

图 5　故障密文

4.3　故障分析

根据攻击方法,首先对第 30 轮的故障密文进行差

分分析,恢复第 31 轮的 16 个 S 盒的密钥,图 6 是实验结果图。

```
<terminated> dispos_data (5) [Java Application] C:\JJ\Java\bin\javaw.exe (2021年4月
16 sbox-key:0x1111
15 sbox-key:0x0001
14 sbox-key:0x1000
13 sbox-key:0x0101

12 sbox-key:0x1011
11 sbox-key:0x1010
10 sbox-key:0x1001
 9 sbox-key:0x1111

 8 sbox-key:0x1100
 7 sbox-key:0x1110
 6 sbox-key:0x1001
 5 sbox-key:0x0111

 4 sbox-key:0x1011
 3 sbox-key:0x1000
 2 sbox-key:0x0100
 1 sbox-key:0x1110
轮子密钥为: 15 14 7 10 5 4 8 15 11 6 0 14 11 1 6 7
程序运行时间: 1017ms
```

图 6 第 30 轮故障密文分析

同理根据第 29 轮的故障密文恢复出第 30 轮的轮子密钥,并且根据两轮轮子密钥恢复初始的80 bits主密钥,图 7 是实验结果图。

```
16 sbox-key:0x0100
15 sbox-key:0x0010
14 sbox-key:0x1100
13 sbox-key:0x1101

12 sbox-key:0x1101
11 sbox-key:0x0010
10 sbox-key:0x1100
 9 sbox-key:0x0010

 8 sbox-key:0x0000
 7 sbox-key:0x1000
 6 sbox-key:0x1110
 5 sbox-key:0x1101

 4 sbox-key:0x1110
 3 sbox-key:0x0011
 2 sbox-key:0x0001
 1 sbox-key:0x1001
轮子密钥为: 10 4 7 13 11 0 7 5 8 11 4 2 2 8 8 13
程序运行时间: 980ms
初始密钥为:15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
```

图 7 第 29 轮故障密文分析

通过式(10)可恢复初始 80bits 主密钥为:15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15。

由上述实验结果可知,轮密钥攻击平均时间为 1000 ms,恢复出的主密钥为 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15。与初始设定的主密钥一致,达到了攻击的目的,证明文中提出攻击方案的有效性。

4.4 复杂度分析

根据故障传播路径对攻击方法进行重新设计,相比于传统的分析方法或者已有针对 PRESENT 算法的差分故障攻击方法具有明显的优势。传统的密钥攻击方法大多数采用的是穷举密钥,由于 PRESENT 算法的轮子密钥长度为 64bits,若采用直接穷举密钥搜索,则轮密钥的攻击复杂度为 2^{64}。而文献[9]则是采用依次扩展的方法,根据每一轮的中间匹配状态来进一步的降低搜索空间,攻击复杂度为 2^{31}。根据 PRESENT 算法的故障传播路径,采用多字节故障模型的方法,通过输入差分的交集来寻找正确密钥。复杂度 C 如下:

$$C=\mathrm{sbox}_{num}\times\mathrm{diff}_{num}\times\mathrm{key}_{num}\times\mathrm{fault}_{num} \tag{11}$$

其中 sbox_{num} 表示 S 盒的数量,值为 16。diff_{num} 代表每个 S 盒输出差分一定的情况下的输入差分数量,经计算分析 diff_{num} 的取值范围为 $4\leqslant\mathrm{diff}_{num}\leqslant8$。$\mathrm{key}_{num}$ 表示每个 S 盒输入差分对应的候选输入的个数,候选输入的个数范围为 $0\leqslant\mathrm{key}_{num}\leqslant4$。$\mathrm{fault}_{num}$ 代表恢复该轮轮子密钥所需要的故障密文数,经过实验分析,平均 300 条故障密文就可以恢复当前的轮子密钥。综上所述,恢复出轮子密钥的复杂度 $C=2^{18}$。

5 结束语

利用优化后的差分故障攻击方法,对写有 PRESENT 算法的智能卡进行随机故障注入,并且对得到的故障数据进行分析,实验结果证明方法具有有效性。与现有的针对 PRESENT 算法的单字节故障注入相比,本文提出的方法没有限定故障注入的位置和故障注入的数量,打破了以往研究对故障诱导的限制,降低了对故障注入设备的要求,为针对 SPN 结构分组密码算法的故障攻击提供了新思路。并且轮密钥的攻击复杂度由 2^{31} 降低到 2^{18},攻击时间降低了19000 ms,效率也得到了极大的提高。

参考文献:

[1] IZADI M, SADEGHIYAN B, SADEGHIAN S S, et al. MIBS: a new lightweight block cipher[C]. 8th International Conference on Cryptology and Network Security. Berlin: Springer, 2009: 334-348.

[2] SUZAKI T, MINEMATSU K, SORIOKA S, et al. TWINE: a lightweight block cipher for multiple platforms[C]. 19th International Conference on Selected Areas in Cryptography. Berlin: Springer, 2012: 339-354.

[3] HONG D, SUNG J, HONG S, et al. HIGHT: a new block cipher suitable for low-resource device[C]. CHES 2006: 46-59.

[4] BANIK S, PANDEY S K, PEYRIN T, et al. Gif: a small present[C]. Proceedings of the 19th International Conference on Cryptographic Hardware and Embedded System. 2017: 321-345.

[5] BOGDANOV A,KNUDSEN L R,LEANDER G,et al. PRESENT: An ultralightweight block cipher [J]. Lecture Notes in Computer Science,2007,4727:450-466.

[6] WANG M Q,SUN Y,SUN N,et al. Algebraic techniques in differential cryptanalysis revisited[C]. Information Security and Privacy. Melbourne,Australia:Springer-Verlag,2011:120-141.

[7] COLLARD B,STANDAERT F X. A statistical saturation attack against the block cipher PRESENT [C]. Proceedings of the Topics in Cryptology. Berlin,Germany:Springer,2009:95-210.

[8] 陈伟建,赵思宇,邹瑞杰. PRESENT 密码的差分故障攻击[J]. 电子科技大学学报. 2019,48(6):865-869.

[9] 李卷孺,谷大武. PRESENT 密码的差分故障攻击[C]. 中国密码学会 2009 年会. 科学出版社,2009:3-13.

[10] Dusart P, Letourneux G, Vivolo O. Differential fault analysis on AES[C]. Applied Cryptography and Network Security. Berlin,Germany:Springer. 2003:293.

[11] Kelsey J,Schneier B,Wagner D,et al. Side channel cryptanalysis of product ciphers[J]. Lecture Notes in Computer Science,2000,8(2):141-158.

[12] BIHAM E,SHAMIR A. Differential cryptanalysis of DES-like cryptosystems[J]. Journal of Cryptology,1991,4(1):3-72.

[13] Bar-El H,Choukri H,Naccache D,et al. The Sorcerer´s Apprentice Guide to Fault Attacks[J]. Proceedings of the IEEE,2006,94(2):370-382.

[14] 高靖哲,赵新杰,矫文成. 针对 CLEFIA 的多字节差分故障分析[J]. 计算机工程,2010,36(19):156-158.

[15] 荣雪芳,吴震,王敏. 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.

Random Differential Fault Attack Against the Lightweight Block Cipher Algorithm PRESENT

HUANG Xiangshu, WANG Min, DU Zhibo, WU Zhen, WANG Yi

(College of Cyberspace Security,Chengdu University of Information Technology,Chengdu 610225,China)

Abstract:The lightweight block cipher algorithm PRESENT uses the SPN network structure, which has the characteristics of small implementation area and low power consumption, so it is widely used in resource-constrained environments. This article designs a multi-byte fault model for the PRESENT algorithm. Random fault injection is performed at any position in the 30th and 29th rounds of the PRESENT algorithm, and the number of bytes injected is not fixed. Using the fault propagation path of the PRESNET algorithm, the relationship between the output difference and the possible input value is constructed, and the correct input is obtained through the parallel S-box analysis method proposed in this paper, and then the correct wheel key is obtained. Finally, by analyzing the key arrangement scheme, only two rounds of the correct wheel keys are needed to derive the initial 80 bits master key. The experimental results show that, compared with the existing fault attacks against PRESENT algorithm, the use of the fault model in this paper can reduce the complexity of the attack from 2^{31} to 2^{18}, and the average duration of the round key attack can be reduced from 20000 ms to 1000 ms. At the same time, the method proposed in this paper improves the single-byte, fixed-location fault model to a multi-byte, arbitrary-location fault model, which is more in line with the actual attack situation, reduces the requirements for fault injection equipment, and improves the practicability of the method .

Keywords:PRESENT algorithm; multibyte fault model; random fault injection; fault propagation path; parallel S-box analysis