文章编号:1671-4229(2022)04-0046-07

Elephant-Delirium 算法安全性分析

侯铖安[1,2], 刘美成[1,2]
(1. 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;
2. 中国科学院大学 网络空间安全学院, 北京 100093)

摘 要: 文章主要关注 Elephant-Delirium 算法的安全性分析。Elephant 算法是美国国家标准与技术研究所主导的轻量级密码算法标准最终轮候选算法之一。Elephant 加密算法的内部将密钥通过一个可逆变换扩展为秘密掩码,然后对内部状态使用置换达到混淆和扩散的目的。Elephant-Delirium 是 Elephant 的加密算法实例,采用 Keccak-f[200]置换作为底层置换。文章利用 Keccak-f[200]置换中非线性操作的代数次数为 2 的性质,构造出 5 轮 Keccak-f[200]置换的零和区分器。在此区分器的基础上,文章使用分治法猜测 Elephant-Delirium 算法第 6 轮输出中的秘密掩码,并利用所构造的零和区分器筛选出正确的秘密掩码。在不重用随机数(nonce)的条件下,文章以 100% 的准确率和 100% 的成功率实现了 6 轮 Elephant-Delirium 的密钥恢复攻击,在单核 CPU 上的实际运行时间约为 2.8 s。这是对 Elephant-Delirium 算法的第一个实际密钥恢复攻击。同时,文章利用立方攻击的思想扩展了优化插值攻击,从而将 8 轮 Elephant-Delirium 算法密钥恢复攻击的复杂度从 $2^{98.3}$ 降到了 $2^{95.2}$。

关键词: Elephant 算法; 立方攻击; 优化插值攻击; 密钥恢复
中图分类号: TN 918.1 **文献标志码:** A

On the security analysis of Elephant-Delirium algorithm

HOU Cheng-an[1,2], *LIU Mei-cheng*[1,2]
(1. State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China;
2. School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100093, China)

Abstract: This paper focuses on the security analysis of Elephant-Delirium algorithm. Elephant is one of the candidate algorithms in the finalist of National Institute of Standards and Technology (NIST) lightweight cryptographic (LWC) project. Its encryption algorithm extends the key to the secret masks through an invertible map, and then uses a permutation on the internal states to achieve confusion and diffusion. The Elephant-Delirium algorithm is an instance of Elephant encryption algorithm which uses Keccak-f[200] as its underlying permutation. This paper constructs a 5-round zero-sum distinguisher using the property that the algebraic degree of nonlinear operation in Keccak-f[200] permutation is 2. Based on this distinguisher, we use the divide and conquer method to guess the secret mask in the output of 6-round Elephant-Delirium algorithm and filter out the right secret mask by checking the zero-sum property. As a result, the secret mask can be recovered with 100% accuracy and 100% success rate. This attack is under the nonce-respecting setting and costs about 2.8 seconds to recover all key bits using a single CPU core. This work is the first practical attack on the Elephant-Delirium algo-

收稿日期: 2022-10-08; **修回日期:** 2022-10-15
基金项目: 国家自然科学基金资助项目(62122085,12231015);中国科学院青年创新促进会资助项目
作者简介: 侯铖安(1999—),男,硕士研究生. E-mail:houchengan@iie.ac.cn
* 通信作者. E-mail:liumeicheng@iie.ac.cn
引文格式: 侯铖安, 刘美成. Elephant-Delirium 算法安全性分析[J]. 广州大学学报(自然科学版),2022,21(4):46-52,86.

rithm. Also, we improve the result of optimized interpolation attack on 8-round Elephant-Delirium algorithm with the help of the cube attack. This improvement reduces the complexity from $2^{98.3}$ to $2^{95.2}$.

Key words: Elephant algorithm; cube attack; optimized interpolation attack; key recovery

现代密码学发展至今,已经衍生出对称密码学和非对称密码学(公钥密码学)两大方向。它们的主要区别是在对称密码算法中,发送者和接收者共享同一个秘密密钥;而在非对称密码中,双方需要分别使用不同而又紧密相关的 2 个密钥。对称密码算法在效率方面具有显著的优势,而非对称密码算法则可以提供丰富的安全功能。只有将对称密码算法与非对称算法结合起来,才能满足人们对安全和效率的需求。

传输层安全(Transport Layer Security, TLS)协议综合应用了对称密码和非对称密码算法,常用于安全超文本传输协议(Hyper Text Transfer Protocol Secure,HTTPS),在计算机网络通信安全中起着至关重要的作用。然而,TLS 协议中使用的密码算法在设计和实现上存在漏洞,近年来有许多对 TLS 协议的攻击被提出[1-2]。这些攻击表明现有的密码算法还远远不能满足人们的安全需求。同时也带给密码算法设计者一个重要的设计原则,就是在密码算法中必须同时考虑数据的机密性和完整性,也就是认证加密(Authenticated Encryption,AE)。目前,密码学界已经在研究和标准化认证加密算法方面做出了许多努力。例如,经过 CAESAR 竞赛,ASCON 算法在安全和效率方面展现出了强大的实力[3]。

在医疗、分布式控制系统、物联网等新兴领域,常常需要将一些资源非常有限的设备进行连接,并协同工作。但目前广泛应用的密码算法大都是为个人电脑或服务器环境而设计,因此,不适用于资源受限的设备。在这样的背景下,轻量级密码算法应运而生。美国标准与技术研究所(National Institute of Standards and Technology, NIST)也开始公开征集、评估和标准化轻量级密码算法(Lightweight Cryptography, LWC),以用于其他 NIST 密码标准无法工作的资源受限场景。2021 年 3 月,经过 3 轮筛选,NIST 公布了包括 ASCON 和 Elephant 在内的 10 个算法进入最终的候选列表,并将在接下来一段时间中对这 10 个算法进行更全面而深入地评估。

Elephant 算法是由 Beyne 等提交到 NIST 的 LWC 竞赛的轻量级认证加密算法[4],并于 2020 年发表在 *IACR Transactions on Symmetric Cryptology*(ToSC)。Elephant 算法具有状态小、并行度高等特性,其底层使用了较为成熟的 Spongent 结构和 Keccak-f[200]置换,这些优点使得 Elephant 从 LWC 竞赛的候选算法中脱颖而出,进入到最终列表的评估。因此,分析 Elephant 算法的安全性具有十分重要的理论价值和现实意义。目前,除了设计者在理想置换模型下的安全分析之外,Zhou 等[5]使用优化插值攻击以 $2^{98.3}$ 的复杂度实现了对 8 轮 Elephant-Delirium 实例的密钥恢复攻击,该结果于 2021 年 4 月正式发表在 *The Computer Journal* 上,也是目前为止唯一的第三方分析。本文利用立方攻击的思想改进了这一攻击结果,将时间复杂度降到 $2^{95.2}$。除了理论分析,本文还首次实现了对 6 轮 Elephant-Delirium 实例的实际密钥恢复攻击。

1　预备知识

1.1　布尔函数与莫比乌斯变换

布尔函数即从二元域 $\mathbb{F}_2$ 上的 n 维向量空间 $\mathbb{F}_2^n$ 映射到 $\mathbb{F}_2$ 的函数。布尔函数既可以用它的真值表表示,也可以用它的代数标准型(Algebraic Normal Form, ANF)表示。令 f: $\mathbb{F}_2^n \mapsto \mathbb{F}_2$ 是一个 n 元布尔函数,则其 ANF 是 $\mathbb{F}[x_0,\cdots,x_{n-1}]/(x_0^2 \oplus x_0,\cdots,x_{n-1}^2 \oplus x_{n-1})$ 中的多项式,即

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^u \tag{1}$$

其中,$x = (x_0,\cdots,x_{n-1})$,$u = (u_0,\cdots,u_{n-1})$,$a_u \in \mathbb{F}_2$,$x^u = \prod_{i=0}^{n-1} x_i^{u_i}$。

使用莫比乌斯变换可以将一个布尔函数 f 从它的真值表形式转换为 ANF,也可以反过来从 ANF 转换为真值表。对于一个 n 元布尔函数,莫比乌斯变换需要 $n2^{n-1}$ 次异或运算。

1.2　高阶差分攻击与立方攻击

高阶差分攻击是差分攻击的推广,其理论基

础是布尔函数的高阶导数[6]。令 $F:\mathbb{F}_2^n\mapsto\mathbb{F}_2^m$ 为一个向量布尔函数。对 $\forall v\in\mathbb{F}_2^n$,$F$ 关于 v 的导数定义为 $D_vF(x)=F(x\oplus v)\oplus F(x)$。进一步地,对 $\mathbb{F}_2^n$ 中的任意 k 维子空间 V 和 V 的任意一组基 $\{v_0,\cdots,v_{k-1}\}$,F 关于 V 的 k 阶导数定义为

$$D_VF(x)=D_{v_0}D_{v_1}\cdots D_{v_{k-1}}F(x)=\bigoplus_{u\in V}F(x\oplus u)。$$

对布尔函数的每一次求导都会降低导数的代数次数[6]。因此,对于任何维数大于该函数的代数次数的子空间,其对应的导数为

$$D_VF(x)=\bigoplus_{u\in V}F(x\oplus u)=0 \tag{2}$$

对 $\forall x\in\mathbb{F}_2^n$ 都成立。

对于一个特定轮数的密码算法,如果能够找到某个输入子空间,使得这个子空间的维数超过输出函数的代数次数,那么对这个输入子空间对应的输出子空间进行求和将恒等于0,也就是得到了一个零和区分器,然后可以利用这个零和区分器来实现密钥恢复攻击。不难看出,输出函数的代数次数对高阶差分攻击至关重要。但一般而言,需要很高的复杂度才能得到输出函数的准确次数。因此,在实际中人们常常只是估计输出函数代数次数的上界。对于一个轮函数的代数次数为 d 的密码算法,由于第二轮输入的代数次数为 d,第二轮的输出函数关于第二轮输入的代数次数也为 d,因此,第二轮输出函数关于第一轮输入的代数次数不超过 d^2,由归纳法可知,经过 r 轮迭代后的输出函数的代数次数不会超过 d^r。例如,Keccak-f[200]的代数次数为2,那么经过4轮后输出函数的代数次数不会超过16。因此,根据高阶差分性质,任意超过16维的子空间都对应一个4轮的零和区分器。

立方攻击是高阶差分攻击的一种变体,由 Dinur 等[7]在2009年的欧密会上提出。立方攻击作为一种用来分析对称密码算法的通用方法,已经成功用于攻击许多密码体制。立方攻击将输出比特的 ANF 看作是关于 n 个秘密变量 $x=(x_1,x_2,\cdots,x_n)$ 和 m 个公开变量 $v=(v_1,v_2,\cdots,v_m)$ 的多项式,记为 $f(x,v)$。对于任何一个集合 $I=\{i_1,i_2,\cdots,i_{|I|}\}\subset\{1,\cdots,m\}$,$f(x,v)$ 可以写作

$$f(x,v)=t_Ip(x,v)\oplus q(x,v),$$

其中,$t_I=v_{i_1}v_{i_2}\cdots v_{i_{|I|}}$ 称为极大项,极大项中的变量 $v_{i_1},v_{i_2},\cdots,v_{i_{|I|}}$ 称为立方变量。$p(x,v)$ 不含有立方变量,称为 t_I 对应的超级多项式,且 $q(x,v)$ 至少比 t_I 少一个变量。

立方攻击的核心是选取立方变量来构建立方集,对应于高阶差分攻击中选择输入子空间。立方集是 $\mathbb{F}_2^m$ 的一个仿射子空间,记为 C_I。C_I 在立方变量上取所有 $2^{|I|}$ 种可能值,而在其他公开变量上为定值。Dinur 等[7]证明了对 C_I 中所有 $2^{|I|}$ 个 m 维向量经过 f 映射后进行求和,其结果恰好是对应的超级多项式:

$$p(x,v)=\bigoplus_{v\in C_I}f(x,v) \tag{3}$$

恰当地选择立方变量将使得超级多项式在变量和次数方面都比原多项式简单得多,因此,更有利于分析函数的性质。

1.3 优化插值攻击

优化插值攻击[8]是对插值攻击[9]的改进,利用莫比乌斯变换降低了攻击的复杂度。

插值攻击考虑一个目标比特 a,其 ANF 可以用密文 C 和密钥 K 表示,即

$$a=F_K(C)=F_K(c_1,\cdots,c_n)=\sum_{u=(u_1,\cdots,u_n)\in\mathbb{F}_2^q}\alpha_uM_u,$$

其中,$\alpha_u\in\{0,1\}$ 是单项式 $M_u=\prod_{i=1}^n c_i^{u_i}$ 的系数,且仅与密钥 K 有关。为了恢复所有的 α_u,即恢复等效密钥,可以将其看作未知变量,然后构建线性方程组来求解。如果 $F_K(C)$ 的代数次数不超过 d,那么根据高阶差分特征,任何 $d+1$ 维明文子空间 V_P 对应的密文子空间 V_C 求和为零:

$$\sum_{C\in V_C}F_K(C)=\sum_{u=(u_1,\cdots,u_n)\in\mathbb{F}_2^q}\alpha_u\times\Big(\sum_{C\in V_C}M_u\Big)=0。$$

这是一个关于 α_u 的线性方程。设 $F_K(C)$ 中非零的 α_u 个数为 N_{α_u},则需要 N_{α_u} 个线性独立的方程才能解出所有的 α_u。因为一个子空间 V_C 给出一个线性方程,所以需要构建 N_{α_u} 个不同的子空间来提供足够的方程。这些子空间可以从更大的子空间 $V_P'\supset V_p$ 中进行组合,且 V_P' 的维数 $\dim(V_P')=d+1+e$ 应当满足 $\binom{d+1+e}{d+1}\geqslant N_{\alpha_u}$。

注意到对于每一个 u 都需要对 V_C 中的所有 M_u 进行求和。一共有 N_{α_u} 个子空间,因此,总的时间复杂度为 $N_{\alpha_u}\times N_{\alpha_u}\times 2^{\dim(V_C)}=N_{\alpha_u}^2\times 2^{d+1}$。优化插值攻击利用莫比乌斯变换将这个复杂度降到

$N_{\alpha_u} \times 2^{d+e} \times \log_2(d+1+e)$,其主要思想是对 N_{α_u} 个不同的 V_C 同时计算 $\sum_{C \in V_C} M_u$。首先,考虑明文 $P \in V_P$,因为密钥是未知但固定的,所以密文是关于 P 的函数,因此, $\sum_{C \in V_C} M_u = \sum_{P \in V_P} M_u(P)$。从 V'_P 的角度看,这是对子空间 V_P 的求和,那么得到的结果即是 V_P 对应的超级多项式。只需要令 $V'_P \setminus V_P$ 的 e 个比特全部为 0,就可以将超级多项式简化为常数 0 或者 1。如果已经通过莫比乌斯变换得到了 M_u 关于 V'_P 的 ANF,那么 $\sum_{P \in V_P} M_u(P)$ 就对应于 ANF 中 V_P 的超级多项式。而且对于所有的 V_P,都可以用这样的方法直接从 ANF 中得到 $\sum_{P \in V_P} M_u(P)$。因此,这个过程的复杂度就等于用莫比乌斯变换求 M_u 关于 V'_P 的 ANF 的复杂度。因为 V'_P 共有 $d+1+e$ 个变量,所以需要 $2^{d+e} \times \log_2(d+1+e)$ 次异或操作。

2　Elephant 密码算法

Elephant 算法是由 Beyne 等[4]提交到 NIST 的 LWC 竞赛的轻量级认证加密算法,是 LWC 竞赛最终列表中的 10 个算法之一。Elephant 算法的结构是基于 nonce 的先加密后认证的构造,根据底层所用置换的不同, Elephant 算法共有 Dumbo、Jumbo 和 Delirium 3 个实例,分别使用的是 Spongent-π[160]、Spongent-π[176]和 Keccak-f[200]置换。每一个实例都使用一个置换和一个线性反馈移位寄存器来将密钥扩展为掩码。这个过程是可逆的,如果可以恢复出秘密掩码,那么就可以直接逆推得到密钥。本文关注的是使用 Keccak-f[200]置换的 Delirium 实例,此后所说的 Elephant 算法都特指 Delirium 实例。

Elephant 算法的内部状态有 200 比特,初始时用 96 比特随机数和 104 比特的 0 进行填充。然后,将密钥扩展后的 200 比特掩码异或到初始状态上。经过 18 轮 Keccak-f[200]置换后,再将掩码异或到所有状态上,得到输出状态。对于一个 200 比特的明文分组,异或输出状态后得到相应的密文。本文只使用一个明文分组进行选择明文攻击,所以略去了 Elephant 算法关于分组的处理和认证阶段。

Keccak-f[200]置换有 200 比特内部状态,总共 18 轮。这 200 比特的状态 $X \in \{0,1\}^{200}$ 被表示为一个 $5 \times 5 \times 8$ 的数组 $a \in \{0,1\}^{5 \times 5 \times 8}$。对于数组 a 中下标为 $(x,y,z) \in \mathbb{Z}_5 \times \mathbb{Z}_5 \times \mathbb{Z}_8$ 的元素 $a_{x,y,z}$,对应的是 X 的第 $8 \cdot (5y+x)+z$ 比特,即 $a_{x,y,z} = X[8 \cdot (5y+x)+z]$。Keccak-$f$[200]的每一轮置换都对所有 200 比特状态依次进行 5 个操作 θ,ρ,π,χ,ι,其中,χ 是唯一的非线性操作,它们的定义如下:

$$\theta: a_{x,y,z} \leftarrow a_{x,y,z} + \sum_{y'=0}^{4} (a_{x-1,y',z-1} + a_{x+1,y',z}),$$

$$\rho: a_{x,y,z} \leftarrow a_{x,y,z} + t_{x,y},$$

$$\pi: a_{x,y,z} \leftarrow a_{x+3y,x,z},$$

$$\chi: a_{x,y,z} \leftarrow a_{x,y,z} + (a_{x+1,y,z} + 1) a_{x+2,y,z},$$

$$\iota: a_{x,y,z} \leftarrow a_{x,y,z} + RC_{i,x,y,z},$$

以上运算均在 $\mathbb{F}_2$ 中进行,其中,$t_{x,y}$ 和 $RC_{i,x,y,z}$ 都是给定的常数,i 表示轮数。

注意到 Keccak-f[200]中的非线性操作 χ 的代数次数只有 2 次,这一性质被许多攻击所利用。

3　6 轮 Elephant 算法的实际密钥恢复攻击

由于 Elephant 算法的非线性操作可以看作 40 个并行的 5 比特 S 盒,输出时异或的秘密掩码可以看作是轮密钥,因此,本文采用了分组密码分析中常见的区分器 - 解密的攻击方式进行密钥恢复攻击。具体来说,如果有一个 r 轮的区分器,那么首先猜测轮密钥,然后将 $r+1$ 轮的输出进行解密。如果猜测的轮密钥恰好是正确的轮密钥,那么解密出来的结果恰好是 r 轮的输出,因此,满足区分器的性质。另外,如果猜测的轮密钥是错误的,则解密出来的结果近似于均匀分布,因此,不能观察到区分器的性质。由于 40 个 S 盒是相互独立的,因此,可以利用分治法来恢复每个 S 盒对应的轮密钥,也就是 Elephant 算法中的秘密掩码。

3.1　5 轮零和区分器的构造

基于 Keccak-f[200]置换的代数次数为 2 的特点,本文利用高阶差分性质来构造零和区分器。Elephant 算法加密时的初始状态是用 96 比特的随机数(nonce)和 104 比特的 0 进行填充的,然后该初始状态异或 200 比特的秘密掩码。其中,nonce 是可以控制的,即可以选择的子空间的最大维数

是96维。根据高阶差分性质,任何65维的子空间都可以构造一个6轮的零和区分器,因为$65 > 2^6$。由于这个零和区分器的复杂度远远超过了人们的计算能力,所以只能降低攻击的轮数。一个直接的想法就是用33维的子空间构造5轮的零和区分器,或者用17维的子空间构造4轮的零和区分器。事实上,可以利用CP-kernel性质(CP-kernel性质是Keccak的设计者提出的θ操作的一个性质),将使用17维子空间的4轮零和区分器推进到5轮。

要将4轮的零和区分器推进1轮,则17维的子空间必须满足一定的条件。可以从代数次数的角度来看这个问题:之所以可以用17维的子空间得到4轮的零和区分器,是因为4轮输出函数的代数次数至多为16,而17维的子空间对应的极大项是一个17次项,因此,其系数必然为0。如果选择17维子空间对应的立方变量,在经过1轮置换后没有乘出二次项,则再经过4轮置换,其输出函数中不可能出现17个变量的乘积项,所以子空间的求和仍然是0。从另一个角度说,在遍历17维输入子空间的时候,恰好遍历了1轮置换的某个17维输出子空间。而这个1轮置换的17维输出子空间,又可以看作是4轮零和区分器的输入子空间,也就必然满足零和的性质。综上所述,本研究的问题转化为寻找满足条件的17维子空间。

这个问题可以使用混合整数线性规划模型进行建模。首先,用$(x_0,\cdots,x_{95})$,$x_i \in \mathbb{F}_2$来表示可以控制的96比特随机数(nonce)。当$x_i = 1$时,表示将其选为立方变量,否则表示不选为立方变量。其次,本研究的目标是最大化子空间的维数,也就是最大化立方变量的个数,即$\sum_{i=0}^{95} x_i$。约束条件是经过1轮置换后,立方变量之间不会相互乘出二次项,这需要借助1轮Keccak-f[200]的ANF来给出。将所有的随机数比特都用一个布尔变量来表示,其他比特赋值为0。然后,使用SageMath软件进行符号计算,得到1轮Keccak-f[200]关于nonce的ANF。所得的ANF中包含一些二次项,每一个二次项,不妨设为$x_i x_j$,给出一个约束条件$x_i + x_j \leqslant 1$,即这2个变量不能同时选为立方变量。最后,使用Gurobi软件求解这个MILP模型,即可得到满足条件的子空间的最大维数,给出所选的立方变量。

在选择立方变量时,必须考虑Keccak-f[200]的CP-kernel性质,因为不考虑CP-kernel性质时,以上模型给出的子空间的最大维数只有8维,不能满足5轮零和区分器所要求的17维子空间。观察θ操作的表达式不难发现,如果将$a_{x,y,z}$和$a_{x,y',z}$,$y' \neq y$同时选为相等的立方变量。则在它们的和中,立方变量将被消掉。这一性质是异或的直接结果,称为CP-kernel性质。θ操作的目的是提供扩散,而CP-kernel性质阻止了扩散。因为ρ和π操作都只是比特位置的移动而不存在扩散,所以CP-kernel性质使得Keccak-f[200]的θ,ρ,π操作退化为比特置换。这样一来,就有可能找到更大维数的子空间。实验的结果也验证了这一点,在考虑了CP-kernel性质后,子空间的最大维数达到28维,因此,可以从中选取17维的子空间来构造零和区分器。

3.2 攻击过程

本研究的攻击是一个选择明文攻击,只使用一个明文块。不妨固定$M = 0^{200}$,则加密得到的密文$C = P(N \parallel 0^{104} \oplus mask_K) \oplus mask_K$。其中,$P$是约简轮数的Keccak-$f$[200]置换,$mask_K = P(K \parallel 0^{72})$是由密钥导出的秘密掩码。

因为θ,ρ,π操作都是线性操作,不会破坏零和的性质,所以5轮的零和区分器可以直接推进到第6轮的χ操作之前。那么,距离6轮的输出,还有χ,ι和异或秘密掩码3个操作。其中,ι操作只是异或了第6轮的轮常数,可以等效到异或秘密掩码的操作中。因此,需要考虑χ和异或等效掩码2个操作。

注意到χ操作等价于40个并行的5比特S盒,如果猜测一个S盒的输出,相当于猜测5比特的掩码,那么就可以对这5比特计算S盒的逆。当猜测正确时,则得到的结果就是这个S盒的输入,那么在这5个输出比特上,都应该满足零和的性质:

$$\sum_{C^i \in V_C^i} \chi^{-1}(C^i \oplus mask_K^i) = (0,0,0,0,0) \quad (4)$$

其中,$i = 0,\cdots,39$表示第i个S盒,$mask_K^i$表示第i个正确掩码。

反之,如果猜测错误,则得到的结果近似于均匀分布,那么5个比特同时满足零和的概率近似于2^{-5}。在实验中发现,对于每一个S盒,除了正确的掩码,还可能有至多3个错误的掩码也满足

零和的性质。为了降低错误的概率,使用 3 个 17 维的子空间。假设错误的结果为均匀分布,则错误的掩码同时在 3 个子空间都满足零和的概率为 2^{-15}。实验结果表明,使用 3 个不同的零和区分器,只有正确的掩码才能同时满足零和的性质。

对所有 40 个 S 盒都进行以上过程,就可以恢复出 200 比特的掩码,进而恢复出密钥。

整个攻击的过程如算法 1 所示。

算法 1:对 6 轮 Elephant 算法的密钥恢复攻击

输入: MILP 模型给出的任意 18 个立方变量 $I = \{x_0,\cdots,x_17\}$

输出:200 比特秘密掩码 $mask_K$

遍历 18 个立方变量的取值,任意给定 nonce 其他比特的取值,构建集合 N_I

对 N_I 中的每一个 nonce,设置明文 $M = 0^{200}$,访问加密函数得到对应的密文,并异或轮常数,得到集合 C_I

for s←0 to 39 do

  for i←0 to 2^5 do

    $sum0 \leftarrow \sum_{k=0}^{2^{26}} \chi^{-1}(C^s_{k,x_{16}=0,x_{17}=0} \oplus i)$;

  //$C^s_{k,x_{16}=0,x_{17}=0}$表示 C_I 中当输入为 $x_j = k_j$, $j \in [0,15]$, $x_{16} = x_{17} = 0$ 时的结果中第 s 个 S 盒的 5 个比特,其中,k_j 表示 k 的第 j 比特

    $sum1 \leftarrow \sum_{k=0}^{2^{26}} \chi^{-1}(C^s_{k,x_{16}=0,x_{17}=1} \oplus i)$;

    $sum2 \leftarrow \sum_{k=0}^{2^{26}} \chi^{-1}(C^s_{k,x_{16}=1,x_{17}=0} \oplus i)$;

    $sum3 \leftarrow \sum_{k=0}^{2^{26}} \chi^{-1}(C^s_{k,x_{16}=1,x_{17}=1} \oplus i)$;

    if $sum0 \oplus sum1 = 0$ and $sum0 \oplus sum2 = 0$ and $sum2 \oplus sum3 = 0$ then

      $mask^s_K = i$

    //$mask^s_K$ 表示秘密掩码中对应第 s 个 S 盒的 5 个比特

    end if

  end for

end for

3.3　攻击复杂度分析

总共需要对 3 个 17 维子空间,这可以从 18 维子空间中选取。而且,这 3 个 17 维子空间可以共享同一个 16 维子空间,这样一来,3 个子空间的并集大小为 2^{18},而不是 3×2^{17}。我们需要 2^{18} 的数据复杂度,同时存储这些数据需要 2^{18} 的空间复杂度。

然后,对 40 个 S 盒分别进行猜测,每个 S 盒的输出有 5 比特,共 2^5 种可能性。对于每一种可能的取值,需要在 2^{18} 的数据上进行求和,然后根据是否满足零和性质来筛选出正确的掩码。因此,攻击的时间复杂度为 $40 \times 2^5 \times 2^{18} \approx 2^{29}$。

3.4　攻击的实现

为了验证以上理论,使用 C ++ 编程实现了攻击。随机生成了超过 100 个密钥,然后调用 Elephant 的参考代码进行加密,最终以 100% 的成功率恢复出所有的秘密掩码。实验在装有 Ubuntu Server 21.04 操作系统的台式机上运行,使用单核 CPU(AMD Ryzen7-3700X,3.6 GHz)。本文所实现的串行版本,计算 1 个 17 维的子空间对应的密文空间,需要大约 1.2 s。在准备好所需的 nonce-密文对后,平均一次密钥恢复攻击需要大约 2.8 s 时间。同时,使用并行计算还可以进一步减少运行时间,既可以对 40 个 S 盒并行执行,也可以在每次判断零和时并行执行。

4　8 轮 Elephant 算法的密钥恢复攻击

本文的攻击是对 Zhou 等[5]工作的改进。Zhou 等对 8 轮的 Elephant 算法实现了优化插值攻击。其攻击模型以第 6 轮的输出比特为目标建立方程,采用了 6 + 2 的模式,即 6 轮的零和区分器加上从第 8 轮输出逆向计算的 2 轮 Keccak-f[200]。一个 65 维的子空间可以给出一个 6 轮的零和区分器,也就可以与逆向计算的 2 轮 Keccak-f[200]结合建立一个方程。对于逆向计算的 2 轮 Keccak-f[200],Zhou 等利用了 Chaigneau 等[10]所提出的线性化方法,证明了可以使用不超过 $2^{23.1}$ 个变量来将原多项式进行线性化。因此,一共需要 $2^{23.1}$ 个 65 维的子空间来构建方程组。因为 $\binom{70}{65} > 2^{23.1}$,所以 70 维的子空间可以提供足够多的 65 维子空间。根据 1.3 小节的结果,这一攻击过程的复杂度为 $2^{23.1} \times 2^{69} \times \log_2 70 \approx 2^{98.3}$。

4.1　攻击过程

本文同样考虑第 6 轮的输出比特。一方面,可以逆向计算 2 轮 Keccak-f[200]得到关于第 8 轮输出的密文表达式;另一方面,可以计算立方和得到关于输入明文的超级多项式。本文的改进主要

针对6轮的零和区分器。通过立方攻击放宽了零和的限制,从而可以使用更小维数的子空间。一个子空间的超级多项式可以给出一个方程:

$$\sum_{C\in V_C} F_K(C) = G_K(V_P),$$

其中,$G_K(V_P)$是V_P对应的超级多项式,V_C是V_P加密后得到的密文空间。

此时,等式左右两边都是非线性的多项式,因此,都必须进行线性化。对于等式的左边,使用与Zhou等[5]相同的方法来进行线性化,即需要引入$2^{23.1}$个新变量来线性化。因此,只需要考虑等式右边的超级多项式的线性化。

首先,从立方攻击的角度来说,如果将其他的公开变量都给定值,因为6轮的输出函数至多是64次的,所以对一个62维的子空间进行求和,得到的超级多项式是关于秘密变量的2次多项式。其次,考虑超级多项式中非线性项的个数。因为所有的二次项都是2个秘密变量的乘积,所以可以很容易地给出一个上界,即所有秘密变量两两组合的总数为$\binom{200}{2}\approx 2^{14.3}$,那么,每个超级多项式中的二次项都是这些二次项的一个子集。也就是说,通过引入$2^{14.3}$个新变量,可以线性化所有的超级多项式。再加上线性化逆向计算的2轮Keccak-f[200]使用了$2^{23.1}$个新变量,总的变量个数为$2^{23.11}$。为了求解这些变量,需要建立$2^{23.11}$个方程,也就需要$2^{23.11}$个62维子空间。由于$\binom{67}{62} > 2^{23.11}$,所以,67维的子空间可以提供足够多的62维子空间来建立方程组。

对于这$2^{23.11}$个62维子空间,需要预计算出对应的超级多项式,以确定方程组的形式。恢复超级多项式是一个离线的过程,只需要计算一次。每个超级多项式中至多含有200 - 62 = 138个变秘密变量,因此,其项数至多有$\binom{138}{2}+\binom{138}{1}\approx 2^{13.3}$。为了重建超级多项式,可以使用待定系数法来恢复每一项的系数,这需要求解$2^{13.3}$个变量的线性方程组。将这一步骤对$2^{23.11}$个子空间都分别执行一次,即可得到所有的$G_K(V_P)$。

在线阶段,攻击的步骤与Zhou等[5]类似。首先,对67维子空间的所有nonce和固定的明文$M=0^{200}$加密得到对应的密文,用这些密文计算出M_u,得到一个比特向量;然后对这个比特向量做莫比乌斯变换,从所得的结果中去除子空间对应的系数,即是一个等效密钥的系数。在遍历了$2^{23.11}$个子空间后,即可求解方程组得到所有等效密钥的值,进而恢复出密钥比特。

4.2 攻击复杂度分析

与Zhou等[5]的攻击相比,本研究的攻击使用了维数更小的子空间和更多的变量。在离线阶段,需要恢复$2^{23.11}$个超级多项式,每个超级多项式需要求解$2^{13.3}$个变量的线性方程组。在线阶段时,需要遍历67维子空间,并存储所得的密文。因此,数据复杂度和空间复杂度都是2^{67}。计算一次莫比乌斯变换需要$2^{66}\times\log_2 67$次异或操作,总共需计算$2^{23.11}$次莫比乌斯变换。因此,总的攻击复杂度约为$2^{23.11}\times 2^{66}\times\log_2 67\approx 2^{95.2}$。

5 总结与展望

本文对Elephant-Delirium算法的安全性进行分析,提出了6轮的实际密钥恢复攻击并改进了8轮的密钥恢复攻击。

对于6轮的实际密钥恢复攻击,采用了区分器-解密的攻击方式。首先,利用底层的Keccak-f[200]置换代数次数为2的特点,使用17维子空间构造出4轮零和区分器;然后,利用MILP模型并借助Keccak-f[200]置换中θ操作的CP-kernel性质阻止立方变量的扩散,使得子空间对应的17个立方变量在经过1轮Keccak-f[200]置换后仍然是1次,从而得到5轮零和区分器。利用构造出的5轮零和区分器对猜测的秘密掩码的正确性进行区分,最终恢复出所有的秘密掩码。

另外,利用立方攻击的思想改进了8轮密钥恢复攻击的结果。在优化插值攻击中使用超级多项式来建立方程,扩展了现有的使用零和区分器的方法。将攻击的时间复杂度从$2^{98.3}$降低到$2^{95.2}$,空间复杂度和数据复杂度也都有降低。值得注意的是,本研究的方法依然可能存在改进空间。例如本文对超级多项式的项数上界只是粗略的估计,如果可以更加精确地估计这个上界,攻击的复杂度也会随之降低。

(下转第86页)