软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(6):1829−1838 [doi: 10.13328/j.cnki.jos.005806]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

对简化版 MORUS 算法的改进动态立方攻击*

李俊志, 关 杰

(信息工程大学,河南 郑州 450001)

通讯作者: 李俊志, E-mail: lijunzhi1998@163.com

摘 要: MORUS 算法是 Wu 等人设计的认证加密算法,现已进入 CAESAR 竞赛的第三轮.动态立方攻击是 Dinur 等人 2011 年提出的针对迭代型序列密码的分析方法.提出了一种改进的动态立方攻击方法,优化了动态立方攻击的立方集合的选取规则,提出了优先猜测关键值并恢复相应的关键秘密信息的方法,据此给出了成功率更高的秘密信息恢复方法.利用该方法分析了初始化 5 步的简化版 MORUS 算法,最终以 $O(2^{95.05})$的复杂度恢复所有 128 比特密钥,攻击的成功率大于 92%.

关键词: MORUS;动态立方攻击;认证加密;流密码;CAESAR

中图法分类号: TP309

中文引用格式: 李俊志,关杰.对简化版 MORUS 算法的改进动态立方攻击.软件学报,2020,31(6):1829−1838. http://www.jos.org.cn/1000-9825/5806.htm

英文引用格式: Li JZ, Guan J. Improved dynamic cube attack on reduced MORUS. Ruan Jian Xue Bao/Journal of Software, 2020, 31(6):1829−1838 (in Chinese). http://www.jos.org.cn/1000-9825/5806.htm

Improved Dynamic Cube Attack on Reduced MORUS

LI Jun-Zhi, GUAN Jie

(Information Engineering University, Zhengzhou 450001, China)

Abstract: MORUS is a third-round candidate authenticated cipher of CAESAR designed by H. Wu *et al*. Dynamic cube attack presented by Itai Dinur *et al*. recovers the secret key of a cryptosystem by exploiting distinguishers given by cube testers. This study proposes an improved dynamic cube attack by optimizing standard of choosing cubes and method of recovering secret expressions. Based on which, a technique is presented for recovering crucial secret expressions preferentially with high successful rate. Then, MORUS of reduced to 5 steps is attacked utilizing improved dynamic cube attack, and the key of 128-bit is recovered with data complexity of $O(2^{95.05})$ and the successful rate of 92% at least.

Key words: MORUS; dynamic cube attack; authenticated encryption; stream cipher; CAESAR

MORUS 算法[1]是 2014 年由 Wu 等人提出的认证加密算法,该算法提交到了 CAESAR 竞赛[2]中,并经过层层筛选进入了竞赛的第 3 轮.MORUS 算法有 MORUS-640-128 和 MORUS-1280-256 两个版本,前者的内部状态为 640 比特,密钥规模为 128 比特;后者的内部状态 1 280 比特,密钥规模 256 比特.本文分析的对象是 MORUS-640-128-v1.MORUS 可以完成加密和认证两个方面的工作,其加密过程类似流密码,将生成的密钥流与明文进行异或,认证过程将消息注入到内部状态中参与运算,生成或验证认证码.算法由循环移位、与和异或这 3 种运算组成,适用于硬件实现同时具有优良的软件实现效率.

MORUS 算法提出以后,由于其高效的软硬件实现效率和简洁的结构受到人们的广泛关注,但是由于其内部状态规模较大(640 比特),初始化迭代轮数较多(16 轮),并没有出现比较好的分析结果.文献[3]对初始化 3 步的

* 基金项目: 国家自然科学基金(61572516, 61602514)
Foundation item: National Natural Science Foundation of China (61572516, 61602514)
收稿时间: 2017-03-30; 采用时间: 2018-09-22

简化版 MORUS 算法进行了区分攻击和密钥分割攻击,其中,区分攻击的复杂度可以忽略,密钥分割攻击的复杂度为 $O(2^{106.8})$.此外,作者利用差分自动推演方法给了对初始化 4 步的简化版 MORUS 算法的差分-区分攻击,所需数据量为 $O(2^{105})$.文献[4]给出了对 MORUS 算法在 SAT 攻击下的一些结果.文献[5]给出了对 4 轮 MORUS 算法的立方攻击(cube attack).文献[6]利用可分性的方法给出了对 5 轮 MORUS 算法的立方攻击.

立方攻击[7]是 2009 年由 Dinur 和 Shamir 提出的,该分析方法将目标算法看成黑盒多项式,没有利用算法的结构信息,可以用于分析内部结构未知的算法.目前,该方法已被应用于分析序列密码、分组密码和杂凑函数等的安全性,对 Trivium、Grain 等算法取得了较好的攻击结果[8,9].动态立方攻击(dynamic cube attack)[10]由 Dinur 和 Shamir 在 2011 年提出,它可以看成是立方攻击的改进,其原理如下:猜测一部分和密钥有关的信息,并根据算法在立方测试(cube test)中表现出来的不同性质区分出正确的猜测值,从而恢复出密钥信息.由于它利用了算法的结构特点,往往可以取得比单纯立方攻击更好的效果.动态立方攻击在序列密码、分组密码和杂凑函数的分析中得到应用,其中,对满轮的 Grain-128 的 2^{118} 大小的密钥子空间,可以低于穷举攻击 2^{15} 倍的复杂度恢复出所有密钥[10],对杂凑函数 Keccak 取得了目前最好的分析结果[11],在分析分组密码 KATAN32 和 SIMON32 时也取得了较好的效果[12].

本文提出一种改进的动态立方攻击方法,优化了动态立方攻击的立方集合的选取规则,提出了优先猜测关键值并恢复相应的关键秘密信息的方法,据此给出了成功率更高的秘密信息恢复方法.利用该方法分析了初始化 5 步的简化版 MORUS 算法,最终以 $O(2^{95.05})$的复杂度恢复所有 128 比特密钥,攻击的成功率大于 92%.

1 基础知识

1.1 立方攻击

立方攻击目前已被应用于分析序列密码、分组密码和杂凑函数等的安全性.立方攻击是一种选择明文攻击(对于分组密码)或选择 IV 攻击(对于序列密码),也是一种密钥恢复攻击,可以应用于密码算法的内部结构未知的条件下.

立方攻击可以实施的条件是密码算法的输出比特可以表示成关于密钥比特和明文比特(对于分组密码)或 IV 比特(对于序列密码)的次数较低的多变量多项式,这个多项式称为主多项式.

令 $p(x_1,\ldots,x_n,v_1\ldots,v_m)$为主多项式在 $GF(2)$上的代数正规型,其中,$x_i(1<i<n)$代表公开变量(明文比特、IV 比特),$v_j(1<j<m)$代表密钥比特.主多项式有如下分解:

$$p(x_1,\ldots,x_n,v_1,\ldots,v_m)=t_I\cdot p_{S(I)}\oplus q(x_1,\ldots,x_n,v_1,\ldots,v_m) \tag{1}$$

其中,t_I是只含有公开变量的单项式,t_I中变量下标的集合 $I(I\subset\{1,2,\ldots,n\})$称为立方集合,$p_{S(I)}$称为 I 在 p 中的超级多项式(superpoly).在上述分解中,$p_{S(I)}$中变量的下标都不在立方集合 I 中,并且 q 中的任一单项式中都至少缺少下标在 I 中的一个变量.

下面给出包含立方攻击主要思想的定理.

定理 1[7]. 如果主多项式有如下分解:

$$p(x_1,\ldots,x_n,v_1\ldots,v_m)=t_I\cdot p_{S(I)}\oplus q(x_1,\ldots,x_n,v_1\ldots,v_m) \tag{2}$$

则 $p_{S(I)}=\bigoplus_{x_{i|i\in I}\in\{0,1\}} p(x_1,\ldots,x_n,v_1,\ldots,v_m)$.如果此时有 $\deg(p_{S(I)})=1$,则称 t_I是一个最大单项式(maxterm),$p_{S(I)}$即为线性超级多项式.

对密码算法的立方攻击可以分为预处理阶段和在线攻击阶段两个阶段:在预处理阶段,攻击者需要找到合适的立方集合,然后寻找尽可能多的最大项和对应的线性超级多项式,并将其组成方程组;在线攻击阶段,攻击者求出预处理阶段得到的线性超级多项式的值,并解方程组恢复密钥.

1.2 立方测试

立方测试的原理也是通过公开变量的值来评估超级多项式,只不过它的目的不是恢复密钥,而是通过立方测试将密码算法和随机函数区分开,或者是根据超级多项式的某些代数性质检测出它的非随机性.一种常用的

立方测试是平衡性测试,根据超级多项式的 0,1 不平衡性,将其与随机函数区分开.此外,其他有效的区分性质包括低代数次数、线性变量和中性变量等.

1.3 动态立方攻击

动态立方攻击的原理如下:猜测一部分和密钥有关的信息,并根据算法在立方测试中表现出来的不同性质区分出正确的猜测值,从而恢复出密钥信息.在文献[10]中,作者给出了进行动态立方攻击的一般步骤,概括如下.

算法 1. 动态立方攻击[10].

- Step 1. 选择需要消去(即使这个内部状态取 0)的内部状态比特,并给出该内部状态比特取 0 时动态变量的取值或条件;
- Step 2. 选择一个进行立方测试时区分效果较好的立方集合,根据所选的立方集合确定需要猜测的秘密变量的表达式,以便在立方求和时可以计算动态变量的值;
- Step 3. 对每个秘密变量可能的猜测值,将动态变量设置成合适的值,在 Step 2 中得到的立方集合的阶较大的立方子集上进行立方求和,将针对不同立方子集对应的立方和构成一个集合,计算集合中元素取值的重量(其中取值为 1 的元素个数),根据重量取值按照由小到大的顺序进行排序,得到一个由重量及相应猜测值构成的有序表;
- Step 4. 根据 Step3 中得到的有序表,给出秘密变量可能的候选值(一般选择表中排序靠前的部分作为秘密变量的候选值),进而求出部分或所有秘密变量的值.

其中:前两步是离线阶段,攻击者具有控制密钥的能力,其复杂度为预计算的复杂度;后两步为在线阶段,目的是恢复密钥.

1.4 改进的动态立方攻击

我们在对动态立方攻击研究时发现,算法 1 的 Step 2 中,立方集合的选取规则和 Step 4 中秘密变量的恢复方法可以进行改进.我们尝试优化了动态立方攻击的立方集合的选取规则,提出了优先猜测关键值并恢复相应的关键秘密信息的方法,据此提出了一种改进的动态立方攻击方法,从而提高了原始攻击的成功率并降低了攻击的复杂度.具体改进如下.

观察 1. 虽然算法 1 的 Step 2 中选取的立方集合对猜测值的区分效果较好,但是并不是它的所有阶较大的立方子集的区分效果都很好,有些立方子集对正确和错误猜测值求和的结果相差很小,如果将此立方子集列入统计,则会对后面的排序结果造成干扰.

改进 1. 为了避免这样的干扰,我们不再选择大的立方集合,而是直接测试较小的立方集合对猜测值的区分能力,从中选取所需的区分能力强的立方集合,进而为下一步恢复秘密信息提供了攻击基础.

观察 2. 当需要猜测的表达式的个数较小时,通过对求和表排序得到正确猜测值的方法成功率较低,有时正确猜测值的排序甚至在表的后半部分中,此时以可接受的成功率得到正确秘密信息的计算复杂度将近似穷举攻击.研究发现:当选择合适的立方集合后,待猜测值组成的集合中存在着某些关键猜测值,在固定其他猜测值的情况下,关键猜测值取值正确时对应立方和为 0 的概率往往比关键猜测值错误时对应立方和为 0 的概率高,可据此恢复关键猜测值.

改进 2. 根据内部状态的表达式选定关键猜测值,离线阶段选定合适的立方集合;在线阶段,穷举猜测值进行立方求和后,得到一个由重量及相应猜测值构成的表,我们按照关键猜测值的取值将表进行分类(t 个关键猜测值则分为 2^t 类),将每类表中所有的重量相加,对应重量和最小的猜测值即为正确的关键猜测值.

以下我们对 5 轮简化版 MORUS 算法进行了改进的动态立方攻击,通过实验验证,改进的方法可以提高正确恢复关键猜测值的成功率.

2 MORUS 算法

MORUS 算法分为初始化、关联数据处理、加密、认证码生成和解密与验证这 5 个阶段.由于本文的攻击

条件将关联数据设置为 0,只涉及初始化阶段和加密阶段,下面只对算法的这两个阶段进行描述,详细的算法请参考设计报告.

2.1 符号说明

- S^i:第 i 步的内部状态,$0\leqslant i\leqslant 16$;
- S_k^i:第 i 步更新第 k 轮的内部状态,$0\leqslant k\leqslant 4$;
- $S_{k,l}^i$:第 i 步第 k 轮内部状态的第 l 块 128 比特分组,$0\leqslant l\leqslant 4$;
- $S_{k,l,j}^i$:第 i 步第 k 轮内部状态第 l 块分组的第 j 比特,$0\leqslant j\leqslant 127$;
- $Rotl(x,n)$:将 128 比特长的 x 分成 4 块 32 比特字,每块循环左移 n 比特;
- 0^n:n 长全 0 比特串;
- 1^n:n 长全 1 比特串;
- $(\cdot)_4$:16 进制表示;
- $const_0$:128 比特常数$(000101020305080d1522375990e97962)_4$;
- $const_1$:128 比特常数$(db3d18556dc22ff12011314273b528dd)_4$.

2.2 MORUS算法的状态更新函数

MORUS 算法的状态更新函数为 $S^{i+1}=StateUpdate(S^i,m_i)$,算法初始化过程共有 16 步这样的更新函数,每步更新函数包含 5 轮相似的更新过程,具体过程如下.

$$S_{1,0}^i = Rotl(S_{0,0}^i \oplus (S_{0,1}^i \,\&\, S_{0,2}^i) \oplus S_{0,3}^i, 5);$$
$$S_{1,3}^i = S_{0,3}^i \lll 32;$$

- 第 1 轮: $S_{1,1}^i = S_{0,1}^i$;
  $S_{1,2}^i = S_{0,2}^i$;
  $S_{1,4}^i = S_{0,4}^i$;

  $S_{2,1}^i = Rotl(S_{1,1}^i \oplus (S_{1,2}^i \,\&\, S_{1,3}^i) \oplus S_{1,4}^i \oplus m_i, 31)$;
  $S_{2,4}^i = S_{1,4}^i \lll 64$;
- 第 2 轮: $S_{2,0}^i = S_{1,0}^i$;
  $S_{2,2}^i = S_{1,2}^i$;
  $S_{2,3}^i = S_{1,3}^i$;

  $S_{3,2}^i = Rotl(S_{2,2}^i \oplus (S_{2,3}^i \,\&\, S_{2,4}^i) \oplus S_{2,0}^i \oplus m_i, 7)$;
  $S_{3,0}^i = S_{2,0}^i \lll 96$;
- 第 3 轮: $S_{3,1}^i = S_{2,1}^i$;
  $S_{3,3}^i = S_{2,3}^i$;
  $S_{3,4}^i = S_{2,4}^i$;

  $S_{4,3}^i = Rotl(S_{3,3}^i \oplus (S_{3,4}^i \,\&\, S_{3,0}^i) \oplus S_{3,1}^i \oplus m_i, 22)$;
  $S_{4,1}^i = S_{3,1}^i \lll 64$;
- 第 4 轮: $S_{4,0}^i = S_{3,0}^i$;
  $S_{4,2}^i = S_{3,2}^i$;
  $S_{4,4}^i = S_{3,4}^i$;

$$S_{0,4}^{i+1} = Rotl(S_{4,4}^{i} \oplus (S_{4,0}^{i} \,\&\, S_{4,1}^{i}) \oplus S_{4,2}^{i} \oplus m_i, 13);$$

$$S_{0,2}^{i+1} = S_{4,2}^{i} <<< 32;$$

- 第 5 轮: $S_{0,0}^{i+1} = S_{4,0}^{i}$;

  $S_{0,1}^{i+1} = S_{4,1}^{i}$;

  $S_{0,3}^{i+1} = S_{0,3}^{i}$.

- S^{i+1} 更新:For k=0 to 4, $S_{0,k}^{i+1} = S_{0,k}^{i}$.

状态更新函数如图 1 所示.

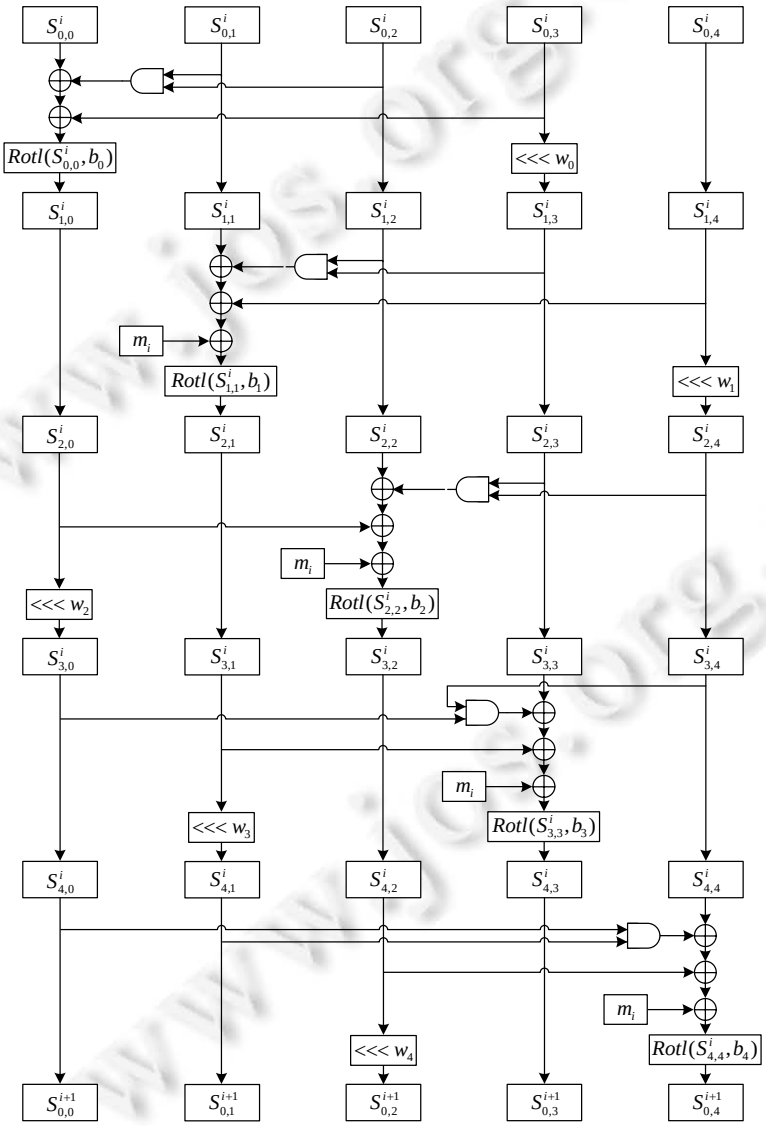

Fig.1 Update function of MORUS

图 1 MORUS 算法的状态更新函数

2.3 MORUS算法的初始化阶段

MORUS 的初始化阶段包括将密钥 K 和初始向量 IV 注入到内部状态中,并运行 16 步状态更新函数.密钥和

IV 的加载方式如下:

$$S_{0,0}^{-16}=\mathrm{IV};$$
$$S_{0,1}^{-16}=K;$$
$$S_{0,2}^{-16}=1^{128};$$
$$S_{0,3}^{-16}=const_0;$$
$$S_{0,4}^{-16}=const_1.$$

将密钥和 IV 注入后,进行 16 步更新:

For i=−16 to −1,

S^{i+1}=*StateUpdate*(S^i,0);

End

然后,密钥再次与第 2 个状态分组进行异或运算:

$$S_{0,1}^{-1}=S_{0,1}^{-1}\oplus K\ .$$

2.4 MORUS算法的加密阶段

在算法的加密阶段,128 比特的明文 P_i 与密钥流进行异或运算得到密文 C_i,设明文消息的长度为 *msglen*,令:

$$v=\lceil msglen/128\rceil.$$

For i=0 to v−1,

$C_i=P_i\oplus S_{0,0}^i\oplus(S_{0,1}^i<<<96)\oplus(S_{0,2}^i\ \&\ S_{0,3}^i)$;

S^{i+1}=*StateUpdate*(S^i,P_i)

3 对 5 步 MORUS 算法的改进动态立方攻击

动态立方攻击的过程可以分为离线阶段和在线阶段:离线阶段主要是收集必要的信息,为恢复秘密信息做准备;在线阶段的主要目的是恢复密钥.下面对简化版 MORUS 算法的动态立方攻击按照这两个阶段进行说明.

由于 MORUS 算法密钥流每次输出 128 比特,相当于 128 个关于密钥和 IV 的多项式,在进行立方测试时,可以分别利用每个比特的信息.下面我们以第 0 比特为例进行分析,其他比特的分析类似.

3.1 离线攻击阶段

- 步骤 1:寻找合适的表达式化简方法.

寻找动态 IV、需要穷举的秘密信息、关键猜测值和对应的立方集合中的必选 IV 集合,使得动态 IV 满足动态立方攻击条件时能够以合适的方式化简表达式.

输出密钥流的第 0 比特为:

$$z_0=P_0\oplus k_{97}\oplus S_{0,0,0}^{-1}\oplus S_{0,1,97}^{-1}\oplus S_{0,2,0}^{-1}\cdot S_{0,3,0}^{-1}=P_0\oplus k_{97}\oplus s_{-1,0}\oplus s_{-1,224}\oplus s_{-1,256}\cdot s_{-1,384}\tag{3}$$

为方便描述,我们令 $s_{-i,128\cdot k+l}=S_{0,k,l}^{-i}$.

公式(3)中对求和结果影响最大的是后面的二次项,需要对其化简,$s_{-1,256}$,$s_{-1,384}$ 的表达式分别为:

$$s_{-1,256}=s_{-2,149}\oplus s_{-2,438}\oplus s_{-2,533}\oplus s_{-2,277}\cdot s_{-2,437}\oplus s_{-2,598}\cdot(s_{-2,123}\oplus s_{-2,507}\oplus s_{-2,251}\cdot s_{-2,379})\tag{4}$$

$$s_{-1,384}=s_{-2,44}\oplus s_{-2,295}\oplus s_{-2,428}\oplus s_{-2,172}\cdot s_{-2,300}\oplus s_{-2,455}\cdot s_{-2,615}\tag{5}$$

我们采取令 $s_{-2,251}$=0 的方法对表达式(4)进行化简.

由于

$$s_{-2,251}=s_{-3,186}\oplus s_{-3,570}\oplus s_{-3,314}\cdot s_{-3,474}\tag{6}$$

故当 $s_{-3,314}$=0 且 $s_{-3,186}\oplus s_{-3,570}$=0 时,有 $s_{-2,251}$=0.

分别将 $s_{-3,314}$ 和 $s_{-3,186}\oplus s_{-3,570}$ 表示成关于密钥和 IV 的表达式.

将 $s_{-3,314}$ 和 $s_{-3,186}\oplus s_{-3,570}$ 简单表示成如下形式:

- $s_{-3,314}=v_{82}\oplus F_1=v_{82}\oplus v_{127}\cdot(v_{111}\oplus F_4)\oplus F_3$;
- $s_{-3,186}\oplus s_{-3,570}=v_{97}\oplus F_2$.

其中,F_1 和 F_2 是关于密钥和 IV 的多项式;而 v_{82} 只出现在 $s_{-3,314}$ 的一次项中,且在 $s_{-3,186}\oplus s_{-3,570}$ 中不出现;v_{97} 只出现在 $s_{-3,186}\oplus s_{-3,570}$ 的一次项中,且在 $s_{-3,314}$ 中不出现;v_{127} 和 v_{111} 均不包含在 F_3 和 F_4 中.

当 $v_{111}=F_4,v_{82}=F_3,v_{92}=F_2$ 时,$s_{-2,251}$ 恒为 0.根据观察 2 我们发现:若选取的立方集合能够以较大概率将 $s_{-2,251}$ 是否为 0 区分开,则能以较大概率判断出 $v_{111}=F_4$,进而根据 v_{111} 的取值恢复出 F_4.这样,我们令 v_{111} 为关键猜测值,F_4 为关键秘密信息,对应的 v_{127} 为必选 IV,同时将 v_{82} 和 v_{97} 列为动态变量.

需要说明的是:上述化简方式并不唯一,还可以找到其他的关键猜测值及其对应的秘密信息,为攻击者提供更多的信息.

- 步骤 2:寻找合适的立方集合.

此步需要根据步骤 1 中确定的表达式化简方法和必选 IV 集合,寻找满足区分条件的足够数量的立方集合.

在对初始化 5 步的 MORUS 算法进行立方测试的过程中我们发现:当立方集合大小为 9 时,输出比特的立方和存在明显的 0/1 不平衡性,而且可以保证在可接受的时间内找到足够数量符合条件的立方集合,我们将立方阶定为 9.

立方集合中包含必选 IV 和候选 IV,其中必选 IV 是由步骤 1 中关键秘密因素确定的.为了降低复杂度,候选 IV 则从 $s_{-3,314}$ 和 $s_{-3,186}\oplus s_{-3,570}$ 的表达式中均不包含的 IV 集合中选取,这样可使需要穷举的秘密信息尽可能少.经统计可知,$s_{-3,314}$ 和 $s_{-3,186}\oplus s_{-3,570}$ 中一共包含了 88 个 IV 比特,未包含 40 个 IV 比特的序号集合如下:

T={1,3,15,17,18,22,27,28,34,35,40,41,42,47,54,55,56,60,64,65,66,67,73,74,75,79,80,83,93,105,112,113,114,116,117,120,122,123,124,126}.

在立方测试时,为了检测 $s_{-2,251}$ 是否为 0 时立方测试时表现出的差别,在离线阶段则赋予攻击者额外的能力,可以在立方求和时强制将 $s_{-2,251}$ 设置为 0.

算法 2. 寻找 MORUS 算法的立方集合.

输入:立方阶 d,候选 IV 集合 A,必选 IV 集合 B,区分度 δ,立方集合目标个数 n;

输出:候选立方集合.

初始化:$C=\varnothing$

- Step 1. 从 A 中任选 $d-|B|$ 个 IV,与 B 组成立方集合 C';
- Step 2. 随机选取 m 组密钥,遍历立方集合 C',其他 IV 设置为 0,运行立方测试算法,在算法运行中强制将 $s_{-2,251}$ 设置为 0,对输出结果求和,结果为 sum_c;
- Step 3. 随机选取 m 组密钥,遍历立方集合 C',其他 IV 设置为 0,运行立方测试算法,对输出结果求和,结果为 sum'_c;
- Step 4. 如果 $sum'_c-sum_c>m\cdot\delta$,则令 $C=C\cup\{C'\}$;否则返回 Step 1,重新寻找立方集合;
- Step 5. 如果 $|C|=n$,或 Step 1 中已将 A 遍历,则输出 C 并终止算法;否则返回 Step 1,重新寻找立方集合.

如例 1,设置候选 IV 集合 $A=\{v_i|i\in T\}$,必选 IV 集合 $B=\{v_{127}\}$,n=400,m=1000,δ=0.05.运行算法 2,找到其中一个立方集合 $C'=\{v_3,v_{18},v_{27},v_{42},v_{65},v_{73},v_{114},v_{126},v_{127}\}$,随机选取 1 000 组密钥,分别执行 Step 2 和 Step 3 时得到:$sum'_c=309, sum_c=180$,差值 129>50,这个立方集合将具有较好的区分效果.

与算法 1 中的立方集合选取方式进行对比,我们的改进在于增加了区分度的判决筛选过程:$sum'_c-sum_c>1000\cdot\delta$.如果立方集合不满足此筛选条件,在线阶段使用此立方集合则会对恢复秘密信息造成干扰,降低成功率.

我们搜索了关于输出密钥流前 32 比特的立方集合及其对应的秘密信息(具体如表 1 所示).本文的实验环境为 Intel i5,3.2GHzCPU,4GB 内存,Windows7 32 位操作系统,对表中每个关键猜测值和必选 IV,搜索到足够数量的立方集合所需时间为几十分钟到几天不等.需要指出的是:对其他的 96 个密钥流输出比特进行考察将得到更多此类信息,能够进一步降低攻击复杂度.

Table 1 Dynamic cube attack key information of MORUS reduced to 5 rounds

表 1 5 步简化版 MORUS 算法动态立方的重要信息

输出比特	必选 IV	关键猜测值	成功率 p_s	可恢复密钥
z_0	v_{127}	v_{111}	90%	k_{11},k_{34}
z_0	v_{107}	v_{102}	98%*	k_{111}
z_1	v_{108}	v_{103}	98%*	k_{112}
z_2	v_{109}	v_{104}	96%*	$k_{87}\oplus k_{113}$
z_3	v_{110}	v_{105}	99%*	k_{114}
z_4	v_{111}	v_{106}	98%*	$k_{89}\oplus k_{115}$
z_5	v_{119}	v_{100}	98%*	k_{31},k_{47},k_{94}
z_6	v_{120}	v_{101}	97%*	k_0,k_{48},k_{95}
z_7	v_{114}	v_{109}	99%*	$k_{92}\oplus k_{118}$
z_8	v_{115}	v_{110}	96%*	$k_{93}\oplus k_{119}$
z_{10}	v_{117}	v_{112}	98%*	$k_{95}\oplus k_{121}$
z_{11}	v_{118}	v_{113}	97%*	$k_{64}\oplus k_{122}$
z_{14}	v_{121}	v_{116}	98%*	k_{44},k_{125}
z_{15}	v_{122}	v_{117}	96%*	$k_{68}\oplus k_{126}$
z_{16}	v_{123}	v_{118}	96%*	k_{127}
z_{18}	v_{125}	v_{120}	98%*	$k_{71}\oplus k_{97}$
z_{20}	v_{127}	v_{122}	99%*	$k_{73}\oplus k_{99}$
z_{21}	v_{96}	v_{123}	99%*	$k_{74}\oplus k_{100}$
z_{22}	v_{97}	v_{124}	97%*	k_{101}
z_{23}	v_{98}	v_{125}	99%*	$k_{76}\oplus k_{102}$
z_{25}	v_{100}	v_{127}	99%*	$k_{78}\oplus k_{104}$
z_{27}	v_{102}	v_{63}	96%*	k_{106}
z_{29}	v_{104}	v_{99}	99%*	k_{108}
z_{30}	v_{105}	v_{100}	96%*	k_{109}

*为概率不小于 96%的项

3.2 在线攻击阶段

- 步骤 3:恢复 MORUS 算法的秘密信息

本步骤是在线攻击阶段,此时密钥固定,主要目的是利用上一步中得到的秘密信息及其对应的立方集合,恢复其中的关键秘密信息,进而恢复部分(或全部)密钥.

算法 3. 恢复 MORUS 算法的关键秘密信息.

输入:秘密信息及其立方集合、关键秘密信息 c;

输出:c 的值.

初始化:令 sum_0=0,sum_1=0.

- Step 1. 将关键猜测值设置为 0,穷举剩下需要猜测的秘密信息,对每个猜测值,设置相应的动态 IV 取值,运行立方测试算法,将输出结果加到 sum_0 上;
- Step 2. 将关键猜测值设置为 1,穷举剩下需要猜测的秘密信息,对每个猜测值,设置相应的动态 IV 取值,运行立方测试算法,并将求和的结果加到 sum_1 上;
- Step 3. 若 $sum_0<sum_1$,则输出 c=0;否则输出 c=1.

利用算法 3 可以恢复关键秘密信息,关键秘密信息一般是一个非线性表达式,对于一些特殊的非线性表达式,我们可采取执行下述操作从中直接恢复某些 k_i 的信息:

当这个表达式中存在密钥比特 k_i 与 IV 比特 v_j 的乘积项 k_iv_j 时,且 v_j 在表达式其他部分中不出现时,可分别令 v_j=0 或 1,执行算法 3,并对两次求得的关键秘密信息做模 2 和,结果即为 k_i.

如例 1,利用算法 3 中的方法恢复关键秘密信息 F_4 时,随机选取了 100 组密钥,实验所得恢复的 F_4 与真实值相等的概率为 p_s=90%.由于 F_4 的表达式中含有项 $v_2(k_{11}\oplus 1)$,可以按照上述方法,通过改变 v_2 的值直接求出 k_{11} 的值.表 1 中列出了恢复关键秘密信息的成功率 p_s 和利用该秘密信息能够直接求出的密钥比特.为了对比,我们使用算法 1 中恢复秘密信息的方法,实验发现成功率约为 65%,远小于本文恢复关键秘密信息的成功率.

由表 1 可知,表中标*的项共 23 个,运行算法 3,可求出 23 个关键秘密信息.其中,能够直接求出 16 个密钥比特和 12 个关于密钥的线性方程,共 28 比特密钥信息.下面给出恢复所有密钥的方法.

算法 4. 恢复简化版 MORUS 算法所有密钥.

- Step 1. 对表 1 中 23 个关键秘密信息,运行算法 3,可求出 23 个关键秘密信息的值;
- Step 2. 对表 1 中 28 个密钥信息,改变相应的 IV 比特,运行算法 3,求出 28 个关键秘密信息的值;
- Step 3. 对于上述 51 个关键秘密信息,穷举其中错误值小于 5 的所有情况,对每一种情况,修正相应的关键猜测值,求出 16 个密钥比特和 12 个关于密钥的线性方程,并得到修正的 23 个关于密钥的方程;
- Step 4. 穷举剩余 112 个密钥比特:
  - Step 4-1. 代入 12 个关于密钥的线性方程和 23 个非线性方程进行验证:若出现矛盾,则返回 Step 3 中穷举下一个错误情况;否则,将此密钥代入 MORUS 算法进行进一步检验;
  - Step4-2. 若通过检验,则输出此密钥为正确密钥,算法终止;否则,返回 Step 3 中穷举下一个错误情况.

为保证攻击的成功率,算法 4 中采用了试错与穷举相结合的方法.Step 1 和 Step 2 的复杂度为$(23+28)\times 400\times 2^3\times 2^9\approx 2^{26.32}$,Step 3 中穷举所有错误情况的复杂度为 $C_{51}^4+C_{51}^3+C_{51}^2+C_{51}^1+C_{51}^0\approx 2^{18.05}$,Step 4 中密钥能够通过每个线性方程的概率为 1/2,而经验证,这 23 个非线性式子每个等于 0 的概率均为 1/2,且近似两两独立,可以认为密钥通过这 23 个非线性方程中每个的概率均为 1/2.因此,密钥通过第 1 层检验的概率为 $1/2^{35}$,故恢复所有 128 比特密钥的计算复杂度为 $O(2^{26.32}+2^{18.05}\times 2^{112-35})\approx O(2^{95.05})$.攻击的数据复杂度为 $O(2^{26.32})$.在计算成功率时,为简化计算,将恢复单个关键秘密信息的成功率 p_s 均假设为最低值 96%(实际的成功率应大于计算值),此时,51 个值中正确的个数 x 服从二项分布 $B(51,0.96)$,近似服从正态分布 $N(48.96,1.96)$,则 $p(x\geqslant 47)\approx 0.92$.因此,算法 4 恢复所有密钥的成功率大于 92%.表 2 给出了现有不同攻击方法的结果的比较.

Table 2 Results of attacks on MORUS

表 2 对 MORUS 算法的攻击结果对比

攻击方法	攻击轮数	计算复杂度	数据复杂度	文献
差分-区分	3	2	2	文献[3]
差分-区分	4	$O(2^{105})$	$O(2^{105})$	文献[3]
基于 SAT 的密钥恢复	16	$O(2^{370})$	-	文献[4]
密钥分割攻击(密钥恢复)	3	$O(2^{106.8})$	110	文献[3]
立方攻击	4	$O(2^{9.98})$	$O(2^{9.98})$	文献[5]
立方攻击(可分性)	5	$O(2^{125})$	$O(2^{23.59})$	文献[6]
动态立方攻击(密钥恢复)	5	$O(2^{95.5})$	$O(2^{26.32})$	本文

从攻击结果的对比可以发现:目前本文给出的改进的动态立方攻击是对 MORUS-640-128 算法轮数最多的恢复密钥攻击之一,并且本文的攻击相比文献[6]的结果在计算复杂度方面有较大优势.可见,动态立方攻击对此类算法是一种比较有效的攻击方法.另外,文献[4]中利用 SAT 求解器恢复内部状态攻击的计算复杂度大大高于穷举攻击,仅对该算法抵抗基于 SAT 的代数攻击能力给出了一个参考.

4 总 结

MORUS 算法的内部状态更新函数与传统的基于移位寄存器的序列密码有明显区别,基于移存器的序列密码每次只更新几个比特,而 MORUS 算法每次更新 128 比特,导致其内部状态各比特的代数表达式复杂程度接近,给动态立方攻击带来了一定的困难.本文改进了动态立方攻击方法,优化了立方子集合的选取规则,给出了新的秘密信息恢复方法.对初始化 5 轮的 MORUS 算法进行了动态立方攻击,最终以 $O(2^{95.05})$的复杂度恢复了所有 128 比特密钥.结果表明:动态立方攻击针对 MORUS 算法的攻击效果很好,并且有进一步提升的空间.下一步拟将改进的动态立方攻击应用于分析其他算法.

References:

[1] Wu HJ, Huang T. The authenticated cipher MORUS (v2). 2016. http://competitions.cr.yp.to/round3/morusv2.pdf

[2] Bernstein DJ. Caesar: Competition for authenticated encryption: Security, applicability, and robustness. In: Proc. of the CAESAR Web Page. 2014. http://competitions.cr.yp.to/index.html

[3] Zhang P, Guan J, Li JZ, Shi TR. Research on the confusion and diffusion properties of the initialization of MORUS . Journal of Cryptologic Research, 2015,2(6):536−548 (in Chinese with English abstract). [doi: 10.13868/j.cnki.jcr.000100]

[4] Dwivedi AD, Kloucek M, Morawiecki P, Kikolic I, Pieperzyk J, Wojtowicz S. SAT-Based cryptanalysis of authenticated ciphers from the CAESAR competition. In: Proc. of the IACR Cryptology ePrint Archive. 2016. 1053.

[5] Salam I, Simpson L, Bartlett H, Dawson E, Pieprzyk J, Wong KK. Investigating cube attacks on the authenticated encryption stream cipher MORUS. In: Proc. of the 2017 IEEE Trustcom/BigDataSE/ICESS. IEEE, 2017. 961−966. [doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.337]

[6] Li YB, Wang MQ. Cryptanalysis of MORUS. Designs, Codes and Cryptography, 2018. [doi: 10.1007/978-3-030-03329-3_2]

[7] Dinur I, Shamir A. Cube attacks on tweakable black box polynomials. In: Proc. of the Annual Int'l Conf. on the Theory and Applications of Cryptographic Techniques. Berlin, Heidelberg: Springer-Verlag, 2009. 278−299. [doi: 10.1007/978-3-642-01001-9_16]

[8] Mroczkowski P, Szmidt J. The cube attack on stream cipher Trivium and quadraticity tests. Fundamenta Informaticae, 2012, 114(3-4):309−318. [doi: 10.3233/FI-2012-631]

[9] Song HX, Fan XB, Wu CK, Feng DG. Cube attack on Grain. Ruan Jian Xue Bao/Journal of Software, 2012,23(1):171−176 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3983.htm [doi: 10.3724/SP.J.1001.2012.03983]

[10] Dinur I, Shamir A. Breaking Grain-128 with dynamic cube attacks. In: Proc. of the Int'l Workshop on Fast Software Encryption. Berlin, Heidelberg: Springer-Verlag, 2011. 167−187. [doi: 10.1007/978-3-642-21702-9_10]

[11] Huang SY, Wang XY, Xu GW, Wang MQ, Zhao JY. Conditional cube attack on reduced-round Keccak sponge function. In: Proc. of the Annual Int'l Conf. on the Theory and Applications of Cryptographic Techniques. Cham: Springer-Verlag, 2017. 259−288. [doi: 10.1007/978-3-319-56614-6_9]

[12] Ahmadian Z, Rasoolzadeh S, Salmasizadeh M, Aref M R. Automated dynamic cube attack on block ciphers: Cryptanalysis of SIMON and KATAN. In: Proc. of the IACR Cryptology ePrint Archive. 2015.

附中文参考文献:

[3] 张沛,关杰,李俊志,施泰荣.MORUS 算法初始化过程的混乱和扩散性研究.密码学报,2015,2(6):536−548. [doi: 10.13868/j.cnki.jcr.000100]

[9] 宋海欣,范修斌,武传坤,冯登国.流密码算法 Grain 的立方攻击.软件学报,2012,23(1):171−176. http://www.jos.org.cn/1000-9825/3983.htm [doi: 10.3724/SP.J.1001.2012.03983]

李俊志(1990－),男,河南新乡人,博士,主要研究领域为序列密码的设计与分析.

关杰(1974－),女,博士,教授,博士生导师,主要研究领域为信息安全,密码学.