基于持续性故障的分组密码算法S盒表逆向分析

王 安[1,2],谷 睿[1],丁瑶玲[1,3,4],张 雪[5],袁庆军[4],祝烈煌[1]
(1. 北京理工大学网络空间安全学院,北京 100081; 2. 密码科学技术国家重点实验室,北京 100878;
3. 中国科学院信息工程研究所信息安全国家重点实验室,北京 100093;
4. 战略支援部队信息工程大学河南省网络密码技术重点实验室,河南郑州 450001; 5. 清华大学高等研究院,北京 100084)

摘 要: 基于故障注入的逆向分析技术通过向运行保密算法的设备中注入故障,诱导异常加密结果产生,进而恢复保密算法内部结构和参数. 在除S盒表外其他运算结构已知的前提下,本文基于持续性故障提出了一种分组密码算法S盒表逆向分析方法. 我们利用算法中使用故障元素的S盒运算将产生错误中间状态并导致密文出错这一特点,构造特殊的明文和密钥,诱导保密算法第二轮S盒运算取到故障值,从而逆向推导出第一轮S盒运算的输出,进而恢复出保密算法S盒表的全部元素. 以类AES-128(Advanced Encryption Standard-128)算法为例,我们的方法以1 441 792次加密运算成功恢复出完整S盒表,与现有的其他逆向分析方法进行对比,新方法在故障注入次数和计算复杂度上有明显优势. 进一步,我们将该方法应用于类SM4算法,并以1 900 544次加密运算恢复出保密S盒表. 最后,我们综合考虑了分组密码算法的两种典型结构Feistel和SPN(Substitution Permutation Network)的特点,对新方法的普适性进行了讨论,总结出适用算法需具备的条件.

关键词: 逆向分析;持续性故障;分组密码;S盒表

基金项目: 国家重点研发计划(No.2021YFB3101500),国家自然科学基金(No.62002021);河南省网络密码技术重点实验室研究课题(No.LNCT2020-A09);北京理工大学青年教师学术启动计划;新能源和智能网联汽车行业密码应用产业链供需对接平台项目(No.2021-0181-1-1)

中图分类号: TN918;TP309 **文献标识码:** A **文章编号:** 0372-2112(2023)03-0537-15

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20211032

Reverse-Engineering Secret S-box of Block Ciphers by Persistent Fault

WANG An[1,2], GU Rui[1], DING Yao-ling[1,3,4], ZHANG Xue[5], YUAN Qing-jun[4], ZHU Lie-huang[1]
(1. *School of Cyberspace Science and Technology*, *Beijing Institute of Technology*, *Beijing* 610054, *China*;
2. *State Key Laboratory of Cryptology*, *Beijing* 100878, *China*;
3. *State Key Laboratory of Information Security* , *Institute of Information Engineering*, *Chinese Academy of Sciences*, *Beijing* 100093, *China*;
4. *Henan Key Laboratory of Network Cryptography Technology*, *Information Engineering University*, *Zhengzhou*, *Henan* 100878, *China*;
5. *Institute of Advanced Study*, *Tsinghua University*, *Beijing* 100084, *China*)

Abstract: Reverse-engineering based on fault analysis works by inducing abnormal ciphertexts by injecting faults into the equipment running a secret cipher, and then restoring its internal structure and parameters. This paper proposes a method of reverse-engineering the S-box table based on persistent fault, when the structure of round function except the S-box table is known. We take advantage of the fact that when S-box operations use the fault element, intermediate state errors appear, leading to ciphertext errors. Therefore, we construct special plaintexts and keys in order to induce errors in the S-box operation of the second round. Then, outputs of the S-box operation in the first round can be derived, i.e. one element of the S-box table is recovered. All elements of the S-box table can be recovered by using different plaintexts and keys. Taking AES-128 (Advanced Encryption Standard-128) algorithm as example, our method restores the complete S-box table by 1 441 792 encryptions. Compared with existing methods, our approach has obvious advantages in number of fault injections and complexity of computations. In addition, we applies this method to a SM4-like algorithm, and recovered its S-box table with an average of 1 900 544 encryptions. Finally, we discuss the universality of the new method, by considering two typical structures of block ciphers, Feistel and SPN (Substitution Permutation Network) structures respectively, and summarize conditions of our method.

收稿日期:2021-08-01;修回日期:2022-11-19;责任编辑:梅志强

Key words: reverse analysis; persistent fault; block cipher; substitution-box table

Foundation Item(s): National Key R&D Program of China (No.2021YFB3101500); National Natural Science Foundation of China (No.62002021); Henan Key Laboratory of Network Cryptography Technology (No.LNCT2020-A09); Beijing Institute of Technology Research Fund Program for Young Scholars; Cryptographic Application Industry Chain Supply and Demand Docking Platform of New Energy and Intelligent Connected Vehicle Industry (No.2021-0181-1-1)

1 引言

Kerckhoffs准则认为密码体制的安全性应仅依赖于密钥的安全性,而不应依赖于对密码算法的保密,因此现代密码分析技术大多以密码算法结构已知为前提. 但为了提升密码算法的保密属性,很多密码设备厂商使用了保密的加密算法. 这些算法多以密码算法标准为设计框架,定制保密S盒或改变运算参数. 逆向分析不同于常规的密码分析,是一种针对算法保密的密码进行的黑盒分析,旨在恢复保密算法的基本操作和参数,是评估保密算法安全性的一种重要手段. S盒作为分组密码算法中重要的非线性结构,其混淆能力直接决定了密码算法的安全性. 因此,针对分组密码算法S盒表的逆向分析具有重要的研究意义,有助于建立安全密码的通用设计规则.

目前,已存在的逆向分析方法主要有数学逆向分析[1~6]、芯片解剖逆向分析[7, 8]和基于侧信道分析的逆向分析[9~18]. 其中,数学逆向分析以逆向数学构造形式为主,数学分析复杂度高且通用性低;芯片解剖逆向需要通过还原电路进行逻辑判断、分析,所以存在周期长、代价高、可行性低等缺点;基于侧信道分析的逆向分析需要收集密码设备运行时的侧信息泄露,具有分析复杂度低、通用性高等特点.

侧信息被首次应用于算法的逆向分析是在2003年. Novak提出在算法多轮迭代中使用相同的S盒表并且多轮运算的中间结果能够进行交叉比较的情况下,可以考虑将侧信道分析方法用于算法逆向分析[9]. 作者通过分析算法运行时的能耗曲线,在无需识别具体值的情况下判断两个精确位置的中间值是否发生碰撞,对SIM卡A3/A8算法的S盒进行了逆向恢复. 2004年,Clavier针对Novak所提方法在实际应用中存在的不足进行了改进[10]. 在没有任何有关密钥的先验知识的前提下,逆向恢复出A3和A8算法的完整S盒表,并将密钥恢复出来. 基于此,Clavier正式提出了逆向工程侧信道分析(Side Channel Analysis for Reverse Engineering,SCARE)的概念. SCARE作为侧信道分析技术的一种新型应用,借助能量消耗等侧信息对加密实现的某些秘密部分进行逆向恢复,具备离散性和非侵入性等优点. 2005年,Daudigny等人将SCARE的概念应用于数据加密标准(Data Encryption Standard,DES)算法[11]. 在采集了目标智能卡运行DES算法时的能量消耗后,使用差分能量分析推断出所谓的调度信息,该信息揭示了特定比特位的移位变换时机,并据此推导出了用于加密轮函数和密钥扩展的置换表. 这类方法是在已知一些基本结构信息的情况下,对一个专有算法进行逆向恢复的. 此外,还有研究者针对某一类结构的算法进行逆向恢复的研究. 2008年,Réal等人使用选择明文的侧信道分析方法对硬件设计的未知Feistel方案进行逆向恢复[12]. 基于无论未知Feistel函数的输入是什么,函数的单轮输出都能被猜测出来的前提,作者提出了两种用于恢复算法的攻击方法,分别是对通用Feistel方案的插值攻击和对常用特定方案的改进攻击. 已有工作大多是在算法大部分结构和内容已知的假设下,对某一秘密组件进行恢复,但是这不适用于算法未知的情况. 为此,2013年Rivain等人提出了一种更适用于未知算法的逆向分析方法[13]. 利用针对S盒的碰撞攻击对分组密码算法的SPN(Substitution Permutation Network)结构进行了恢复,给出了函数等效表示. 同年,Clavier等人做了类似工作,利用选择明文碰撞攻击,对类AES(Advanced Encryption Standard)算法的密钥及各运算过程的秘密参数进行了逆向恢复,该方法对带有掩码防护的算法实现方案同样有效[14].

故障分析作为主动发起攻击并收集侧信息加以分析的一类方法,同样可以用于算法的逆向分析中. 2011年Pedro等人提出了故障注入逆向分析(Fault Injection for Reverse Engineering,FIRE)的概念,利用故障注入来恢复保密密码算法的S盒表[15]. 故障注入本来是用于向执行加密算法的芯片中注入故障,进而通过观察输出来恢复密钥的. Pedro等人的工作是通过收集和分析故障注入后芯片的输出,构造二进制变量方程式,进而利用高斯消除法求解S盒表元素. 唐明等人在2011年结合差分分析和故障分析方法成功恢复出Twofish算法的密钥,在2012年实现了基于差分故障分析恢复SPN结构和Feistel结构保密算法的S盒表[16]. 2013年Clavier等人基于无效故障模型逆向分析类AES算法,对零值故障注入前后加密同一明文时S盒输出结果进行比较[17]. 如果故障注入前S盒表的输出已经等于零,那么注入的故障不会产生影响;如果故障注入前S盒表的输出不为零,则注入的故障会导致S盒输出发生改变. 在前一种情况下由于两次密文结果均相等,会发生无效故障. 在该模型下,即使密钥未知,攻击者依然能够恢复出类AES算法中所

有未知参数. 2019年Caforio等人基于持续性故障模型[18]对减轮AES算法的S盒表进行了恢复[19]. 注入故障将S盒表位置a的元素置为0后,构造AES算法的密钥,并利用持续性故障分析(Persistent Fault Analysis,PFA)恢复出该加密情况下算法最后一轮的轮密钥. 通过对S盒表各种取值情况下的密钥扩展过程进行离线计算,找到可以由密钥成功生成最后一轮轮密钥的扩展过程,从而恢复出密钥扩展过程使用的S盒表. 但该方法的故障模型假设性较强,要求注入128次故障,并且要求故障注入位置可控,这在现实中是很难实现的. 此外,该方法只适用于5轮AES算法的S盒表恢复.

基于故障注入技术的逆向分析作为基于侧信道分析的逆向分析中的一类,需要向密码设备注入故障. 该技术对故障模型和注入精度有较高要求,因此基于故障注入技术逆向恢复分组算法S盒表是密码算法逆向分析领域极具研究前景的一项课题. 本文以仅S盒表未知的保密密码算法为研究对象,提出了一种基于持续性故障的分组密码算法S盒表逆向分析方法,并将其应用于多种分组密码算法中,通过实验验证了方法的有效性. 本文具体贡献如下:

(1)我们提出了一种基于持续性故障注入的分组密码算法S盒表逆向分析方法. 以S盒表保密的类AES-128算法为例,首先,通过控制明文使第二轮特定S盒输入遍历所有可能取值,确保能够取到故障元素并导致密文出错. 然后,固定明文中与特定S盒相关的字节并进行大量随机明文加密,通过分析密文出错比率确定诱导特定S盒故障的明文取值,在算法其他运算结构已知的前提下逆向推导出第一轮S盒运算输出,从而恢复出S盒表的一个元素. 最后,通过调整主密钥改变影响第二轮S盒输入的轮密钥取值,以相同的思路恢复出S盒表中其他元素. 实验结果显示我们的方法只需1次故障注入和1 441 792次加密就可恢复完整S盒表,相比于目前最优的Caforio等人的方法[19],计算复杂度降低了2^{10}次加密.

(2)我们将该方法应用于具有广义Feistel结构的类SM4算法,并结合Feistel和SPN这两种常用分组密码算法设计结构的特点,讨论了该方法的普适性. 实验结果显示,只需1次故障注入和1 900 544次加密就可恢复类SM4算法的完整S盒表. 由于本方法仅着眼于加密算法前两轮加密过程,对算法网络结构依赖性不强,故具有较强的普适性. 我们通过对各两种结构中保密S盒表恢复过程的分析,总结了新方法适用所需条件并给出举例说明.

2　背景知识

2.1　相关工作

Zhang等人于2018年首次提出持续性故障的概念及相应的密码分析技术[18],利用故障注入引起的密文统计偏差,通过大量加密并分析密文结果恢复密钥,此后,又于2020年给出在ATmega163L微控制器上的实际分析案例[20]. 该方法假设故障注入导致S盒表位置u的值由v变为v^*,在只注入单个元素故障的情况下,对AES-128算法的第10轮密钥K^{10}的16字节进行逐个恢复,实验结果显示该方法仅需约1 500条密文就可成功恢复出K^{10}. 作者以恢复最后一轮轮密钥的第i字节$k_i^{10}(i\in\{0,1,2,\cdots,15\})$为例阐述了三种密钥恢复的策略,分别描述如下:

策略1　已知v的情况下,通过对大量加密后的密文结果进行统计,得到密文的第i字节密文的分布情况,找出从未出现过的字节值,记作$t_{\min}$在密文数量足够多的情况下,k_i^{10}可以直接推导为:$k_i^{10}=v\oplus t_{\min}$.

策略2　假设v已知,由于S盒满足一一映射关系,在持续性故障存在的情况下,S盒运算将不会再得到结果值v. 于是$v\oplus k_i^{10}$就不可能出现在密文中,从故障设备中收集足够数量的密文,得到密文第i字节值,记为t. 得到t所有的取值情况后,可以由$k_i^{10}\neq v\oplus t$排除k_i^{10}的取值.

策略3　假设v^*已知,由于持续性故障的存在,出现了S盒表中多个位置的值均为v^*的情况. 于是$v^*\oplus k_i^{10}$会以较高概率出现在密文的第i字节中,约为其他取值出现概率的2倍. 通过对密文第i字节值的统计得到较高概率出现的取值,记为$t_{\max}$. 在密文数量足够多的情况下,k_i^{10}可以直接推导为:$k_i^{10}=v^*\oplus t_{\max}$.

本文方法的提出受到了策略1思想的启发. 虽同为基于持续性故障的方法,Zhang等人提出的PFA用于解决密钥恢复问题,而本文方法用于逆向恢复分组密码算法S盒表.

2.2　故障模型

本文主要研究如何恢复分组密码算法中的保密S盒表,提出了一种基于持续性故障的逆向恢复方法. 我们向运行分组密码算法的密码设备或芯片中注入持续性故障后,通过观察和分析设备运行结果,推测算法运算中间状态,进而恢复S盒表元素. 在介绍恢复方法之前,我们先给出以下前提假设:

假设1　持续性故障注入发生在分组算法密钥生成和加解密运算之前. 注入故障后,密钥扩展、加解密过程中凡是有S盒表参与的运算均会受到影响. 该假设弱于文献[19]的假设,后者假设持续性故障只影响加解密过程,不影响密钥生成过程,并不适用于密钥生成和加解密过程交错执行的实现方案.

假设2　攻击者无法控制S盒表中故障注入的位置u和故障结果值v^*. 该假设与文献[19]的假设相同,弱于文献[19]中对两值控制的假设.

假设3　攻击者有能力通过技术手段获得u、v^*的值以及v^*在S盒表中的位置u^*的值. 该假设与文献

[19]的假设相同.

假设 4 攻击者可以选择特定的明文和密钥进行加密. 该假设与文献[18]的假设相同,强于文献[18]的假设.

对于假设 2 中已知的三个变量,可通过如下方式获得:

(1)遍历一个 S 盒的所有输入,保持其他 S 盒的输入不变,对比注入故障前后的密文,出错密文对应的 S 盒输入值即为 u.

(2)遍历一个 S 盒的所有输入,观察注入故障后的密文,密文值相同的两个 S 盒的输入即为 u 和 u^*,已知 u 的情况下,可获得 u^* 值[21].

(3)注入故障前后,密文字节的值中出现概率显著高于其他值的即为 v^*,未出现字节即为 v[18].

2.3 符号说明

本文所用符号及其说明如表 1 所示.

表 1 符号说明

符号	说明
u	S 盒表故障注入的位置
S(·)	S 盒运算 $y_i=\mathrm{S}(i)$, $i\in\{0,1,2,\cdots,255\}$, $y_i\in\{0,1,2,\cdots,255\}$,是双射函数
S′(·)	注入故障后的 S 盒运算,除故障位置 u 外取值与 S(·)相同,非双射函数
$\mathrm{S}^{-1}(\cdot)$	S 盒逆运算 $i=\mathrm{S}^{-1}(y_i)$, $i\in\{0,1,2,\cdots,255\}$, $y_i\in\{0,1,2,\cdots,255\}$ 是双射函数
v	故障注入前 S 盒运算的取值,即 S(·)
v^*	故障注入后 S 盒运算的取值,即 S′(u)
u^*	故障注入前 v^* 对应 S 盒表的位置,即 $\mathrm{S}(u^*)=v^*$
$P=p_0\|\|p_1\|\|p_2\|\|\cdots\|\|p_{15}$	类 AES-128 算法明文, 其中 p_i 是明文第 i 字节, $i\in\{0,1,2,\cdots,255\}$
$K=k_0\|\|k_1\|\|k_2\|\|\cdots\|\|k_{15}$	类 AES-128 算法主密钥, 其中 k_i 是密钥的第 i 字节, $i\in\{0,1,2,\cdots,255\}$
$K^r=k_0^r\|\|k_1^r\|\|k_2^r\|\|\cdots\|\|k_{15}^r$	类 AES-128 算法参与第 r 轮轮密钥加运算的轮密钥, 其中 k_i^r 是第 r 轮轮密钥的第 i 字节, $i\in\{0,1,2,\cdots,15\}$, $r\in\{1,2,\cdots,10\}$
$C=c_0\|\|c_1\|\|c_2\|\|\cdots\|\|c_{15}$	类 AES-128 算法密文, 其中 c_i 是密文第 i 字节, $i\in\{0,1,2,\cdots,15\}$
$X=X_0\|\|X_1\|\|X_2\|\|X_3$	SM4 算法明文, 其中 X_i 是明文第 i 个 32 位字
$X_i=X_i^0\|\|X_i^1\|\|X_i^2\|\|X_i^3$	SM4 算法明文第 i 个字, 其中 X_i^j 是第 i 字明文第 j 个字节
$\mathrm{MK}=\mathrm{MK}_0\|\|\mathrm{MK}_1\|\|\mathrm{MK}_2\|\|\mathrm{MK}_3$	SM4 算法密钥, 其中 MK_i 是密钥的第 i 个字
$rk=rk^0\|\|rk^1\|\|rk^2\|\|\ldots\|\|rk^{31}$	SM4 算法轮密钥, 其中 rk^r 是第 r 轮轮密钥
$rk^r=rk_0^r\|\|rk_1^r\|\|rk_2^r\|\|rk_3^r$	SM4 算法第 r 轮轮密钥, 其中 rk_i^r 是第 r 轮密钥的第 i 字节
$Y=Y_0\|\|Y_1\|\|Y_2\|\|Y_3$	SM4 算法密文, 其中 Y_i 是密文第 i 个 32 位字

3 基于持续性故障的类 AES 算法 S 盒表逆向分析

3.1 S 盒表单元素逆向恢复

3.1.1 逆向恢复单元素流程

首先,构造明文和密钥使得第二轮特定 S 盒的输入遍历所有可能取值,确保能够取到故障值 u,将导致特定 S 盒取值相同的明文组成一个集合. 然后,从每个集合中随机选取若干明文,收集故障注入前后的加密结果. 通过比较密文进行错误率分析,找出全出错的集合,其对应的特定 S 盒输入必为 u. 最后,在算法其余结构已知的情况下,通过计算反向推导得到对应的第一轮 S 盒输出. 由于明文和密钥已知,可计算出第一轮 S 盒输入,这样就恢复出了 S 盒表的一个元素.

下面以类 AES-128 算法为例对本文的逆向恢复方法做详细说明,如图 1 所示为类 AES-128 算法的故障攻击诱导示意图.

首先,构造明文和密钥控制类 AES-128 算法第一轮 S 盒的输入. 对于密钥,根据标准 AES-128 算法密钥扩展过程,K^1 前 4 字节 $(k_0^1,k_1^1,k_2^1,k_3^1)$ 涉及 S 盒运算,为保证 K^1 计算过程的正确性,在已知 $\mathrm{S}(u^*)=v^*$ 的前提下,将 $k_i(i\in\{12,13,14,15\})$ 置为 u^*. 另外,在密钥扩展过程中,中间值的第 12 字节会与 0x01 做异或运算,为使 $(k_0^1,k_1^1,k_2^1,k_3^1)$ 四个字节取值相等,将 k_0 置为 0x01. 将主密钥其他字节设为 0,于是有密钥 $K=\begin{bmatrix}1&0&0&u^*\\0&0&0&u^*\\0&0&0&u^*\\0&0&0&u^*\end{bmatrix}$, 经密钥扩展得到 $K^1=\begin{bmatrix}v^*&v^*&v^*&u^*\oplus v^*\\v^*&v^*&v^*&u^*\oplus v^*\\v^*&v^*&v^*&u^*\oplus v^*\\v^*&v^*&v^*&u^*\oplus v^*\end{bmatrix}$. 对于明文,固定明文 $p_i(i\in\{0,5,10,15\})$ 的 4 个字节,使得明文与密钥的异或结果 $p_i\oplus k_i$ 在 $i\in\{0,5,10,15\}$ 时相等,记为 m. 经过第一轮加密 S 盒运算后会得到相同的输出 $\mathrm{S}(m)$,记为 n. 接着经过行移位操作,$p_i\oplus k_i$ $(i\in\{0,5,10,15\})$ 这 4 个值 n 会被移动到前 4 个字节的位置. 根据式(1)可知经过列混合操作后的前 4 个字节都为 n. 之后与由密钥通过密钥扩展得到的 K^1 进行

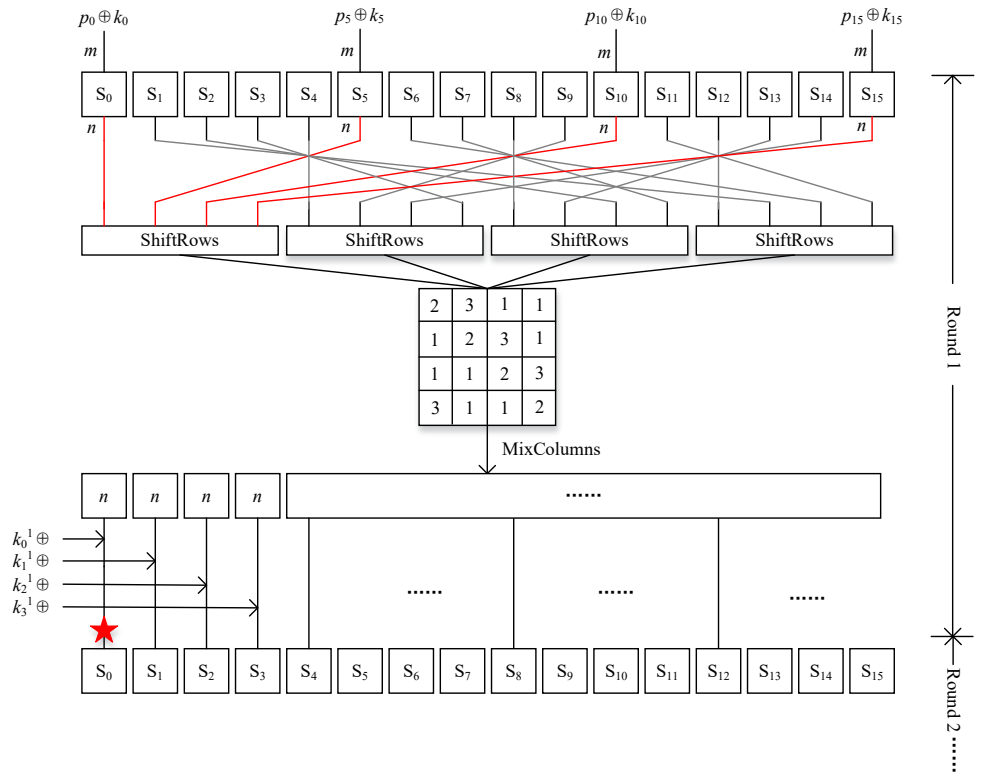


图1　类AES-128算法故障诱导示意图

轮密钥加操作,这4个n会作为中间结果的前4字节参与运算. 如果$n \oplus k_0^1$恰好等于u(即图1中五角星触发故障值),那么在第二轮加密中第0字节的S盒运算就会取到故障结果值v^*而不是v,从而影响后续几轮的加密过程导致密文恒出错.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} n \\ n \\ n \\ n \end{bmatrix} = \begin{bmatrix} 2 \cdot n \oplus 3 \cdot n \oplus 1 \cdot n \oplus 1 \cdot n \\ 1 \cdot n \oplus 2 \cdot n \oplus 3 \cdot n \oplus 1 \cdot n \\ 1 \cdot n \oplus 1 \cdot n \oplus 2 \cdot n \oplus 3 \cdot n \\ 3 \cdot n \oplus 1 \cdot n \oplus 1 \cdot n \oplus 2 \cdot n \end{bmatrix} = \begin{bmatrix} n \\ n \\ n \\ n \end{bmatrix} \tag{1}$$

然后,通过不断调整明文,遍历$p_j \oplus k_j (j \in \{0,5,10,15\})$所有的取值,找到加密过程恒出错的情况,也就是找到了算法第二轮S盒运算输入的第0字节等于u的情况,记录此时m的取值.

最后,再利用式(2)推导出n值,这样就恢复出一个元素,即$S(m)=n$.

$$n = u \oplus k_0^1 \tag{2}$$

在构造明文时我们令$p_j \oplus k_j$在$j \in \{0,5,10,15\}$上取值相同,原因在于在K^1前4字节取值相等的情况下,如果$p_j \oplus k_j (j \in \{0,5,10,15\})$的取值不一致,经过第一轮列混合运算后的前4字节取值也将不受控制,存在非第0字节取值等于$u \oplus k_0^1$,从而引发故障的可能性,此时将无法明确故障是否由第二轮S盒运算输入的第0字节取值引发. 此外,由于列混合运算是有S盒运算的多个输出字节值参与的,无法明确故障是由$\{p_j \oplus k_j | j \in \{0,5,10,15\}\}$中哪个元素值促成的,这会对元素索引值$m$的恢复造成干扰,导致S盒表单元素恢复失败.

算法1给出了恢复类AES-128算法S盒表单元素的伪代码. 第3~7行完成待加密的明文集合的构建;第8行是在注入故障前使用正确S盒对明文数据加密,得到密文集合c;在注入故障后,第10行使用错误S盒对明文集合再次进行加密,得到又一密文集c^*;第11~21行通过比较c和c^*元素,找出因注入故障而n_{enc}次加密全部出错的情况,即可恢复出一个元素. 这里n_{enc}表示的是能够判定加密过程恒出错所需故障注入前后针对一组数据进行的最少运算次数.

3.1.2　单元素逆向恢复实验

本文在基于Intel(R) Core(TM) i7-8550U处理器的PC上,利用Python语言实现了算法1,并在软件环境下通过修改标准AES-128算法S盒表中某个位置的值来模拟故障注入情况. 本节实验中,令u=0x6D,v^*=0x9D,u^*=0x75,有密钥$K=\begin{bmatrix} 1 & 0 & 0 & 0x75 \\ 0 & 0 & 0 & 0x75 \\ 0 & 0 & 0 & 0x75 \\ 0 & 0 & 0 & 0x75 \end{bmatrix}$和

$K^1=\begin{bmatrix}0x9D & 0x9D & 0x9D & 0x9D\oplus 0x75\\ 0x9D & 0x9D & 0x9D & 0x9D\oplus 0x75\\ 0x9D & 0x9D & 0x9D & 0x9D\oplus 0x75\\ 0x9D & 0x9D & 0x9D & 0x9D\oplus 0x75\end{bmatrix}$. 在明确$p_j\oplus k_j\,(j\in\{0,5,10,15\})$的取值后，随机选取明文$P$其他字节值进行大量加密，在$m$的256种取值情况下分别进行200次加密. 通过分析51 200个密文结果，可以发现当m等于某个值$(m\neq u)$时，密文结果恒出错. 图2展示的是$p_{15}\oplus k_{15}$不同取值下密文出错情况，图中每条线代表m的某个取值下，不同数量密文出错概率. m在大多数取值下，随着加密次数的增多，密文出错概率都趋于一定范围且小于1. 只有两种m取值下的密文出错概率恒为1(红线标注)，其中一个代表$m=u$作为输入值参与的第一轮S盒运算使结果恒出错的情况；另外一种m取值就对应于图1中故障诱导成功的情况.

基于这样的规律，分析密文结果并找到密文恒出错时对应的m值$(m\neq u)$. 已知K和K^1，根据式(2)可以得到n值，这样就可以恢复出S盒表中的一个元素. 根据图2可以发现少于5 000次加密就能将密文恒出错的情况分离出来. 不断调整实验次数后，加密18次(故障注入前后各加密9次，即算法1中$n_{enc}=9$)就可以判定加密过程是否恒出错. 恢复结果如图3所示，横坐标代表m取值的256种情况，纵坐标代表m取某一值时加密过程出错的概率. 当$m=0x17$和$m=0x6D$时密文恒出错. 显然，$m=0x6D$对应于m取到u的情况，而$m=0x17$对应于$n\oplus k_0^1=u$的情况. 于是有$m=0x17$，$n=u\oplus k_0^1=0x6D\oplus 0x9D=0xF0$，得到S(0x17)=0xF0. 因此，在$u=0x6D$，$v^*=0x9D$，$u^*=0x75$前提假设下，只需$18/2\times 256=2\,034$条明文样本就可以恢复出一个元素.

算法1 恢复类AES-128算法S盒表单元素

输入:故障注入位置u,故障注入后$S'(u)$取值v^*,故障注入前$S^{-1}(v^*)$取值u^*,每组数据在故障注入前后各加密次数n_{enc}
输出:S盒表的一个元素

```
K[]:={1,0,0,0,0,0,0,0,0,0,0,0,u*,u*,u*,u*}
随机生成明文数组p[256×n_enc][16]
FOR i=0 TO 255 DO
    FOR j=0 TO n_enc−1 DO
        p[i×n_enc+j][0]=0x01⊕i
        p[i×n_enc+j][5]=i
        p[i×n_enc+j][10]=i
        p[i×n_enc+j][15]=u*⊕i
    END FOR
END FOR
使用密钥K对明文数组中每一个明文进行加密,得到密文数组c
注入故障
利用密钥K对明文数组中每一个明文进行再次加密,得到数组c*
FOR i:=0 TO 255 DO
    errorCount:=0;//临时变量errorCount记录错误次数
    FOR j:=0 TO n_enc−1 DO
        IF c[i×n_enc+j]≠c*[i×n_enc+j] THEN
            errorCount:=errorCount+1
        END IF
    END FOR
    IF errorCount=n_enc AND
        p[i][0]⊕K[0]≠u THEN
        BREAK
    END IF
END FOR
RETURN S(p[i][0]⊕K[0]):=u⊕v*
```

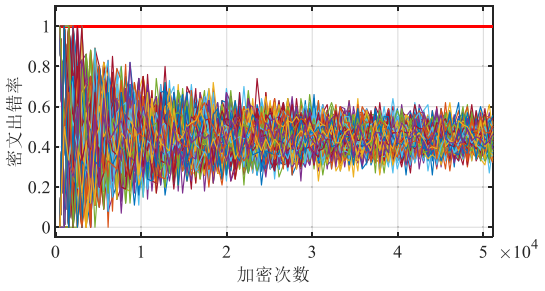


图2 m取值不同时类AES-128算法密文出错情况

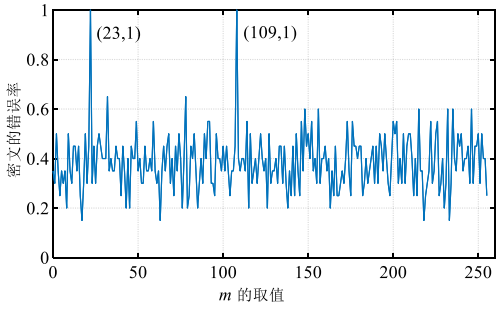


图3 $n_{enc}=9$,m取值不同时类AES-128算法密文错误率

对随机设置的u和v^*进行实验，利用不同数量的明文样本可以成功恢复一个元素的概率分布如图4所示. 在10 000次实验中，有3 178次实验仅需4 096个明文样本(对m的每种取值在故障注入前后各加密16次)即可恢复出一个元素. 成功恢复一个元素所需明文样本数平均为3 840个，即加密7 680次(对m的每种取值在故障注入前后各加密15次).

3.2 完整S盒表逆向恢复

3.2.1 逆向恢复S盒表流程

为了获得算法第一轮S盒运算输出的更多取值情况，我们考虑到第二轮S盒运算输入第0字节取值的影响因子除了有第一轮S盒运算输出，还应有其他已知且

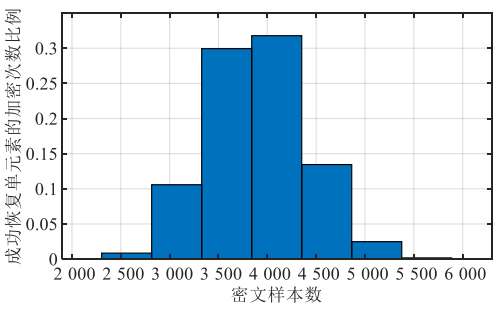


图4　恢复一个元素所需密文样本数的概率分布(类AES-128算法)

可控的因素. 于是,在单元素故障诱导成功的情况下,通过控制其他因素的变换,可以反推出不同的第一轮S盒输出.

结合分组密码算法每个轮函数都有轮密钥参与的运算特点,我们考虑改变影响第二轮S盒运算输入第0字节取值的轮密钥. 仍以类AES-128算法为例,第一轮S盒运算输出经过行移位、列混合运算后,会与第一轮轮密钥K^1进行轮密钥加运算,异或后的结果是第二轮S盒运算的输入. 这里,K^1已知且可控,于是我们通过构造不同的密钥得到不同的K^1,在u一定的条件下,根据式(2)可计算出不同的n值,这样就恢复出不同的元素. 遍历影响k_0^1的所有取值情况,即可恢复完整的S盒表.

为获得密钥更多的取值情况,调整K的构造形式为$K=\begin{bmatrix}1\oplus\Delta & 0 & 0 & u^*\\ 0\oplus\Delta & 0 & 0 & u^*\\ \Delta & & 0 & u^*\\ & & 0 & u^*\end{bmatrix}$,其中$\Delta\in[0\text{x}00,0\text{x}FF]$. 遍历$\Delta$的256种取值,$k_0^1$就有256种取值情况,就有256种$n$的取值与之对应. 分别找到对应情况下的$m$值,就可以恢复出更多元素. 然而,实际情况中存在某些Δ取值下无法成功恢复的情况,原因在于这些Δ对应的K在密钥扩展中使用了u. 根据故障模型中"注入故障后,密钥扩展、加解密过程中凡是有S盒参与的运算操作均会受影响"这一点,分组密码算法的密钥扩展中如果存在S盒运算操作,并且恰好取到了故障值,就会导致某一轮K^r错误,造成该密钥参与下的后续加密过程出错,无法分离出密文恒出错的情况,从而恢复失败.

针对这种情况,本文考虑变换$k_i(i\in\{12,13,14,15\})$构造新的密钥进行再一次的实验,产生新的元素结果集,对照已有实验结果进行查漏补缺,直至恢复全部元素. 虽然本文仅已知$\text{S}(u^*)=v^*$,但我们可以从已有的实验结果集中任取一元素α,令$a=\alpha,b=\text{S}(\alpha)$. 调整密钥$K=\begin{bmatrix}1\oplus\Delta & 0 & 0 & a\\ 0\oplus\Delta & 0 & 0 & a\\ 0\oplus\Delta & 0 & 0 & a\\ 0\oplus\Delta & 0 & 0 & a\end{bmatrix}$,经密钥扩展得到$K^1=\begin{bmatrix}b\oplus\Delta & b\oplus\Delta & b\oplus\Delta & a\oplus b\oplus\Delta\\ b\oplus\Delta & b\oplus\Delta & b\oplus\Delta & a\oplus b\oplus\Delta\\ b\oplus\Delta & b\oplus\Delta & b\oplus\Delta & a\oplus b\oplus\Delta\\ b\oplus\Delta & b\oplus\Delta & b\oplus\Delta & a\oplus b\oplus\Delta\end{bmatrix}$. 再次遍历$\Delta$的256种情况,获得又一个元素结果集. 对该结果集与已有结果集进行取并集运算,就可以恢复出更多元素. 多次调整密钥的取值进行实验,直到恢复出完整的S盒表. 但是该方法仅仅忽略了因密钥扩展过程出错导致对应的S盒表元素恢复失败的情况,理论上虽能够恢复完整S盒表,但可能存在计算资源浪费等问题. 为此,本文考虑从密钥构造入手彻底解决此类问题.

按照现有密钥的构造形式,确定u^*和Δ后,$k_i^r(i\in\{12,13,14,15\},r\in\{0,1,2,\cdots,10\})$都将有明确的值. 只要一组加密存在$k_i^r=u$的情况,就无法恢复出相应的元素. 为此,考虑将$k_{13}$置为$u^*$,并随机选取$k_i(i\in\{12,14,15\})$,随机化$k_i^r$取值,降低$k_i^r=u$致使密钥扩展过程出错的概率,提升S盒表的逆向恢复效率. 于是我们改进K的构造形式为$K=\begin{bmatrix}1\oplus\Delta & 0 & 0 & \text{ran}_1\\ 0 & 0 & 0 & u^*\\ 0 & 0 & 0 & \text{ran}_2\\ 0 & 0 & 0 & \text{ran}_3\end{bmatrix}$,经密钥扩展后有$K^1=\begin{bmatrix}v^*\oplus\Delta & v^*\oplus\Delta & v^*\oplus\Delta & \text{ran}_1\oplus v^*\oplus\Delta\\ \text{S}(\text{ran}_2) & \text{S}(\text{ran}_2) & \text{S}(\text{ran}_2) & u^*\oplus\text{S}(\text{ran}_2)\\ \text{S}(\text{ran}_3) & \text{S}(\text{ran}_3) & \text{S}(\text{ran}_3) & \text{ran}_2\oplus\text{S}(\text{ran}_3)\\ \text{S}(\text{ran}_1) & \text{S}(\text{ran}_1) & \text{S}(\text{ran}_1) & \text{ran}_3\oplus\text{S}(\text{ran}_1)\end{bmatrix}$,其中$\text{ran}_1$、$\text{ran}_2$和$\text{ran}_3$是[0x00, 0xFF]范围内的随机数. 基于新的密钥构造方式,在一组加密情况下进行大量运算的过程中,只有k_0^1的取值是固定的,$k_i^1(i\in\{1,2,3\})$的取值在每一次运算中都是随机的,这样保证了在一组加密情况下成功诱导故障的只能是第二轮S盒第0字节的运算. 此时就不必要求第一轮轮函数中列混合运算结果的前4字节保持一致,在构造明文时也就不必要求$p_i\oplus k_i(i\in\{0,5,10,15\})$取值相同. 不过,为避免出现$p_i\oplus k_i(i\in\{0,5,10\})$对逆向恢复的干扰,确保$p_i\oplus k_i(i\in\{0,5,10\})$作为输入值参与S盒运算的正确性,令$p_i\oplus k_i(i\in\{0,5,10\})$. 同时,记$p_{15}\oplus k_{15}$取值为$m$,经过第一轮S盒运算后的输出$\text{S}(m)$,记为$n$. 经过标准AES-128算法的行移位操作,$p_i\oplus k_i(i\in\{0,5,10,15\})$会被移动到前四个字节的位置. 根据式(1),经过列混合操作后的第一个字节值为n.

不断调整明文,遍历$p_{15}\oplus k_{15}$的256种取值情况,找到加密过程恒出错的情况,利用式(2)恢复出对应元素. 遍历Δ的256种取值后,可以恢复出256个元素. 类AES-128算法S盒表全部元素的恢复算法描述见算法2. 第3~5行构建了256个密钥,形成密钥集合K;第

7~13行针对256个密钥分别构建了$256\times n_{enc}$个明文，形成$256\times256\times n_{enc}$大小的明文集合；第14行是在注入故障前对明文数据加密，得到密文集合c；在注入故障后，第16行对明文集合再次进行加密，得到又一密文集c^*；第17~30行通过比较c和c^*元素，找出每一种密钥的情况下，因注入故障而n_{enc}次加密全部出错的情况，进而恢复出256个元素，即完整S盒表．这里n_{enc}表示的是能够判定加密过程恒出错所需故障注入前后针对一组数据进行的最少加密次数．

3.2.2 S盒表逆向恢复实验

本节同样在基于Intel(R) Core(TM) i7-8550U处理器的64位PC上，利用Python语言进行验证．保持u=0x6D，v^*=0x9D，u^*=0x75不变，首先按照改进前的密钥构造方法进行实验．Δ遍历256种取值情况，理论上就可以恢复出完整的S盒表，而实际上并不是所有的Δ取值情况下，加密n_{enc}次都可以分离出密文恒出错对应的m值($m\neq u$)．如图5，左侧柱形(蓝色标注)代表Δ=0x00时恢复单元素的情况，加密4 608次就可以分离出密文错误率恒为1的情况；右侧柱形(黄色标注)代表256种取值($\Delta\in$[0x00, 0xFF])恢复单元素的平均情况，平均加密约6 656次有89%的可能性恢复出一个元素．由此看出，在Δ的某种取值下存在11%的可能性无法恢复出一个元素．理论上Δ遍历256种取值后可以恢复出256个元素，但实际上仅能恢复228个元素，这恰好验证了图5的结果．实验值与理论值有偏差的原因就在于Δ某些取值下的密钥在密钥扩展过程中使用了u，导致某一轮K^r错误，造成该密钥参与下的后续加密过程出错，无法分离出密文恒出错的情况，从而恢复失败．

在已知S盒表任一元素的情况下，Δ遍历256种取值后平均可恢复约227个元素，剩余未知的29个S盒元素可以使用暴力搜索法恢复，穷搜复杂度为29!(10^{30}的数量级)．至此，类AES-128算法S盒表逆向恢复的复杂度将由穷举全表的256！大大降低至29！．

利用$S(u^*)=v^*$完成实验后，从恢复的元素集合中任取一个元素y_i，令$a=i$进行再一次实验，Δ遍历256种情况后又可以恢复出约89%的S盒表元素．像这样从已有结果集中多次随机选取元素进行实验，并对各结果集进行取并运算，就可以恢复出更多S盒元素，直到恢复出完整的S盒表．根据图6的四幅子图，横坐标表示已知条件中u^*的取值情况，纵坐标表示以已知u^*为起点的实验中使用不同数量的y_i恢复S盒表元素的数量．可以发现，使用一个y_i(此时的(i, y_i)就是前提条件中的(u^*, v^*))平均能恢复出89%的S盒表元素，约226.8个．从已有结果集中再选取一个y_i(此时的(i,y_i)作为已知条件中的(a,b))，恢复结果集与已有结果集进行取并运算后会得到平均约251.9个元素；从已有结果集中再选取两个y_i可恢复平均约254.7个元素；从已有

算法2 恢复类AES-128算法完整S盒表

输入:故障注入位置u,故障注入后$S'(u)$取值v^*,故障注入前$S^{-1}(v^*)$取值u^*,故障注入前后的加密次数n_{enc}
输出:S盒表$S(i)=\{y_i|i\in[0,255]\}$

```
生成种子K[]:={1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,u*,0,0};
生成密钥数组K_set[][],其中K_set[i][]:=K[],i∈[0,255];
FOR i:=0 TO 255 DO
    K_set[i][]:=K_set[i][0]⊕i;
    K_set[i][0]:=ran_1;
    K_set[i][11]:=ran_2;
    K_set[i][15]:=ran_3;
END FOR
随机生成明文数组p[256×256×n_enc][16];
FOR t:=0 TO 255 DO
    FOR i:=0 TO 255 DO
        FOR j:=0 TO n_enc-1 DO
            index:=t×256×n_enc+i×n_enc+j//临时变量index为明文索引号
            p[index][0]:=0x01⊕u*⊕t;
            p[index][5]:=0x00⊕u*;
            p[index][10]:=0x00⊕u*;
            p[index][15]:=ran_3⊕i;
        END FOR
    END FOR
END FOR
使用密钥K对明文数组进行加密,得到密文数组c;
注入故障;
用密钥K对明文数组进行再次加密,得到密文数组c*;
FOR t:=0 TO 255 DO
    FOR i:=0 TO 255 DO
        errorCount:=0;//临时变量errorCount记录错误次数
        FOR j:=0 TO n_enc-1 DO
            index:=t×256×n_enc+i×n_enc+j;
            IF c[index]≠c*[index] THEN
                errorCount:=errorCount+1;
            END IF
        END FOR
        IF errorCount=n_enc AND
         p[t×256×n_enc+i×n_enc+j][0]⊕K_set[t][15]≠u
         THEN
            S(p[i][15]⊕K_set[t][15])=u⊕v*⊕t;
            BREAK;
        END IF
    END FOR
END FOR
RETURN S[i].
```

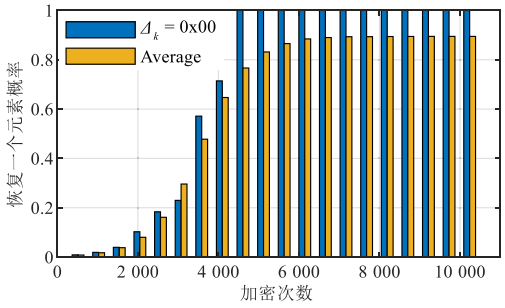
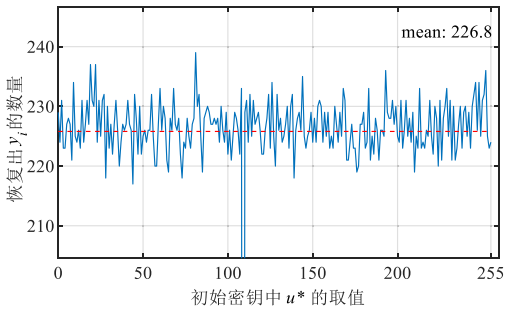


图5　Δ不同取值下单元素恢复成功率随加密次数变化情况(类AES-128算法)

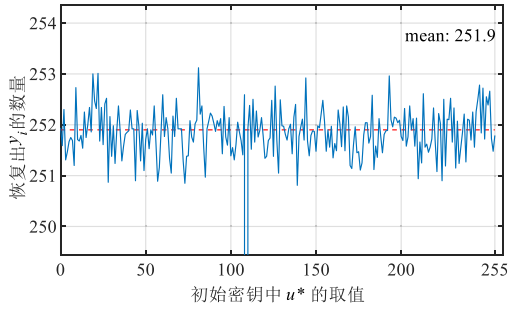
结果集中再选取三个y_i可恢复完整S盒表. 在图6的4幅子图中,密钥中$u^*=$0x6D时,纵坐标的值都为0,这种情况可以忽略,因为$u^*=u$并不符合故障模型的假设. 该方法虽避开了密钥扩展出错导致单元素恢复失败的问题,但是从图6(b)~图6(d)中可以看出,使用了2个y_i以后,再次选取元素进行实验的效果并不显著,恢复数量没有明显增多. 在这个过程中会有多次针对已恢复的S盒表元素的重复计算,一定程度上增加了时间和计算成本.

按照改进后的密钥构造方式进行实验. 将k_{13}置为u^*, 并随机选取$k_i(i\in\{12,14,15\})$, 使得$k_i^r(i\in\{12,13,14,15\},r\in\{0,1,2,\cdots,10\})$的取值随机化. 这样密钥扩展中有随机数参与,可有效解决密钥生成过程恒出错的问题,理论上每种Δ取值下都能恢复出一个元素. 保持u,v^*,u^*取值不变,遍历Δ的256种情况,能够得到除S(u)以外的255个S盒元素. 由于注入的故障,S(u)不会参与任何加密运算过程,所以无法对其进行推导,但在255个元素已知的情况下,S(u)取值也是显而易见的. 此外,在这种密钥构造方法下,K^1前4字节$(k_0^1,k_1^1,k_2^1,k_3^1)$以极低的概率有相等的取值,针对明文的构造限制因此可以放宽. 本文方法的思路是诱导算法第二轮S盒运算输入的第0字节的取u,所以只需令$p_i\oplus k_i(\ i\in\{0,5,10\})$相等,遍历$p_{15}\oplus k_{15}$所有的取值情况即可.

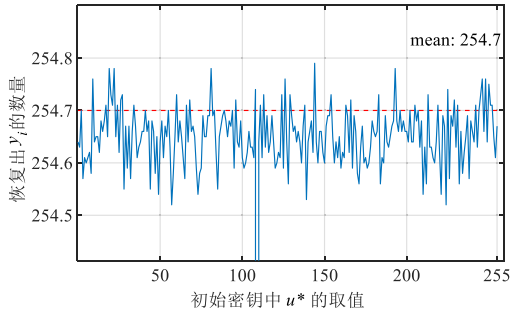
在仅一个元素已知的前提下,密钥构造方法改进前后的方法效果对比如图7. 横坐标代表已知条件中u^*的不同取值,纵坐标代表仅已知某u^*的条件下能够恢复出的S盒表元素个数. 图中的蓝色曲线和红色直线分别代表密钥构造方法改进前后的情况,其中红线部分除了在$u^*=$0x6D时纵坐标取值为0,其余情况纵坐标取值均为255. 从中可以看出:方法改进后,已知一个元素,即仅已知S(u^*)=v^*进行实验就可以恢复S盒表全部元素. 于是,在恢复完整S盒表的加密次数上,相比于方法改进前的7 864 320次,改进后的方法平均仅需要



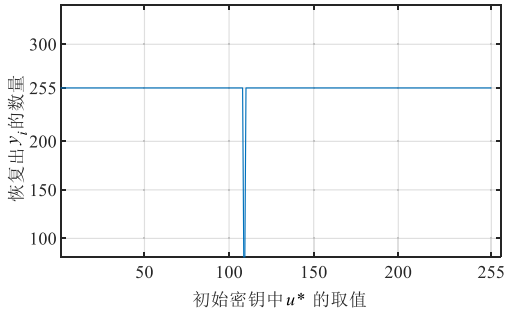
(a) 已知1个y_i



(b) 已知2个y_i



(c) 已知3个y_i



(d) 已知4个y_i

图6　使用不同数量y_i恢复S盒表情况对比

1 441 792次加密,约为$2^{20.46}$.

3.3　与同类方法的效率对比

与本文方法一样同为基于故障注入逆向恢复类

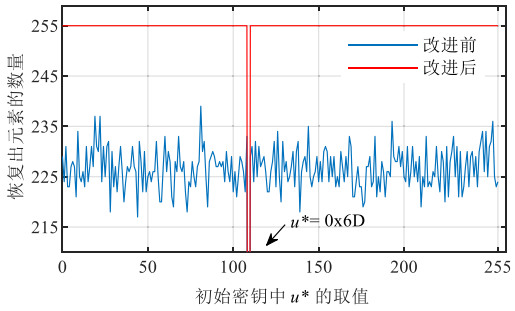


图7 改进密钥构造方案前后S盒表恢复情况对比

AES-128算法S盒表的方法还有很多. 在IFIP 2011中, Pedro等人在类AES算法第9轮列混合运算之前注入单字节故障,通过建立故障注入前后第9轮列混合运算结果的差分值同密文差分值之间的等价关系,反向推导出完整S盒表[15]. 在FDTC 2013中,Clavier等人基于无效故障分析,通过注入零值故障对类AES算法结构进行逆向分析,在恢复出ShiftRows,MixColumns,RotWord等运算参数的基础上恢复出完整S盒表[17]. 在SPACE 2019中,Caforio等人使用持续性故障分析PFA恢复出减轮AES算法中最后的轮密钥,对特定位置注入故障后构造密钥进行密钥扩展运算,直到扩展出最后的轮密钥,说明扩展过程使用的S盒表都是正确的,以此恢复5轮类AES算法的S盒表[18].

本文将从能否构建特殊密钥、前提需已知类AES算法S盒表元素的数量、方法最多可以恢复多少轮S盒表元素、恢复过程注入故障的次数、逆向恢复S盒表的运算复杂度这五个方面,将本文方法与上述方法进行对比,结果如表2.

表2 类AES算法S盒表逆向恢复方法对比

	IFIP 2011[15]	FDTC 2013[17]	SPACE 2019[18]	本文
密钥可选	否	否	是	是
S盒表元素已知数量	0	0	0	1
轮数限制	10	10	5	10
故障注入次数	400	25 514	128	1
运算复杂度	2^{57}	无描述	2^{31}	$1\,441\,792\approx 2^{20.46}$

4 基于持续性故障的S盒表逆向分析方法普适性研究

4.1 基于持续性故障逆向恢复类SM4算法S盒表

4.1.1 逆向恢复单元素流程

通过构造明文和密钥,我们同样能够控制类SM4算法第一轮F函数中S盒运算的输入. 在进行大量加密的过程中寻找第二轮F函数中S盒运算输入的第0字节成功诱导故障的情况,从而逆向恢复单元素. 类SM4算法每一轮函数中的线性变换L以一个4字节的字为运算单位. 这意味着,要想逆向恢复线性变换L之前的S盒运算输出的任一字节值,需已知线性变换L的4字节全部输出. L变换具有的一个重要特点是输入的4字节取值相同,那么对应输出的4字节取值不仅相同,还会是输入值的变形,可以通过计算得到. 如果4字节输入为(n,n,n,n),相应4字节输出就为$(n\ll 2, n\ll 2, n\ll 2, n\ll 2)$.

针对类SM4算法的故障诱导指示如图8. 利用线性变换L的运算特点,可以令第一轮F函数中S盒输入值$X_1^i\oplus X_2^i\oplus X_3^i\oplus rk_i^0\ (i\in\{0,1,2,3\})$这四个字节取值相同,即图中的$m$. 经过第一轮S盒运算后得到输出值S($m$),记为$n$. 接着$\tau$变换的4字节输出$(n,n,n,n)$将作为一个32位字参与线性变换$L$. 完成第一轮$F$函数后得到的$X_4=(X_0^0\oplus(n\ll 2), X_0^1\oplus(n\ll 2), X_0^2\oplus(n\ll 2), X_0^3\oplus(n\ll 2))$会作为输入参与第二轮$F$函数. 如果第二轮S盒运算输入的第0字节$X_2^0\oplus X_3^0\oplus X_4^0\oplus rk_0^1$恰好等于$u$(即图8五角星触发故障值),那么相应S盒输出结果就是故障值v^*,将直接影响第二轮F函数的运算结果,从而影响剩余30轮F函数的加密过程,造成密文恒出错. 通过对特定的明文和密钥进行大量加密,遍历m的全部取值情况,将具有相同m取值的运算看作一组加密过程,通过分析密文找出加密过程恒出错对应的一组加密情况. 计算$m=X_1^0\oplus X_2^0\oplus X_3^0\oplus rk_0^0$, $n=(u\oplus X_2^0\oplus X_3^0\oplus rk_0^1\oplus X_0^0)\ggg 2$,这样就能恢复出S盒表的一个元素.

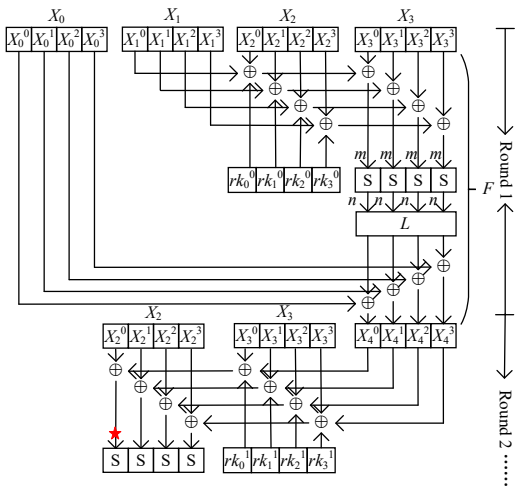


图8 类SM4算法故障诱导指示图

对于密钥的构造,根据标准SM4算法流程,密钥MK与系统参数FK将通过异或运算产生(K_0,K_1,K_2,K_3). 根据标准SM4算法密钥扩展过程,T'变换与加密算法轮函数中的T基本相同,其中的线性变换L'同样是以32位字为运算单位. 要想明确rk_0^1的取值,意味着

$K_2\oplus K_3\oplus K_4\oplus \mathrm{CK}_1$中各字节作为输入所对应的S盒输出应已知. 同理,明确K_4,即rk^0的取值,需建立在$K_1\oplus K_2\oplus K_3\oplus \mathrm{CK}_0$各字节作为输入对应的S盒输出已知的基础上. 在前提仅已知$\mathrm{S}(u^*)=v^*$的情况下有式(3),经恒等变换后有式(4).

$$\begin{cases} K_1^j\oplus K_2^j\oplus K_3^j\oplus \mathrm{CK}_0^j=u^* \\ K_2^j\oplus K_3^j\oplus K_4^j\oplus \mathrm{CK}_1^j=u^* \end{cases}\left(j\in\{0,1,2,3\}\right) \tag{3}$$

$$\begin{cases} K_2^j\oplus K_3^j=u^*\oplus \mathrm{CK}_0^j\oplus K_1^j \\ K_2^j\oplus K_3^j=u^*\oplus \mathrm{CK}_1^j\oplus K_4^j \end{cases}\left(j\in\{0,1,2,3\}\right) \tag{4}$$

基于此,令$K_1^j=0\mathrm{x}00(j\in\{0,1,2,3\})$,$K_2^j=u^*(j\in\{0,1,2,3\})$,$K_3^j=\mathrm{CK}_0^j(j\in\{0,1,2,3\})$,有$K_4^j=\mathrm{CK}_0^j\oplus\mathrm{CK}_1^j(j\in\{0,1,2,3\})$. 标准SM4算法密钥扩展过程$T'$变换的线性变换$L'$与加密轮函数中$T$变换的线性变换$L$有类似的特点. 如果$L'$输入的4字节取值相同,那么其输出的4字节取值也将相同,并且能够通过计算得到. 构造K_1,K_2,K_3的取值后,根据SM4算法密钥生成

$$\mathrm{MK}=\begin{bmatrix} 0\mathrm{xBF}\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16^3)) & 0\mathrm{x}56 & u^*\oplus 0\mathrm{x}67\oplus\mathrm{ran}_0 & 0\mathrm{xB2}\oplus\mathrm{ran}_0 \\ 0\mathrm{x}95\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16^2)) & 0\mathrm{xAA} & u^*\oplus 0\mathrm{x}7\mathrm{D}\oplus\mathrm{ran}_1 & 0\mathrm{x}77\oplus\mathrm{ran}_1 \\ 0\mathrm{x}9\mathrm{E}\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16)) & 0\mathrm{x}33 & u^*\oplus 0\mathrm{x}91\oplus\mathrm{ran}_2 & 0\mathrm{x}2\mathrm{C}\oplus\mathrm{ran}_2 \\ 0\mathrm{xE}2\oplus(L'(v^*,v^*,v^*,v^*)\&(15)) & 0\mathrm{x}50 & u^*\oplus 0\mathrm{x}97\oplus\mathrm{ran}_3 & 0\mathrm{xC}9\oplus\mathrm{ran}_3 \end{bmatrix} \tag{5}$$

为保证第一轮F函数中4个S盒输入取值相同,同时减少影响第二轮S盒运算输入的变量个数,令明文$X_0=0\mathrm{x}00000000$,$X_2^i=X_3^i(i\in\{0,1,2,3\})$,$X_1^i\oplus rk_i^0(i\in\{0,1,2,3\})$4个字节相等. 通过调整明文,遍历$X_1^i\oplus rk_i^0(i\in\{0,1,2,3\})$所有的取值情况,保证$X_2$和$X_3$相等的前提下对随机选取两者完成大量加密,找到加密过程恒出错的情况,即故障诱导成功. 此时$m=X_1^i\oplus rk_i^0(i\in\{0,1,2,3\})$,$n=(u\oplus rk_0^1)\ggg 2$,这样就恢复出一个元素.

4.1.2　逆向恢复S盒表流程

在类SM4算法第二轮F函数S盒运算输入的第0字节为u的条件下,rk_0^1的取值不同,意味着第一轮F函数中S盒运算输出值不同,相应就有不同的S盒输入与之对应,从而恢复出更多的S盒表元素. 变换$rk_0^1=\Delta\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16^3))=\Delta\oplus(L'(v^*,v^*,v^*,v^*)\ggg 24)$,$\Delta\in[0\mathrm{x}00,0\mathrm{xFF}]$. 根据密钥生成$rk^1=K_1\oplus T'(K_2\oplus K_3\oplus K_4\oplus\mathrm{CK}_1)$,可以令$K_1^j=\Delta(j\in\{0,1,2,3\})$. 此外,为满足式(3)的等价关系,令$K_2^j=\Delta\oplus u^*\oplus\mathrm{ran}_j(j\in\{0,1,2,3\})$,$K_0^j=\Delta\oplus\mathrm{CK}_0^j\oplus\mathrm{CK}_1^j\oplus(T'(u^*,u^*,u^*,u^*)\&(15\times16^{3-j}))=\Delta\oplus\mathrm{CK}_1^j\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16^{3-j}))$,同时保持$K_3$不变. 调整MK的构造形式如式(6). 遍历$\Delta\in[0\mathrm{x}00,0\mathrm{xFF}]$,$rk_0^1$有256种取值情况,对应256种$n$的取值,分别找到对应的$m$值,可以恢复出更多元素.

$$\mathrm{MK}=\begin{bmatrix} \Delta\oplus0\mathrm{xBF}\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16^3)) & \Delta\oplus0\mathrm{x}56 & \Delta\oplus u^*\oplus0\mathrm{x}67\oplus\mathrm{ran}_0 & \Delta\oplus0\mathrm{xB}2\oplus\mathrm{ran}_0 \\ \Delta\oplus0\mathrm{x}95\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16^2)) & \Delta\oplus0\mathrm{xAA} & \Delta\oplus u^*\oplus0\mathrm{x}7\mathrm{D}\oplus\mathrm{ran}_1 & \Delta\oplus0\mathrm{x}77\oplus\mathrm{ran}_1 \\ \Delta\oplus0\mathrm{x}9\mathrm{E}\oplus(L'(v^*,v^*,v^*,v^*)\&(15\times16)) & \Delta\oplus0\mathrm{x}33 & \Delta\oplus u^*\oplus0\mathrm{x}91\oplus\mathrm{ran}_2 & \Delta\oplus0\mathrm{x}2\mathrm{C}\oplus\mathrm{ran}_2 \\ \Delta\oplus0\mathrm{xE}2\oplus(L'(v^*,v^*,v^*,v^*)\&(15)) & \Delta\oplus0\mathrm{x}50 & \Delta\oplus u^*\oplus0\mathrm{x}97\oplus\mathrm{ran}_3 & \Delta\oplus0\mathrm{xC}9\oplus\mathrm{ran}_3 \end{bmatrix} \tag{6}$$

4.1.3　逆向恢复实验

本节在基于Intel(R) Core(TM) i7-8550U处理器的64位PC上,利用Python语言进行了仿真实验,并在软件环境下通过修改标准SM4算法S盒表中某个位置的值来模拟故障注入情况. 实验中令$u=0\mathrm{x}57$,$v^*=0\mathrm{x}33$,$u^*=0\mathrm{x}28$,有$\mathrm{MK}=\begin{bmatrix} 0\mathrm{x}73 & 0\mathrm{x}56 & 0x4\mathrm{F}\oplus\mathrm{ran}_0 & 0\mathrm{xB}2\oplus\mathrm{ran}_0 \\ 0\mathrm{x}59 & 0\mathrm{xAA} & 0x55\oplus\mathrm{ran}_1 & 0\mathrm{x}77\oplus\mathrm{ran}_1 \\ 0\mathrm{x}52 & 0\mathrm{x}33 & 0x\mathrm{B}9\oplus\mathrm{ran}_2 & 0\mathrm{x}2\mathrm{C}\oplus\mathrm{ran}_2 \\ 0\mathrm{x}2\mathrm{E} & 0\mathrm{x}50 & 0x\mathrm{BF}\oplus\mathrm{ran}_3 & 0\mathrm{xC}9\oplus\mathrm{ran}_3 \end{bmatrix}$,$rk^0=[0\mathrm{x}1\mathrm{C}\ \ 0\mathrm{x}24\ \ 0\mathrm{x}24\ \ 0\mathrm{x}24]$,$rk^1=[0\mathrm{xCC}\ \ 0\mathrm{xCC}\ \ 0\mathrm{xCC}\ \ 0\mathrm{xCC}]$.

恢复结果如图9所示,横坐标代表m取值的256种情况,纵坐标代表m取某一定值时加密过程出错的概率. 当$m=76(0\mathrm{x}4\mathrm{C})$和$m=87(0\mathrm{x}57)$时密文全部出错. 显然,$m=87(0\mathrm{x}57)$是由于$m$取到$u$导致加密过程出错,而$m=76(0\mathrm{x}4\mathrm{C})$对应于$n=(u\oplus rk_0^1)\ggg 2$的情况. 于是有$m=0\mathrm{x}4\mathrm{C}$,$n=(u\oplus rk_0^1)\ggg 2=(0\mathrm{x}57\oplus0\mathrm{xCC})\ggg 2$,得到$\mathrm{S}(i)=y_i$,$i=0\mathrm{x}4\mathrm{C}$,$y_i=0\mathrm{xE}6$.

在u,v^*,u^*值保持不变的条件下,6 144次加密(对

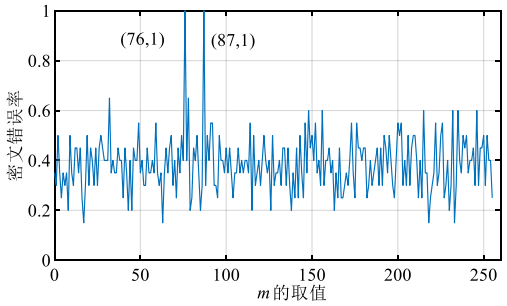


图9 $n_{enc}=12$时,m不同取值下密文错误率(类SM4算法)

m的每种取值在故障注入前后各加密12次)就可以恢复出一个元素. 对随机设置的u和v^*进行实验,不同加密次数成功恢复一个元素的概率分布如图10所示. 在10 000次实验中,有1 797次实验需要7 168次加密(对m的每种取值在故障注入前后各加密14次)可恢复出一个元素. 成功恢复一个元素所需加密次数平均为3 712个,即加密7 424次(对m的每种取值在故障注入前后各加密14.5次).

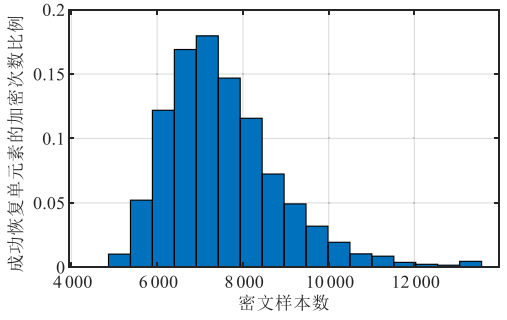


图10 恢复一个元素所需加密次数概率分布(类SM4算法)

之后,改变Δ的取值,以Δ=0x45为例进行实验的逆向恢复结果如图11所示,横坐标代表m取值的256种情况,纵坐标代表m取某一定值时加密过程出错的概率. 当$m=7$(0x07)和$m=87$(0x57)时密文全部出错. 显然$m=87$(0x57)时由于m取到u导致加密过程出错,而$m=7$(0x07)对应于$n=(u\oplus rk_0^1)\gg 2$的情况. 于是有$m=0$x07,$n=(u\oplus rk_0^1)\gg 2=(0$x57$\oplus 0$x89$)\gg 2=0$xB7,得到S($i$)=$y_i$,$i=0$x07,$y_i=0$xB7. 类似地,选定$\Delta$为某一个值时可以恢复一个元素,$\Delta$遍历256种取值情况,就可以恢复出完整的S盒表.

随机变换MK_2和MK_3的取值,并随机选取Δ进行10 000次实验,不同加密次数下成功恢复一个元素的概率分布如图12所示. 在10 000次实验中都没有出现元素恢复失败的情况,其中有1 788次实验使用7 168次加密(故障注入前后各加密14次)成功恢复出了一个元素. 实验显示恢复一个元素,平均需7 424次加密,因此恢复完整S盒表,平均约需加密1 900 544次.

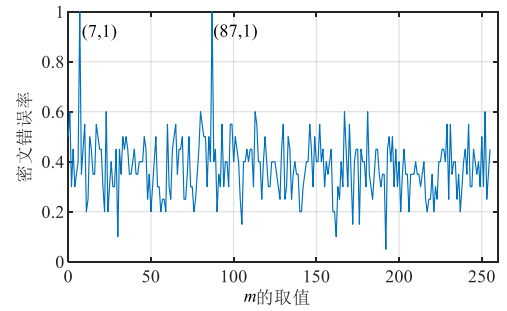


图11 $\Delta=$0x45时256种m取值对应的出错情况(类SM4算法)

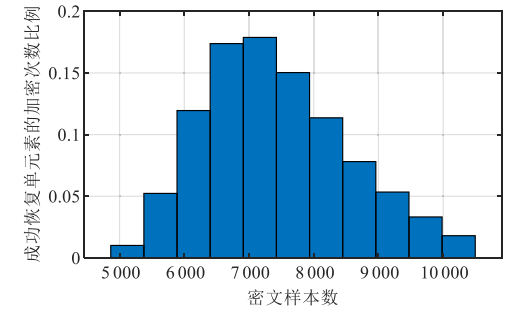


图12 Δ不同取值时恢复一个元素所需加密次数(类SM4算法)

4.2 基于持续性故障逆向恢复S盒表方法的普适性原理讨论

向运行分组密码算法的设备或芯片内注入持续性故障,假设由于故障的注入,算法S盒表位置u的元素由v变为v^*. 在分组密码算法执行过程中,如果第一轮S盒运算输入的某个字节值为u,相应地会得到错误的字节输出v^*参与后续运算,加密过程中间值出错,最终的密文也一定是错误的. 如果第一轮S盒运算没有取到故障值,密文结果只会以一定概率出错,原因在于后续加密过程中S盒运算可能会取到u,从而影响加密结果. 这样的道理同样适用于分组密码算法第二轮函数.

本文基于对不同结构算法的分析抽象出使用该方法逆向恢复分组算法S盒表的一般过程. 在已知u,v^*,u^*且明文和密钥可控的前提下,结合算法已知运算结构的特点,构造特殊的明文和密钥完成多次加密,旨在削弱算法前两轮S盒运算之间其他基本运算组件对第一轮S盒输出的混淆/扩散作用,使第一轮S盒运算输出的某些字节值或其变形能对第二轮S盒运算输入的第0字节产生直接影响.

恢复S盒表单元素的一般流程为:

(1)算法第一轮特定的S盒输入将影响第二轮S盒输入第0字节的取值,将第一轮特定S盒输入取值相同的运算看作一组加密. 通过比较故障注入前后的加密结果,分析故障注入后每一组密文的错误率. 找到一组

密文恒出错,且第一轮S盒运算输入值不等于u的情况,说明这组加密中第二轮S盒运算输入的第0字节为u.

(2)在第二轮S盒运算输入第0字节为u,并且影响该值的轮密钥已知可控的情况下,可以通过计算反向推导出相应的第一轮S盒运算输出值或其变形,这样就恢复出了S盒表的一个元素值y_i.

(3)在明文和密钥已知的情况下,第一轮S盒运算输入的各字节取值可以通过计算得到,这样S盒表元素y_i对应的索引i也能被恢复.

基于此,不断调整算法密钥,进而变换影响第二轮S盒输入的轮密钥取值,在第二轮S盒运算输入的第0字节为u的条件下,可以逆向推导出第一轮S盒运算输出的更多取值情况. 根据已知的明文和密钥找到对应的第一轮S盒运算输入,从而恢复出S盒表的更多元素,直至恢复完整的S盒表.

综上,本文所提出的方法能够逆向恢复各种结构的分组密码算法S盒表,具备普适性的原因在于:方法仅着眼于算法前两轮S盒运算之间的过程,在掌握了仅S盒表未知的算法结构后,通过构造特殊的明文和密钥,使得第一轮S盒运算输出的某些特定字节能够直接影响第二轮S盒运算输入的第0字节取值. 无论基于哪一种结构,也无论算法前两轮S盒运算之间还有哪些基本运算组件,影响第二轮S盒运算输入第0字节的因素,除了已知可控的部分外,就只有第一轮S盒运算的输出值. 由此,本文逆向恢复S盒表元素的方法与算法具体的结构无关.

4.3 基于持续性故障逆向恢复S盒表方法适用的算法场景

为验证有效性,本文将该方法应用于多种分组密码算法. 各算法S盒表的逆向恢复有成功,也有失败. 通过对各算法结构进行分析,本文总结了方法所适用的算法需具备以下几个条件:

(1)在分组密码算法的加密运算过程和密钥扩展过程中,凡是涉及S盒表的运算操作都应使用相同的S盒表.

在逆向恢复类Midori-128算法S盒表的过程中,为使第二轮SubCell运算成功引发故障,确保只有第一轮SubCell运算输出的第15字节对第二轮SubCell运算输入的第0字节取值产生实质性影响,令明文与密钥在第0,5,10这三个字节上的异或值为u^*,这样经过第一轮SubCell运算才能够得到相同的输出v^*,从而削弱这三个字节值对第二轮SubCell运算输入第0字节取值的影响. 但是标准Midori-128算法运算使用了4种S盒表,这三个字节值作为输入进行SubCell运算时使用的S盒表并不相同,这样就无法保证这三个字节值作为输入会得到相同的S盒输出结果,无法控制故障诱导成功的情况,导致S盒表单元素恢复失败. 基于类Midori-128算法全轮运算使用相同S盒表的假设,利用本文方法可以成功恢复出S盒表全部元素.

(2)分组密码算法执行过程中S盒运算的输入值和输出值都是8比特,S盒表中有256个元素,单个S盒运算出错的概率为1/256.

例如,在类轻量级分组密码Present的S盒表逆向恢复中,将第一轮S盒运算输入中某个半字节值相等的情况看作一组加密. 根据实验结果发现故障注入后的密文结果都是错误的,无法通过分析各组密文的错误率找到加密过程恒出错的情况,从而无法恢复出S盒表的一个元素. 针对该情况进行理论分析可以发现,标准Present算法使用的是4比特位宽的S盒,单个S盒操作出错的概率为1/16,Present算法31轮轮函数中共有496个S盒运算,故障注入后Present算法S盒运算错误的概率为$1-1.25\times10^{-14}$. 由此得出,算法使用的S盒位宽小,故障注入后的S盒运算将有较高的概率取到故障值,从而导致密文错误率没有明显的区分,元素恢复失败.

本文针对具有不同S盒表元素数目的算法进行了运算过程的分析,计算了故障注入后S盒运算没有取到故障值,加密过程没有受到故障影响,密文结果正确的概率. 记S盒表输入位数为b,全轮运算迭代数为r,一轮迭代中S盒运算数为t,故障注入后S盒运算正确率为$\left(1-\frac{1}{2^b}\right)^{r\times t}$. 选取有代表性的算法进行了六个指标的比较,具体如表3所示. 其中,DES算法和SM4算法是基于Feistel网络结构的,其余算法是基于SPN结构的;Midori系列算法虽与AES算法等同为SPN结构,但是其密钥扩展过程并不涉及S盒运算. 故这次比较不考虑密钥扩展过程的S盒运算操作,并且对于使用多种S盒的算法,一律按照算法全轮仅使用一种S盒的情况进行比较. 观察分析表中数据,可以发现S盒位宽越小,S盒表中元素数量越少,加密过程S盒表取到故障值导致密文出错的概率越大.

(3)分组密码算法前两轮S盒运算之间的其他运算

表3　各算法S盒表使用情况

算法	S盒规模	S盒表元素数	轮数	每轮S盒运算次数	注入故障后密文正确率
Present	4	16	31	16	1.25×10^{-14}
Midori-64	4	16	16	16	6.68×10^{-8}
DES	6	64	16	8	0.133 2
Midori-128	8	256	20	16	0.285 8
AES	8	256	10	16	0.534 6
SM4	8	256	32	4	0.605 9

组件对第一轮特定的S盒输出产生的混淆/扩散作用能够通过构造特殊明文和密钥的方式降低到最小,使得第一轮特定的S盒运算输出经运算变形后能够对第二轮S盒运算输入的第0字节取值产生直接的清晰的影响. 这样在分析各组加密的错误率并找到加密过程恒出错的情况后,能够通过计算逆向推导出第一轮S盒运算特定字节位置的输出值或其变形,从而恢复出S盒表的一个元素.

例如,标准SM4算法两轮S盒运算之间的线性变换L是以32位字为运算单位的,不是4个字节分别进行运算,这样第一轮线性变换输出的4字节结果中每一字节取值都和线性变换输入的每一个字节息息相关. 在这种情况下,即使找到了加密过程恒出错的情况,利用第二轮S盒运算输入第0字节为u的条件推导出了第一轮线性变换输出的第0字节,也将因为第一轮线性变换输出的其他字节取值未知,而无法明确第一轮线性变换运算的输入,即第一轮S盒运算的输出值,更无法明确是第一轮S盒运算输入的哪个字节对故障诱导起到决定性作用. 通过对线性变换L进行分析,发现当运算输入为4字节相等的值时,相应的4字节输出值也是相等的,且输出字节值是输入字节值的变形. 通过构造特殊的明文和密钥,保证第一轮线性变换输入的四个字节相等,降低线性变换对第一轮S盒运算输出的作用效果,进而明确第一轮S盒运算输出对第二轮S盒运算输入第0字节取值产生的影响. 利用这种办法,本文实验成功逆向恢复了类SM4算法S盒表.

基于以上讨论,为避免受到本文所提出的逆向攻击,针对密码算法设计提出以下三点建议:

(1)使用2种以上S盒表,且令不同种S盒参与同一线性混淆运算,这样即使诱发了第二轮S盒出错,也无法推导出第一轮S盒的输出.

(2)使用轻量级S盒(例如:4比特S盒),且在密钥扩展中加入S盒运算,这样注入故障后密文将以极大概率出错,从而无法探测第二轮S盒出错情况,进而无法恢复第一轮S盒输出.

(3)设计复杂的线性组件,使第二轮多个S盒的输入不一致且都受到第一轮同一S盒的影响,这样即使检测到第二轮S盒出错,也无法恢复出第一轮S盒输出.

5 结论

本文提出了一种基于持续性故障的分组密码算法S盒表逆向分析方法,完成了对类AES-128算法S盒表的逆向恢复工作. 在只注入一次持续性故障的条件下,在仅已知一个元素的情况下平均进行1 441 792次加密就能恢复完整S盒,与其他方法相比我们的方法在故障注入次数和逆向恢复运算复杂度上有明显优势. 另外,本文使用相同的思路解决了类SM4算法S盒表的逆向恢复问题. 在只注入一次持续性故障的情况下,平均加密1 900 544次恢复类SM4算法完整S盒表. 最后,本文结合Feistel和SPN这两种常用分组密码算法网络结构的特点,讨论了新方法的普适性. 通过对多种分组密码算法的调研,我们总结了新方法适用的算法场景需要具备的条件,并分别进行了解释和举例说明.

参考文献

[1] MATSUI M. Linear cryptanalysis method for DES cipher [M]//Advances in Cryptology — EUROCRYPT'93. Berlin, Heidelberg: Springer, 1994: 386-397.

[2] CHO J Y. Linear Cryptanalysis of reduced-round pRESENT[C]//Cryptographers' Track at the RSA Conference. Berlin, Heidelberg: Springer, 2010: 302-317.

[3] GILBERT H, CHAUVAUD P. A chosen plaintext attack of the 16-round Khufu cryptosystem[C]//Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology. Santa Barbara: Springer, 1994: 359-368.

[4] DAEMEN J, KNUDSEN L R, RIJMEN V. The block cipher square[C]//Proceedings of the 4th International Workshop on Fast Software Encryption. Haifa: Springer, 1997: 149-165.

[5] BIRYUKOV A, SHAMIR A. Structural cryptanalysis of SASAS[C]//International Conference on the Theory and Application of Cryptographic Techniques. Innsbruck: Springer, 2001: 394-405.

[6] TIESSEN T, KNUDSEN L R, KÖLBL S, et al. Security of the AES with a secret S-Box[C]//International Workshop on Fast Software Encryption. Berlin, Heidelberg: Springer, 2015: 175-189.

[7] TORRANCE R, JAMES D. The state-of-the-art in IC reverse engineering[C]//International Workshop on Cryptographic Hardware and Embedded Systems. Berlin, Heidelberg: Springer, 2009: 363-381.

[8] QUADIR S E, CHEN J L, FORTE D, et al. A survey on chip to system reverse engineering[J]. ACM Journal on Emerging Technologies in Computing Systems, 2016, 13 (1): 1-34.

[9] NOVAK R. Side-Channel Attack on substitution blocks [C]//International Conference on Applied Cryptography and Network Security. Berlin, Heidelberg: Springer, 2003: 307-318.

[10] CLAVIER C. Side channel analysis for reverse engineering (SCARE)-An improved attack against a secret A3/A8

GSM algorithm[EB/OL]. (2004-01). https://eprint. iacr. org/2004/049.

[11] DAUDIGNY R, LEDIG H, MULLER F, et al. SCARE of the DES[C]//Applied Cryptography and Network Security. Berlin, Heidelberg: Springer, 2005: 393-406.

[12] RÉAL D, DUBOIS V, GUILLOUX A M, et al. SCARE of an unknown hardware feistel implementation[C]//International Conference on Smart Card Research and Advanced Applications. Berlin, Heidelberg: Springer, 2008: 218-227.

[13] RIVAIN M, ROCHE T. SCARE of secret ciphers with SPN structures[C]//International Conference on the Theory and Application of Cryptology and Information Security. Berlin, Heidelberg: Springer, 2013: 526-544.

[14] CLAVIER C, ISOREZ Q, WURCKER A. Complete SCARE of AES-like block ciphers by chosen plaintext collision power analysis[C]//International Conference on Cryptology in India. Cham: Springer, 2013: 116-135.

[15] PEDRO M SAN, SOOS M, GUILLEY S. FIRE: Fault injection for reverse engineering[C]//Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011: 280-293.

[16] TANG M, QIU Z, DENG H, LIU S, ZHANG H. Reverse engineering analysis based on differential fault analysis against secret s-boxes[J]. China Communications, 2012, 9 (10): 10-22.

[17] CLAVIER C, WURCKER A. Reverse engineering of a secret AES-like cipher by ineffective fault analysis[C]// 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography. Piscataway: IEEE, 2013: 119-128.

[18] ZHANG F, LOU X X, ZHAO X J, et al. Persistent fault analysis on block ciphers[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018, 3: 150-172.

[19] CAFORIO A, BANIK S. A study of persistent fault analysis[C]//International Conference on Security, Privacy, and Applied Cryptography Engineering. Cham: Springer, 2019: 13-33.

[20] ZHANG F, ZHANG Y R, JIANG H L, et al. Persistent fault attack in practice[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 2: 172-195.

[21] ZHENG S H, LIU X D, ZANG S J, et al. A persistent fault-based collision analysis against the advanced encryption standard[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 40(6): 1117-1129.

作者简介

王　安　男,1983年1月出生于山东省烟台市,现为北京理工大学网络空间安全学院研究员、博士生导师. 主要研究领域为侧信道分析与防护技术.
E-mail: wanganl@bit.edu.cn

谷　睿　女,1997年02生于黑龙江省伊春市,2021年毕业于北京理工大学计算机学院,硕士阶段主要从事侧信道攻击与分析方向的学习和研究.
E-mail: 18716035993@163.com

丁瑶玲(通讯作者)　女,1987年11月出生于吉林省白城市. 现为北京理工大学网络空间安全学院特别副研究员、硕士生导师. 主要研究领域为侧信道分析与防护技术.
E-mail: dyl19@bit.edu.cn

张　雪　女,1993年出生于山东省临沂市,2016年毕业于山东大学数学学院,现为清华大学高等研究院博士生,主要研究方向为密码方案与其数学困难问题的分析与攻击方法.
E-mail: zhangxue2012_9@163.com

袁庆军　男,1993年1月出生于河北省衡水市. 现为战略支援部队信息工程大学讲师. 从事网络空间安全、侧信道分析方向的研究工作.
E-mail: gcxyuan@outlook.com

祝烈煌　男,1976年9月出生于浙江省衢州市. 现为北京理工大学网络空间安全学院特聘教授、博士生导师. 主要研究领域为密码算法及安全协议、区块链技术、云计算安全、大数据隐私保护等. 中国电子学会会员编号:E190010255M.
E-mail: liehuangz@bit.edu.cn