FeW 的差分故障攻击

谢敏，李嘉琪，田峰
（西安电子科技大学综合业务网理论及关键技术国家重点实验室，陕西 西安 710071）

摘 要：为了评估轻量级分组密码算法 FeW 的安全性，提出并讨论了一种针对 FeW 算法的差分故障攻击方法。该方法采用单字节随机故障模型，选择在 FeW 算法的最后一轮右侧引入单字节随机故障，利用线性扩散函数的特点获取差分信息，并基于 S 盒差分分布统计规律实现密钥恢复。实验结果表明，平均 47.73 次和 79.55 次故障注入可以分别完全恢复 FeW-64-80 和 FeW-64-128 的主密钥，若在恢复密钥过程中加入 2^{10} 的穷举计算，所需平均故障注入次数分别降至 24.90 和 41.50。该方法可以有效地攻击 FeW 算法。

关键词：FeW 算法；轻量级分组密码；差分故障攻击；单字节故障模型
中图分类号：TP309.2
文献标识码：A
doi: 10.11959/j.issn.1000−436x.2020077

Differential fault attack on FeW

XIE Min, LI Jiaqi, TIAN Feng
State Key Laboratory of Integrated Services Networks, Xidian University, Xi’an 710071, China

Abstract: In order to evaluate the security of the lightweight block cipher FeW, a differential fault attack method was proposed and discussed using a single byte random fault model. In this method, a single byte random fault was introduced on the right side of the last round of FeW to recover the key based on the statistical characteristics of S-box difference distribution, and the difference information was obtained using the characteristics of the linear diffusion function.The experiment results show that the complete key recovery can be achieved with an average of 47.73 and 79.55 fault injections for FeW-64-80 and FeW-64-128 respectively. If 2^{10} exhaustive calculations are added to the key recovery process, the number of average fault injections required can be reduced to 24.90 and 41.50.This attack is effective on FeW.

Key words: FeW algorithm, lightweight block cipher, differential fault attack, single byte fault model

1 引言

近年来，随着物联网的快速发展，轻量级分组密码算法相关的研究十分热门。传统密码分析方法在轻量级算法研究中取得了不错的成果，如在 MIBS[1]、LBlock[2]、TWINE[3]等算法的分析中，利用多种分析方法相结合的方式获得更高轮数的攻击[4-6]。与传统密码分析方法不同，差分故障攻击（DFA, differential fault attack）[7]利用加密算法的特点和最大似然估计技术来推测加密系统中的关键信息，其密钥搜索空间远小于差分密码分析和线性密码分析等方法。故障攻击[8]自提出后一直不断发展和改进，差分故障攻击作为故障攻击的一种方法，对轻量级密码算法构成严重威胁[9-11]。

FeW 是 Kumar 等[12]在 2014 年提出并于 2019 年正式发表的一种轻量级分组密码算法，它可以在规格很小的硬件和微型控制器上实现。此外，Kumar 等[13]还提出了一种基于 FeW 的哈希函数 HeW，它

收稿日期：2020−01−08；**修回日期**：2020−03−23
基金项目：国家重点研发计划基金资助项目（No.2018YFE0126000）；国家自然科学基金资助项目（No.U1636209）；陕西省重点研发计划基金资助项目（No.2019ZDLGY13-07, No.2019ZDLGY13-04）
Foundation Items: The National Key Research and Development Program of China (No.2018YFE0126000), The National Natural Science Foundation of China (No.U1636209), The Key Research and Development Program of Shaanxi Province (No.2019ZDLGY13-07, No.2019ZDLGY13-04)

在软硬件实现上都拥有很高的效率。FeW 的设计采用了广义 Feistel 结构，分组长度为 64 bit，轮数为 32 轮，密钥长度分为 80 bit 和 128 bit 这 2 个版本，分别记为 FeW-64-80 和 FeW-64-128。算法结构中包含半字节置换、非线性层和线性层。FeW 对于差分、不可能差分、线性、零相关、相关密钥等分析方法都具有良好的安全性。Shibayama 等[14]利用高阶差分攻击了 12 轮 FeW-64-80，时间复杂度和数据复杂度均为 $2^{63.2}$。Aayush 等[15]使用神经网络模式识别的技术分析了 FeW 算法，实验结果表明，FeW 算法的遗传偏差很小，能有效抵御该类攻击。但是目前还没有人评估过 FeW 面对差分故障攻击时的安全性，因此对 FeW 进行差分故障攻击是十分必要的。

2 FeW 算法介绍

2.1 FeW 的加密算法

FeW 算法的分组长度为 64 bit，密钥长度分为 80 bit 和 128 bit 这 2 种，明文经过 32 轮迭代得到密文。该算法采用了广义 Feistel 结构，如图 1 所示，其中使用到的 S 盒如表 1 所示。

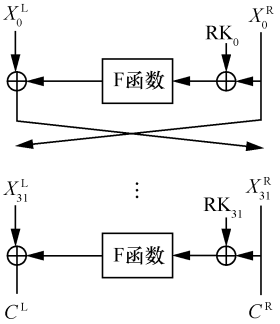

图 1 FeW 的轮结构

表 1 FeW 的 S 盒

x	$S(x)$	x	$S(x)$
0	2	8	B
1	E	9	4
2	F	10	6
3	5	11	B
4	C	12	0
5	1	13	7
6	9	14	3
7	A	15	D

2.2 FeW 的 F 函数

FeW 的 F 函数如图 2 所示，它由 P 置换（如表 2 所示）、8 个 4 bit 的 S 盒以及 2 个 16 bit 的线性扩散函数 L_1 和 L_2 构成。线性扩散函数 L_1 和 L_2 分别为

$$L_1(x)=x\oplus(x\lll 1)\oplus(x\lll 5)\oplus(x\lll 9)\oplus(x\lll 12) \tag{1}$$

$$L_2(x)=x\oplus(x\lll 4)\oplus(x\lll 7)\oplus(x\lll 11)\oplus(x\lll 15) \tag{2}$$

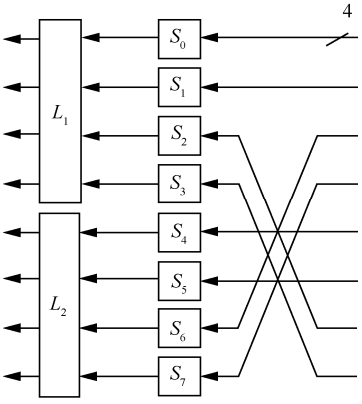

图 2 FeW 的 F 函数

表 2 P 置换

x	$P(x)$
0	0
1	1
2	6
3	7
4	4
5	5
6	2
7	3

2.3 FeW 的密钥编排算法

2.3.1 FeW-64-80 的密钥编排算法

FeW-64-80 的 80 bit 主密钥存储在名为 MK = $(K_0K_1\cdots K_{78}K_{79})$ 的特定寄存器中，并由其更新算法进行 64 轮计算，每次取最左侧 16 bit 为密钥编排子密钥 k_i，每 2 次更新计算所得共计 32 bit 作为轮密钥 RK_i。当 $i<64$ 时，MK 的更新算法执行以下计算。

步骤 1 MK $\lll 13$。

步骤 2 $[K_0K_1K_2K_3]\leftarrow S[K_0K_1K_2K_3]$。

$[K_{64}K_{65}K_{66}K_{67}]\leftarrow S[K_{64}K_{65}K_{66}K_{67}]$。

$[K_{76}K_{77}K_{78}K_{79}]\leftarrow S[K_{76}K_{77}K_{78}K_{79}]$。

步骤 3 $[K_{68}K_{69}K_{70}K_{71}K_{72}K_{73}K_{74}K_{75}]\leftarrow[K_{68}K_{69}K_{70}K_{71}K_{72}K_{73}K_{74}K_{75}]\oplus[i]_2$。

步骤 4　取最左 16 bit 为 k_i，更新 $i=i+1$。

2.3.2　FeW-64-128 的密钥编排算法

FeW-64-128 的 128 bit 主密钥存储在名为 $\mathrm{MK}=(K_0K_1\cdots K_{126}K_{127})$ 的特定寄存器中，并由其更新算法进行 64 轮计算，每次取最左侧 16 bit 为密钥编排子密钥 k_i，每 2 次更新计算所得共计 32 bit 作为轮密钥 RK_i。当 $i<64$ 时，MK 的更新算法执行以下计算。

步骤 1　$\mathrm{MK}<<<13$。

步骤 2　$[K_0K_1K_2K_3]\leftarrow S[K_0K_1K_2K_3]$。

$[K_4K_5K_6K_7]\leftarrow S[K_4K_5K_6K_7]$。

$[K_{112}K_{113}K_{114}K_{115}]\leftarrow S[K_{112}K_{113}K_{114}K_{115}]$。

$[K_{124}K_{125}K_{126}K_{127}]\leftarrow S[K_{124}K_{125}K_{126}K_{127}]$。

步骤 3　$[K_{116}K_{117}K_{118}K_{119}K_{120}K_{121}K_{122}K_{123}]\leftarrow[K_{116}K_{117}K_{118}K_{119}K_{120}K_{121}K_{122}K_{123}]\oplus[i]_2$。

步骤 4　取最左 16 bit 为 k_i，更新 $i=i+1$。

3　FeW 的 DFA 过程

3.1　故障模型

本文用到的符号说明如下。

i：轮数，$0\leqslant i\leqslant 31$。

$X_i^{\mathrm{L}}=(X_{i,0}^{\mathrm{L}},X_{i,1}^{\mathrm{L}},\cdots,X_{i,7}^{\mathrm{L}})$：第 i 轮左半段 32 bit 输入的半字节块表示。

$X_i^{\mathrm{R}}=(X_{i,0}^{\mathrm{R}},X_{i,1}^{\mathrm{R}},\cdots,X_{i,7}^{\mathrm{R}})$：第 i 轮右半段 32 bit 输入的半字节块表示。

$P=(X_0^{\mathrm{L}},X_0^{\mathrm{R}})$，$C=(C^{\mathrm{L}},C^{\mathrm{R}})$：明文，密文。

$C^{\mathrm{R}}=(C_0^{\mathrm{R}},C_1^{\mathrm{R}},\cdots,C_7^{\mathrm{R}})$：密文右半段 32 bit 的半字节块表示。

$C^*=(C^{\mathrm{L}^*},C^{\mathrm{R}^*})$：故障注入后加密得到的密文。

$\Delta C=(\Delta C^{\mathrm{L}},\Delta C^{\mathrm{R}})=(C^{\mathrm{L}}\oplus C^{\mathrm{L}^*},C^{\mathrm{R}}\oplus C^{\mathrm{R}^*})$：故障注入前后所得密文差分。

$\mathrm{RK}_i=(\mathrm{RK}_{i,0},\mathrm{RK}_{i,1},\cdots,\mathrm{RK}_{i,7})$：第 i 轮 32 bit 轮密钥的半字节块表示。

S_k：第 k 个 S 盒，$0\leqslant k\leqslant 7$。

$\mathrm{IN}=(\mathrm{in}_0,\mathrm{in}_1,\cdots,\mathrm{in}_7)$：第 31 轮中 8 个 S 盒 32 bit 输入的半字节块表示。

$\Delta\mathrm{IN}=(\Delta\mathrm{in}_0,\Delta\mathrm{in}_1,\cdots,\Delta\mathrm{in}_7)$：第 31 轮中 8 个 S 盒 32 bit 输入差分的半字节块表示。

$\mathrm{OUT}=(\mathrm{out}_0,\mathrm{out}_1,\cdots,\mathrm{out}_7)$：第 31 轮中 8 个 S 盒 32 bit 输出的半字节块表示。

$\Delta\mathrm{OUT}=(\Delta\mathrm{out}_0,\Delta\mathrm{out}_1,\cdots,\Delta\mathrm{out}_7)=(y_0y_1y_2y_3\cdots y_{30}y_{31})$：第 31 轮中 8 个 S 盒 32 bit 输出差分的半字节块表示和比特表示。

$x=(x_0x_1\cdots x_{15})$：线性扩散函数 L_1 的 16 bit 输入的比特表示。

本文采用的故障模型包括以下 2 个基本假设。

假设 1　攻击者可以选择需要的明文，并且可以获得相应的正确密文和错误密文，即选择明文攻击（CPA, chosen plaintext attack）。

假设 2　攻击者可以在加密过程中的某一轮导入单字节故障，但是故障导入的具体位置和故障值是未知的。

3.2　攻击过程

3.2.1　轮密钥恢复

FeW-64-80 和 FeW-64-128 具有相同的轮结构、迭代次数和分组长度，因此以下分析过程对它们均完全适用。

假设故障发生在 X_{31}^{R} 的首字节 $X_{31,0}^{\mathrm{R}}$ $X_{31,1}^{\mathrm{R}}$ 上，则有如图 3 所示的故障传播路径。

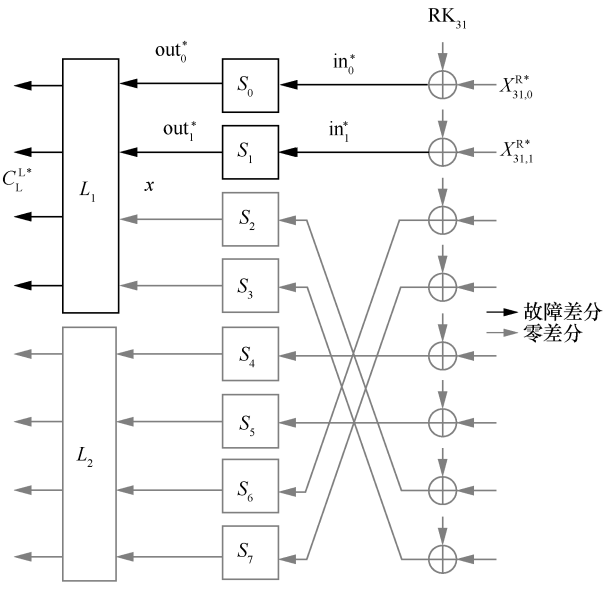

图 3　故障传播路径

由图 3 可知，该单字节故障经过线性扩散函数 L_1 后会扩散到两字节的密文，即影响密文 C^{L^*} 中最左侧 16 bit 的 $C_{\mathrm{L}}^{\mathrm{L}^*}$。对于第 31 轮右半段的输入 X_{31}^{R} 和 S 盒的输入 IN，有 $\mathrm{IN}=\mathrm{RK}_{31}\oplus X_{31}^{\mathrm{R}}$，根据 FeW 算法最后一轮的结构易知 $X_{31}^{\mathrm{R}}=C^{\mathrm{R}}$，因此一旦确定了 S 盒的输入 IN，也就确定了轮密钥 RK_{31}。

下面，利用算法结构推导出第 31 轮 S 盒的输入 IN。

1) 计算 S_0 的输入差分 Δin_0 和 S_1 的输入差分 Δin_1

$$\Delta \text{in}_0 = \text{in}_0 \oplus \text{in}_0^* = (X_{31,0}^{\text{R}} \oplus \text{RK}_{31,0}) \oplus (X_{31,0}^{\text{R}\,*} \oplus \text{RK}_{31,0}) = X_{31,0}^{\text{R}} \oplus X_{31,0}^{\text{R}\,*} = C_0^{\text{R}} \oplus C_0^{\text{R}*} \quad (3)$$

同理可得，$\Delta \text{in}_1 = C_1^{\text{R}} \oplus C_1^{\text{R}*}$。

2) 计算 S_0 的输出差分 Δout_0 和 S_1 的输出差分 Δout_1

由图 3 可知，故障传播到 C^{L} 处时将影响它最左侧的 16 bit 密文，设这 16 bit 密文差分为 $\Delta C_{\text{L}}^{\text{L}} = (c_0 c_1 c_2 \cdots c_{14} c_{15})$，则

$$\Delta C_{\text{L}}^{\text{L}} = C_{\text{L}}^{\text{L}} \oplus C_{\text{L}}^{\text{L}*} = L_1(x) \oplus L_1(x^*) = L_1(x \oplus x^*) \quad (4)$$

又由 S_2 和 S_3 不是活动 S 盒可知

$$x \oplus x^* = (\Delta \text{out}_0, \Delta \text{out}_1, \Delta \text{out}_2, \Delta \text{out}_3) = (y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 00000000) \quad (5)$$

从而有

$$\begin{aligned}(c_0 c_1 c_2 \cdots c_{14} c_{15}) = {} & (y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 00000000) \oplus \\ & (y_1 y_2 y_3 y_4 y_5 y_6 y_7 00000000 y_0) \oplus \\ & (y_5 y_6 y_7 00000000 y_0 y_1 y_2 y_3 y_4) \oplus \\ & (0000000 y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 0) \oplus \\ & (0000 y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 0000)\end{aligned} \quad (6)$$

由此，可得关于未知量 $y_0, y_1, \cdots, y_7$ 的 16 个方程，即

$$\begin{aligned}
c_0 &= y_0 \oplus y_1 \oplus y_5 \\
c_1 &= y_1 \oplus y_2 \oplus y_6 \\
c_2 &= y_2 \oplus y_3 \oplus y_7 \\
c_3 &= y_3 \oplus y_4 \\
c_4 &= y_4 \oplus y_5 \oplus y_0 \\
c_5 &= y_5 \oplus y_6 \oplus y_1 \\
c_6 &= y_6 \oplus y_7 \oplus y_2 \\
c_7 &= y_7 \oplus y_0 \oplus y_3 \\
c_8 &= y_1 \oplus y_4 \\
c_9 &= y_2 \oplus y_5 \\
c_{10} &= y_3 \oplus y_6 \\
c_{11} &= y_0 \oplus y_4 \oplus y_7 \\
c_{12} &= y_1 \oplus y_5 \\
c_{13} &= y_2 \oplus y_6 \\
c_{14} &= y_3 \oplus y_7 \\
c_{15} &= y_0 \oplus y_4
\end{aligned} \quad (7)$$

求解上述方程组，可得 S_0 和 S_1 的输出差分为

$$\begin{aligned}
y_0 &= c_3 \oplus c_{14} \oplus c_{11} \\
y_1 &= c_8 \oplus c_3 \oplus c_{14} \oplus c_{15} \oplus c_{11} \\
y_2 &= c_{13} \oplus c_{10} \oplus c_{14} \oplus c_{15} \oplus c_{11} \\
y_3 &= c_{14} \oplus c_{15} \oplus c_{11} \\
y_4 &= c_3 \oplus c_{14} \oplus c_{15} \oplus c_{11} \\
y_5 &= c_9 \oplus c_{13} \oplus c_{10} \oplus c_{14} \oplus c_{15} \oplus c_{11} \\
y_6 &= c_{10} \oplus c_{14} \oplus c_{15} \oplus c_{11} \\
y_7 &= c_{15} \oplus c_{11}
\end{aligned} \quad (8)$$

若故障发生在 X_{31}^{R} 的末字节，活动 S 盒将变为 S_2 和 S_3，此时可以得到关于 $y_8, y_9, \cdots, y_{15}$ 的 16 个方程，进而求解出 S_2 和 S_3 的输出差分。

对于发生在 X_{31}^{R} 第三个字节的故障，此时的活动 S 盒为 S_4 和 S_5。由算法的结构可知，故障经 L_2 计算后会扩散到 $C^{\text{L}*}$ 右侧的 16 bit 密文 $C_{\text{R}}^{\text{L}*}$，令该 16 bit 密文差分 $\Delta C_{\text{R}}^{\text{L}} = (c_{16}, c_{17}, \cdots, c_{31})$，对于此时的未知量 $y_{16}, y_{17}, \cdots, y_{23}$，同理可得

$$\begin{aligned}
y_{16} &= c_{28} \oplus c_{31} \\
y_{17} &= c_{28} \oplus c_{31} \oplus c_{29} \oplus c_{16} \\
y_{18} &= c_{28} \oplus c_{31} \oplus c_{26} \oplus c_{29} \oplus c_{16} \oplus c_{20} \\
y_{19} &= c_{28} \oplus c_{31} \oplus c_{26} \oplus c_{29} \oplus c_{16} \oplus c_{20} \oplus c_{27} \oplus c_{21} \\
y_{20} &= c_{28} \oplus c_{31} \oplus c_{26} \oplus c_{29} \oplus c_{16} \\
y_{21} &= c_{28} \oplus c_{31} \oplus c_{26} \oplus c_{29} \oplus c_{16} \oplus c_{20} \oplus c_{27} \\
y_{22} &= c_{28} \oplus c_{26} \oplus c_{29} \oplus c_{16} \oplus c_{20} \oplus c_{27} \oplus c_{21} \\
y_{23} &= c_{26} \oplus c_{29}
\end{aligned} \quad (9)$$

若故障发生在 X_{31}^{R} 的第二个字节，活动 S 盒将变为 S_6 和 S_7，此时可以得到关于 $y_{24}, y_{25}, \cdots, y_{31}$ 的 16 个方程，进而求解出 S_6 和 S_7 的输出差分。

综上，在第 31 轮引入一次单字节故障，总能够获得该轮 2 个活动 S 盒的输出差分。

3) 求 S 盒输入 IN 及轮密钥

利用 FeW 算法 S 盒的差分特性来确定相应活动 S 盒的输入，为此给出了 S 盒输入输出差分对应的所有可能输入，如表 3 所示。由表 3 可知，对任意一对可能的输入输出差分，FeW 算法 S 盒的所有可能输入至多有 4 个。设第 31 轮 8 个 S 盒的输入 $\text{in}_0, \text{in}_1, \cdots, \text{in}_7$ 的候选值集合分别为 $E_0, E_1, \cdots, E_7$，候选值个数分别为 $n_0, n_1, \cdots, n_7$。首次故障引入可以使其中 2 个 S 盒的输入候选值个数减少到 2 个或 4 个。采用 2 种方式来确定最终的 IN。

①多次引入故障直至 IN 确定

一次故障引入可以得到某 2 个 S 盒的输入 in_α、

in_β 的候选值集合 O_α、O_β，其中 $\alpha,\beta \in \{0,1,2,3,4,5,6,7\}$，将集合 O_α、O_β 与 E_α、E_β 分别取交集得到新的 E_α 和 E_β，则 n_α 和 n_β 随之减小。通过不断引入新的故障，直到所有 S 盒输入候选值个数 $n_0,n_1,\cdots,n_7$ 都为 1，则此时 $E_0,E_1,\cdots,E_7$ 中唯一的元素即可确定第 31 轮 S 盒的输入 IN。

②利用故障攻击完成初筛，然后通过穷举计算确定最终的 IN。

由表 3 知，对于任意 in_α，首次故障引入将以较大概率使 E_α 中的元素减少到 2 个。一旦在 X_{31}^{R} 所有位置完成首次故障引入，即为初筛完成，此时剩余的候选值将非常少，可以通过少量的加密验证来

表 3　　**固定输入差分下输出差分与输入的对应关系**

Δin	Δout	in	Δin	Δout	in	Δin	Δout	in
	3	6,7		2	1,4		1	4,F
	7	C,D		3	0,5		2	5,E
	A	2,3		4	9,C		8	1,A
1	C	0,1	5	5	2,7	B	A	0,B,7,C
	D	4,5		B	A,F,B,E		B	2,9
	E	A,B,E,F		C	3,6,8,D		E	3,8,6,D
	F	8,9						
	3	C,E		1	A,D		2	0,C,7,B
	5	4,6		6	8,F		5	5,9
	A	D,F		7	1,6,9,E		7	4,8
2	B	1,3,5,7	7	8	0,7,B,C	C	8	3,F
	C	9,B		9	3,4		9	1,D
	D	0,2,8,A		14	2,5		C	2,E
							F	6,A
	1	1,2		6	5,D		1	6,B
	2	9,A		7	7,F		2	2,F
	3	8,B		9	0,8,2,A		5	0,D
	4	D,E		A	1,9,6,E		6	3,E
3	6	4,7	8	C	4,C	D	8	4,9
	7	0,3		D	3,B		A	5,8
	8	5,6					C	7,A
	D	C,F					E	1,C
	3	9,D		1	5,C		1	0,E
	5	A,E,B,F		3	3,A		2	3,D,6,8
	6	2,6		4	6,F		3	1,F
	B	8,C		5	1,8		9	5,B
4	E	0,4	9	6	0,9	E	A	4,A
	F	1,5,3,7		7	2,B		E	7,9
				9	7,E		F	2,C
				11	4,D			
	2	1,4		1	3,9		1	7,8
	3	0,5		4	0,A,2,8		4	4,B
	4	9,C		6	1,B		5	3,C
5	5	2,7	A	9	6,C	F	7	5,A
	B	A,F,B,E		C	5,F		8	2,D
	C	3,6,8,D		D	7,D		D	1,E,6,9
				F	4,E		F	0,F

确定最终的 IN 。

设初筛完成后 $\mathrm{in}_0,\mathrm{in}_1,\cdots,\mathrm{in}_7$ 的候选值个数分别为 $m_0,m_1,\cdots,m_7$ ，此时 IN 的可能值共有 $T=m_0m_1\cdots m_7$ 种，故所需加密验证次数最多为 $T=m_0m_1\cdots m_7$ 次。

确定 IN 的值后，即可恢复轮密钥 $\mathrm{RK}_{31}=\mathrm{IN}\oplus C^{\mathrm{R}}$ 。

3.2.2 主密钥恢复

根据 FeW 的密钥编排算法，攻击者若能获得主密钥寄存器 MK 的某一轮全部位置的比特值，就能通过密钥编排算法向上逆推最终获得全部主密钥。FeW-64-80 和 FeW-64-128 具有不同的密钥编排算法，下面分别给出它们的主密钥恢复过程。

对于 FeW-64-80，从其密钥编排算法的最后一轮向上逆推，易知密钥编排算法第 58 轮 MK 的值仅由 RK_{31} 的 25 bit、RK_{30} 的 26 bit 和 RK_{29} 的 29 bit 共同决定。利用 3.2.1 节中差分故障攻击获得 RK_{31} 后，向上解密一轮，在 X_{30}^{R} 注入故障，可恢复 RK_{30}，之后再向上解密一轮，用同样的方式可获得 RK_{29}，在此 3 轮轮密钥的基础上，利用密钥编排算法的逆算法即可完全恢复 FeW-64-80 的 80 bit 主密钥。

同理，对于 FeW-64-128，易知其密钥编排算法第 54 轮 MK 的值仅由 RK_{31} 的 22 bit、RK_{30} 的 26 bit、RK_{29} 的 26 bit、RK_{28} 的 26 bit 和 RK_{27} 的 29 bit 共同决定，利用 3.2.1 节中差分故障攻击获得该 5 轮轮密钥，即可完全恢复 FeW64-128 的 128 bit 主密钥。

3.3 实验仿真结果和分析

3.3.1 实验环境

硬件配置为一台 PC 机（CPU 为 Intel Core i5-7300HQ 2.5 GHz，操作系统 64 位，内存 16 GB），编程环境为 Eclipse 平台下的 Java 语言。

3.3.2 实验设计

恢复最后一轮轮密钥为一次完整实验，该实验过程对密钥编排算法不同的 FeW-64-80 和 FeW-64-128 没有区别，但本文仍分为 FeW-64-80 和 FeW-64-128 这 2 组实验，其中明文、主密钥、单字节故障位置及故障差分值均完全随机选取。每次故障引入后，执行以下过程。

1) 由 3.2 节所示的攻击过程，完成对 IN 候选值集合 $E_0,E_1,\cdots,E_7$ 的筛选，故障次数加 1。

2) 判断 IN 候选值个数 $n_0,n_1,\cdots,n_7$ 是否均小于 16，若不是，则直接返回步骤 1)继续执行；若是，则对 IN 恰好完成初筛，记录该故障次数为完成初筛所用的故障数，并根据此时 IN 候选值个数 $n_0,n_1,\cdots,n_7$ 计算得到本次实验所需的加密验证次数 $T=n_0n_1\cdots n_7$，完成初筛后不再执行此判断。

3) 判断 $n_0,n_1,\cdots,n_7$ 是否均为 1，若不全为 1，则继续引入新的故障；若全为 1，记录此时的故障次数即为恢复轮密钥所需的故障次数，一次实验结束。

3.3.3 实验结果

表 4 展示了针对 FeW-64-80 和 FeW-64-128 这 2 种算法各 2 组（共 4 组）一万次实验的结果。由表 4 可知，不同的密钥编排算法对单轮轮密钥恢复没有影响。因此，将 4 组结果相结合作为 FeW 算法单轮差分故障攻击的结论，即平均 15.91 次故障引入可以恢复 32 bit 轮密钥；若采用初筛后穷举的方式，则需要平均 8.30 次故障引入和 199 次加密验证才可以恢复 32 bit 轮密钥。

表 4 2 组一万次模拟攻击结果

算法	实验	恢复轮密钥平均所需故障/次	完成初筛平均所需故障/次	单轮加密验证平均次数/次
FeW-64-80	第一组	15.90	8.33	191
	第二组	15.84	8.31	209
FeW-64-128	第一组	15.98	8.29	195
	第二组	15.92	8.27	201

图 4 展示了 FeW 一万次实验中恢复轮密钥所需故障次数的分布情况。结果表明，大多数情况下 20 次以内的故障注入即可恢复轮密钥。

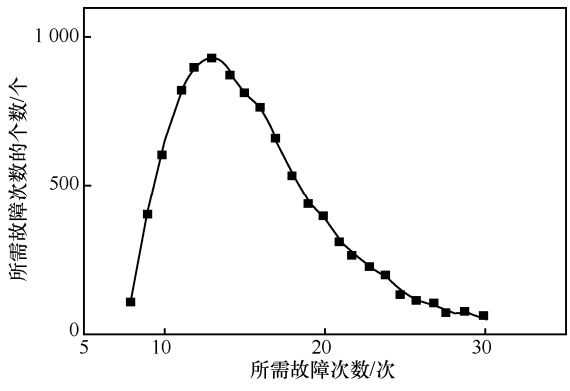

图 4 Few 一万次实验中恢复轮密钥所需故障次数的分布情况

根据实验结果可知，对于 FeW-64-80，平均需要 15.91×3=47.73 次故障引入才可以直接恢复主密钥。若采用初筛后穷举的方式，则进行 199×3=597，约 2^{10} 次加密验证才可将故障次数降为 8.30×3=24.90 次。

对于 FeW-64-128，平均需要 15.91×5=79.55 次故障引入才可以直接恢复主密钥。若采用初筛后穷举的方式，则进行 199×5=995，约 2^{10} 次加密验证才可将故障次数降为 8.30×5=41.50 次。

4　结束语

本文利用 FeW 算法线性扩散函数的特点，采用差分故障对其进行了攻击。通过在第 31 轮右侧输入引入单字节随机故障，较好地完成了 FeW 的差分故障攻击。实验结果表明，平均需要 15.91 次故障可恢复 FeW 单轮轮密钥，根据 FeW 的密钥编排算法，恢复 80 bit 主密钥和 128 bit 主密钥分别需要 3 轮和 5 轮轮密钥；此外，由于故障引入恢复效率的变化，本文提出了一种初筛加穷举的密钥恢复策略，结合 2^{10} 的穷举搜索，可将主密钥恢复平均所需故障引入次数降低一半，为同类攻击提供了新的思路。差分故障攻击对 FeW 是十分有效的，为了避免此类攻击，需要对加密设备进行保护。

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] WU W L, ZHANG L. LBlock: a lightweight block cipher[C]//9th International Conference on Applied Cryptography and Network Security. Berlin: Springer, 2011: 327-344.

[3] 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.

[4] 陈平, 廖福成, 卫宏儒. 对轻量级密码算法 MIBS 的相关密钥不可能差分攻击[J]. 通信学报, 2014, 35(2): 190-193.
CHEN P, LIAO F C, WEI H R. Related-key impossible differential attack on a lightweight block cipher MIBS[J]. Journal on Communications, 2014, 35(2): 190-193.

[5] 谢敏, 田峰, 李嘉琪. TWINE 算法的相关密钥不可能飞来去器攻击[J]. 通信学报, 2019, 40(9): 184-192.
XIE M, TIAN F, LI J Q. Related-key impossible boomerang cryptanalysis on TWINE[J]. Journal on Communications, 2019, 40(9): 184-192.

[6] 谢敏, 牟彦利. LBlock 算法的相关密钥不可能飞来去器分析[J]. 通信学报, 2017, 38(5): 66-71.
XIE M, MU Y L. Related-key impossible boomerang cryptanalysis on LBlock[J]. Journal on Communications, 2017, 38(5): 66-71.

[7] BIHAM E, SHAMIR A. Differential fault analysis of secret key cryptosystem[C]//Proceedings of the CRYPTO 1997. California: Santa Barbara, 1997: 513-525.

[8] SIKHAR P, DEBDEEP M. Fault tolerant architectures for cryptography and hardware security[M]. Berlin: Springer, 2018.

[9] JAP D, BREIER J. Differential fault attack on LEA[C]//Information and Communication Technology-EurAsia Conference. Berlin: Springer, 2015: 265-274.

[10] MORADI A, SHALMANI M, SALMASIZADEH M. A generalized method of differential fault attack against AES cryptosystem[C]//Cryptographic Hardware and Embedded Systems-CHES 2006. Berlin: Springer, 2006: 91-100.

[11] 高杨, 王永娟, 王磊, 等. 轻量级分组密码算法TWINE差分故障攻击的改进[J]. 通信学报, 2017, 38(Z2): 178-184.
GAO Y, WANG Y J, WANG L, et al. Improvement differential fault attack on TWINE[J]. Journal on Communications, 2017, 38(Z2): 178-184.

[12] KUMAR M, PAL S, PANİGRAHİ A. FeW: a lightweight block cipher[J]. Turkish Journal of Mathematics and Computer Science, 2019, 11(2): 73-58.

[13] KUMAR K, DEY D, PAL S, et al. HeW: a hash function based on lightweight block cipher FeW[J]. Defence Science Journal, 2017, 67(6): 636-664.

[14] SHIBAYAMA N, IGARASHI Y, KANEKO T. A new higher order differential of FeW[C]//Sixth International Symposium on Computing and Networking Workshops (CANDARW). LoS Alamitos: IEEE Computer Society, 2018: 466-471.

[15] AAYUSH J, GIRISH M. Analysis of lightweight block cipher FeW on the basis of neural network[C]//Harmony Search and Nature Inspired Optimization Algorithms. Berlin: Springer, 2018: 1041-1047.

[作者简介]

谢敏（1976− ），女，湖南桃源人，博士，西安电子科技大学副教授，主要研究方向为编码与密码。

李嘉琪（1993− ），男，陕西榆林人，西安电子科技大学硕士生，主要研究方向为分组密码算法的分析。

田峰（1995− ），男，河南安阳人，西安电子科技大学硕士生，主要研究方向为分组密码算法的分析。