高阶差分视角下的积分攻击

董 乐[1),2),3)] 吴文玲[1),2)] 吴 双[1)] 邹 剑[1),2),3)]

[1)](中国科学院信息工程研究所 北京 100093)

[2)](中国科学院软件研究所 北京 100190)

[3)](中国科学院研究生院 北京 100149)

摘 要 积分攻击和高阶差分攻击是分组密码的两种重要分析技术.尽管两者的理论基础并不相同,但是它们的攻击过程却十分相似.该文从高阶差分分析的视角来解释AES和Rijndael-256的积分区分器,证明高阶差分分析对此类算法同样有很强的分析能力.此外,改进了Rijndael-256的3轮区分器的数据复杂度.最后,给出了SPONGENT杂凑函数中间置换的14轮零和区分器.

关键词 积分攻击;高阶差分攻击;AES;Rijndael-256;SPONGENT

中图法分类号 TP309 **DOI号**:10.3724/SP.J.1016.2012.01906

Another Look at the Integral Attack by the Higher-Order Differential Attack

DONG Le[1),2),3)] WU Wen-Ling[1),2)] WU Shuang[1)] ZOU Jian[1),2),3)]

[1)](*Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093)

[2)](*Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

[3)](*Graduate University of Chinese Academy of Sciences, Beijing* 100149)

Abstract Integral attack and higher-order differential attack are two important techniques to analyze the block ciphers. There are different theoretical foundations for the two attacks, but they operate with similar procedures. In this paper, we expound the integral distinguishers of AES and Rijndael-256 in a higher-order differential cryptanalysis view, and we demonstrate the strong power of the higher-order differential cryptanalysis for the AES-like ciphers. Besides, the data complexity of the three-round distinguisher of Rijndael-256 is improved. We give a fourteen-round zero-sum distinguisher for the inner permutation of SPONGENT hash function at last.

Keywords integral attack; higher-order differential attack; AES; Rijndael-256; SPONGENT

1 引 言

自20世纪末,积分攻击[1-2]一直是分组密码的一种重要分析手段.它最早的分析对象是一个名为Square[3]的分组密码算法,所以开始被称为Square攻击,后来在对AES[1](Advanced Encryption Standard)的分析中扮演重要角色.2002年,Knudsen等人[2]将这一类分析方法命名为积分攻击,意为一个与差分攻击相对应的分析方法.这种分析方法,尤其适用于基于一些双射模块构造的算法.高阶差分攻击源于1994年Lai[4]提出的离散函数的高阶微分的概念.1995年,Knudsen系统地提出了高阶差分的概念,并将其应用于Feistel类密码[5].虽然这两种分析方法的理论基础不同,但是它们都是利用一组选择明文来构造区分器,使得在区分器的末端得到的密文的异或和为零(或者在某些比特为零),我们称为平衡状态或者局部平衡状态.更具体地说,都

收稿日期:2012-05-14;最终修改稿收到日期:2012-07-18.本课题得到国家重大专项基金(2011ZX03002-005-02)、中国科学院软件研究所创新基金领域前沿项目资助.**董 乐**,男,1980年生,博士研究生,讲师,主要研究方向为杂凑函数和分组密码的分析.E-mail:dongle127@163.com.**吴文玲**,女,1966年生,博士,研究员,主要研究领域为分组密码的设计与分析.**吴 双**,男,1983年生,博士,助理研究员,主要研究方向为杂凑函数的分析与设计.**邹 剑**,男,1985年生,博士研究生,主要研究方向为杂凑函数的分析与设计.

是首先找到一些字节或者比特，习惯地称为活跃字节或者活跃比特，取遍这些字节或者比特的所有能取的值，同时其它比特的值取定不变，来构造一个明文结构，使得这些明文所有对应的密文的异或和，或者某些比特的异或和为零.

积分攻击是对 AES 算法的最好的分析方法之一，并且对 AES 类算法的积分攻击是此攻击最具代表性的工作之一. 依靠对若干个明文字节的值的遍历(其它字节取定值)，在 3 轮和 4 轮核心积分区分器的基础上，对 AES 和 Rijndael-256 分别可以构造 4 轮和 6 轮的积分区分器[1,6]. 另一方面，如果想对算法低轮数的版本进行分析，可以以比特为单位选择活跃单元. 2008 年，Z'aba 等人[7]提出了比特积分的思想，首次将积分攻击用于分析 PRESENT 这种基于比特设计的分组密码. 2011 年，Wei 等人选择明文中 5 个字节的最低位比特作为活跃比特，以 2^5 的数据复杂度构造了 Rijndael-256 的 3 轮积分区分器[8]. 但是，对于此类区分器此数据复杂度是否最小还不明朗.

本文通过对 AES 类算法的积分攻击的分析，给出 AES 积分攻击中的经典的 3 轮区分器和 Rijndael-256 的积分攻击中经典的 4 轮区分器的高阶差分解释，并且确定了通过对活跃字节位置的适当选取，可以得到 Rijndael-256 的 4 轮，最后状态为全平衡的高阶差分区分器. 此外，我们对 Rijndael-256 的 3 轮低数据复杂度的积分区分器，给出了高阶差分解释，并且利用代数次数的增加规律进一步改进了数据复杂度. 最后，我们还对基于比特设计的杂凑函数 SPONGENT[9]其中一个版本的置换，进行了分析. 通过对其代数次数增加情况的计算，给出了这一置换的 14 轮零和区分器.

本文第 2 节介绍积分攻击与差分攻击等相关工作；第 3 节对本文涉及的算法 AES、Rijndael-256 和 SPONGENT 进行简要的介绍；第 4 节给出后面章节所用到的两个定理；我们在第 5 节用高阶差分解释 AES 和 Rijndael-256 的积分区分器；第 6 节解释并改进 Rijndael-256 的比特级积分区分器；第 7 节利用高阶差分构造 SPONGENT 内部置换的零和区分器；第 8 节总结并给出下一步的研究方向.

2　相关工作

2.1　积分攻击

积分攻击[1-2,6]源于对 AES 的前身 Square 分组密码算法的一种攻击. 攻击过程中，一些明文字节被选定为常数，记为 C，其它字节，一般称为活跃字节，取遍所有的 256 个值，这些字节记为 A. 如果在若干轮之后得到一个平衡状态，即异或和为零的状态，则我们说构造了一个积分区分器. 最后利用猜测密钥进行部分解密来恢复密钥.

2.2　高阶差分攻击

高阶差分攻击的理论基础为密码函数的高阶微分的定义及性质. 1994 年，Lai[4]给出了密码函数微分的定义.

定义 1.　设 P 为一个 F_2^n 上的置换. 对于任意的 $a\in F_2^n$，P 在点 a 处的微分为函数

$$D_aP(x)=P(x\oplus a)\oplus P(x).$$

而 P 在点 $a_1,a_2,\cdots,a_i$ 处的 i 阶微分定义为

$$D_{a_1,a_2,\cdots,a_i}P(x)=D_{a_i}(D_{a_{i-1}}(\cdots D_{a_1}P(x)))$$
$$=\bigoplus_{(k_1,k_2,\cdots,k_i)\in F_2^i}P(x\oplus\bigoplus_{j=1}^{i}k_ja_j).$$

一个函数的一阶微分的代数次数有性质 $\deg(D_aP(x))\leq\deg P(x)-1$. 所以，对于任意的 $(\deg P+1)$ 维的子空间 V，或者是任意的 $(\deg P+1)$ 个点 $a_1,a_2,\cdots,a_{\deg P+1}$，置换 P 的 $(\deg P+1)$ 阶微分为零.

高阶差分攻击就是利用这些性质，首先计算所攻击函数的代数次数的上界 d，然后选择 $d+1$ 个比特作为活跃比特，取定其它比特的值，遍历这些活跃比特的所有值，则可以在函数输出端得到平衡状态，即构造了一个高阶差分区分器. 2009 年，Aumasson 等人利用高阶差分的思想，构造了杂凑函数 KECCAK、Luffa 和 Hamsi 的“零和区分器”. 这种区分器从中间起始向两边构造高阶差分路径，最后在两端得到平衡状态.

本文中积分攻击和高阶差分攻击中的平衡状态指的都是“异或和为零”的状态；活跃字节和活跃比特都是指攻击的时候需要遍历的字节和比特.

3　算法简述

3.1　AES 和 Rijndael-256 算法简述

AES 分组密码算法是美国于 2001 年颁布的高级加密标准，分组长度为 128 比特，密钥长度分别为 128、192 和 256 比特，对应这 3 种长度的密钥，加密过程分别有 10、12 和 14 轮. AES 的状态基于字节可以写成一个 4×4 的矩阵

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}.$$

其中下标对应该字节所在的行和列的位置. 下文我们用 $a_{i,k}^{(j)}$ 来表示第 j 轮的输入状态中第 $i+1$ 行第 $k+1$ 列的字节,此外,我们用 $b_{i,k}^{(j)}$ 表示第 j 轮的 S 层输出状态中第 $i+1$ 行第 $k+1$ 列的字节.

AES 的轮函数有 4 个运算:

(1) 字节代换(SubBytes):状态中的每一个字节通过同一个 8×8 的 S 盒层,图中简记为 SB.

(2) 行移位(ShiftRows):第 i 行以字节为单位,向左循环移动 $i-1$ 个位置,图中简记为 SR.

(3) 列混淆(MixColumns):状态的每一列乘上一个 MDS(Maximum Distance Separable)矩阵,图中简记为 MC.

(4) 轮密钥加(AddRoundKey):中间状态和轮子密钥进行异或,图中简记为 AK.

此外,在第 1 轮加密之前,有一个轮密钥加层;最后一轮没有列混淆层.

Rijndael-256[10] 是 AES 的前身 Rijndael 分组密码算法的一个版本,它的分组长度为 256 比特,密钥长度也是 256 比特,轮数为 14 轮. 基于字节其状态可以记为一个 4×8 的矩阵

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} & a_{0,6} & a_{0,7} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} \end{pmatrix}.$$

Rijndael-256 的轮函数与 AES 类似,需要说明的是,行移位运算为,以字节为单位,第 1、2、3、4 行分别向左循环移动 0、1、3、4 个位置.

下文表示中间状态的时候,我们用矩形表示代替矩阵表示,字节的表示方法不变. 本文没有涉及 AES 和 Rijndael-256 的密钥生成算法,详细描述请参考文献[1,10].

3.2 SPONGENT-88/80/8 算法简述

SPONGENT 是在 CHES 2011 会议上提出的一个新的轻量级杂凑函数. 它采用了海绵结构,其中中间置换又应用了 PRESENT[11] 类型的置换. 海绵结构是一种由固定置换构建杂凑函数的新选择,它分为"吸收"和"榨取"两个部分. 吸收部分每次将 r 比特消息块异或在状态前 r 比特上,然后进入一个大小为 b 比特固定置换,直到处理完所有的消息. 接着便进入榨取部分,每次在将状态的前 r 比特输出之后,再进入同样的固定置换,直到输出比特的长度达到摘要的长度 n. SPONGENT 使用一种类似轻量级分组密码 PRESENT 的函数作为中间置换,此置换应用于 b 比特的中间状态. 轮函数包含 3 个运算:

(1) S 盒层(sBoxLayer):每 4 个比特进入同样的 4×4 的 S 盒.

(2) 比特置换层(pLayer):相当于 PRESENT 的比特置换的逆向扩展,它将状态的第 j 比特移动到第 $P_b(j)$ 比特,其中

$$P_b(j)=\begin{cases} j\times b/4 \bmod (b-1), & j\in\{0,\cdots,b-2\} \\ b-1, & j=b-1 \end{cases}.$$

(3) 轮常数加层(lCounter):状态的某些比特异或上轮常数.

SPONGENT-88/80/8 是 SPONGENT 杂凑函数的一个版本,它每次吸收或者榨取的比特数为 8,置换的大小为 80+8=88 比特,置换所含轮数为 45,摘要长度也是 88 比特. 有关此算法的更多信息请参考文献[9].

4 基础知识

为了更好地将积分攻击和高阶差分攻击联系起来,本节我们讨论一些常见函数的代数次数具备的性质. 我们记 n 元布尔函数 $f(x)$ 为 F_2^n 到 F_2 的一个函数,其中 x 为 n 元自变量 $x=(x_1,x_2,\cdots,x_n)$. 使得 $f(x)=1$ 的 x 的个数称为 $f(x)$ 的汉明重量,记为 $W_H(f)$. 特别地,如果 $W_H(f)=2^{n-1}$,我们称 $f(x)$ 为平衡布尔函数. 布尔函数有多种表示方式,其中常用的一种为多项式表示. 我们常常将变元升幂并且下标按字典序写出的多项式表达式

$$f(x)=a_0+a_1x_1+a_2x_2+\cdots+a_nx_n+a_{12}x_1x_2+\cdots+a_{n-1,n}x_{n-1}x_n+\cdots+a_{12\cdots n}x_1x_2\cdots x_n$$

称为 $f(x)$ 的代数正规型. 布尔函数 $f(x)$ 的代数次数定义为其代数正规型中具有非零系数的乘积项中的最大次数,而一个乘积项的次数定义为此项所含的变元的个数. 不影响理解的情况下,代数次数有时简称为次数. 我们称 F_2^n 到 F_2^m 上的函数 $F(x)$ 为向量布尔函数,其中 $x=(x_1,x_2,\cdots,x_n)\in F_2^n$,$F(x)=(f_1(x),f_2(x),\cdots,f_m(x))=(y_1,y_2,\cdots,y_m)\in F_2^m$,向量布尔函数的每一个分量 $f_i(x)$ 都是一个布尔函数. 若向量布尔函数像集中的每一个元素的原像个数都相同,则称其为平衡向量布尔函数. 首先我们给出一个已知的结论.

引理 1[12]. 设 $f(x)$ 为 F_2^n 到 F_2 的 n 元布尔函数,若它的汉明重量 $W_H(f)$ 为偶数,则其代数正规型无最高次项. 特别地,平衡布尔函数没有最高次项.

证明.　易知 $f(x)$代数正规型的最高次项系数

$$a_{12\cdots n}=\Big(\sum_{x\in F_2^n} f(x)\Big)\bmod 2 = W_H(f) \bmod 2,$$

这里求和为实数求和，则 $a_{12\cdots n}=0$ 当且仅当 $W_H(f)$ 为偶数，即最高次项不出现.　　证毕.

引理 1 的详细说明请参看文献[12]. 我们将此结果扩展至向量布尔函数的情况.

定理 1.　设 F 为 F_2^n 到 F_2^m 的平衡向量布尔函数，其中 $n\geq m$，则 F 的每个分量 $f_i(x)$ 的代数次数至多为 $n-1$.

证明.　由 F 为平衡向量布尔函数知，F 的像集 F_2^m 中的每一个元素均有 2^{n-m} 个原像. 考虑其中任一个分量 $f_i(x)$，F_2^m 中有一半的元素在此位置为 1，所以使得 $f_i(x)=1$ 的 x 的个数为 $2^{n-m}\times 2^{m-1}=2^{n-1}$，即 $f_i(x)$ 为平衡布尔函数. 由引理 1，其次数至多为 $n-1$.　　证毕.

利用定理 1 我们可以对一些向量布尔函数进行分析. 有时从一个函数的局部入手，反推至整个函数，可以得到更好的结果. 我们考虑 F 为 F_2^n 到 F_2^m 的向量布尔函数. 对其 n 元自变量 $x=(x_1,x_2,\cdots,x_n)$ 中的某 $n-t$ 个变元取常值，剩下的 t 个变元仍为变量，新的 t 元自变量记为 $\tilde{x}$. 并考察其像 $(f_1(\tilde{x}),f_2(\tilde{x}),\cdots,f_m(\tilde{x}))$ 中的 t 个分量，则每个分量都变成一个 t 元布尔函数，将它们仍按照原来的顺序写成向量的形式 $(f_1'(\tilde{x}),f_2'(\tilde{x}),\cdots,f_t'(\tilde{x}))$，便构成了一个 F_2^t 到 F_2^t 的向量布尔函数，我们称其为 F 的局部向量布尔函数，记为 F'. 我们从其局部的向量布尔函数 F' 入手进行分析，再进行变量扩充，可以得到下面的结果.

定理 2.　设 F 为 F_2^n 到 F_2^m 的向量布尔函数，如果对于选定的 $n-t$ 个变元取任意常值，并考察其像的特定的 t 个分量，所得的 F_2^t 到 F_2^t 的局部向量布尔函数 F' 均是平衡的，则

(1) 其每个分量的代数次数至多是 $t-1$.

(2) 将其它任意 $s-t$ 个常值比特重新取为变量，将其扩充为一个 F_2^s 到 F_2^t 的函数 F''，则扩充函数每个分量的代数次数至多为 $s-1$.

证明.　F' 为平衡向量布尔函数，由定理 1 可知其每个分量的代数次数至多为 $t-1$.

若其扩充的函数 F'' 中某个分量的代数次数达到 s，即其代数正规型中最高次 s 次项的系数为 1，将重新取为变量的 $s-t$ 个比特全部取 1，则可得一个新的对于选定的 $n-t$ 个变元取常值，并考察其像的特定的 t 个分量的 F_2^t 到 F_2^t 的局部向量布尔函数，并且此函数有一个分量存在最高次 s 次项，则其不是平衡的，与已知矛盾. 故扩充函数每个分量的代数次数至多为 $s-1$.　　证毕.

一般的分组密码和杂凑函数都是基于置换构造，并且都是相似置换的迭代组成. 但是，对于局部进行分析，并非都是置换，所以关键之处在于找到局部到局部的置换，或者是平衡函数. 根据这个定理 2，如果我们可以证明状态的局部(例如一个字节)到后面某状态的局部是一个平衡函数或者是双射的话，那么令“始状态”的这个局部的所有比特为变量，状态的其它比特为常数，“末状态”的这个局部的代数次数，一定小于“始状态”的这个局部的比特数.

5　用高阶差分解释已有的积分区分器

本节我们利用代数次数的变化规律解释一些经典的积分区分器，分别对 AES 的 3 轮积分区分器、Rijndael-256 的 4 轮积分区分器给出高阶差分解释.

5.1　解释 AES 的 3 轮积分区分器

Daemen 和 Rijmen 在提交他们设计的 AES 候选算法时，给出了所提交算法的 Square 攻击过程，此攻击也可看做是 AES 的积分攻击. AES 的 3 轮积分区分器如图 1 所示.

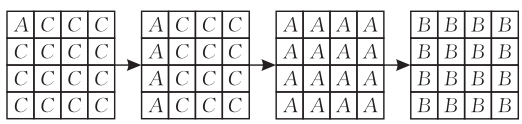


图 1　AES 的 3 轮积分区分器

在明文状态取遍一个字节的 256 个值，其它字节取常数，到第 3 轮结尾得到一个平衡状态，路径的推导细节请参看文献[10]. 而我们也可以从高阶差分的角度，利用对代数次数的分析，得到这个“0-和”路径，如图 2 所示，方框中数字代表此字节中所有比特的代数次数上界，空白方框代表每个比特代数次数均为零的字节.

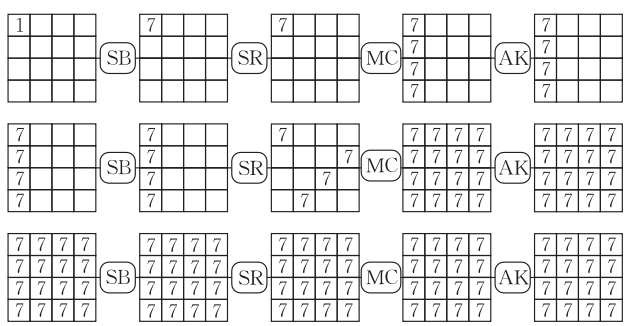


图 2　AES 代数次数增加情况

现在我们给出 AES 的 3 轮积分区分器的高阶差分路径：

(1)起始状态. 取 $a_{0,0}^{(1)}$ 的 8 个比特为变量，记为 $x_1,x_2,\cdots,x_8$，其它字节任取为常数. 起始状态中此字节的每个比特代数次数为 1，其它常数比特的代数次数为 0.

(2)第 1 轮. 经过第 1 轮 S 盒以后，$b_{0,0}^{(1)}$ 每个比特代数次数为 7，其它仍为 0. 经过行移位不变. 经过列混淆后第 1 列的 32 个比特的代数次数都变为 7. 加上轮密钥后保持不变.

(3)第 2 轮. $a_{0,0}^{(1)}$ 到第 2 轮 S 盒层输出状态中字节 $b_{0,0}^{(2)},b_{1,0}^{(2)},b_{2,0}^{(2)},b_{3,0}^{(2)}$ 都是置换. 我们以 $b_{0,0}^{(2)}$ 为例，若在 $a_{0,0}^{(1)}$ 处引入任何一个非零差分，则其可以扩散到第 1 轮输出状态第 1 列的每个字节，所以，$b_{0,0}^{(2)}$ 处的差分必然不为零，即 $a_{0,0}^{(1)}$ 到 $b_{0,0}^{(2)}$ 是一个单射. 而 $b_{0,0}^{(2)}$ 的每个取值，逆向计算都有一个 $a_{0,0}^{(1)}$ 的值与其对应，即 $a_{0,0}^{(1)}$ 到 $b_{0,0}^{(2)}$ 也是一个满射，所以为双射，即置换. 由定理 1 可知，第 2 轮 S 盒层输出状态第 1 列的每一个字节，所含比特的代数次数上界仍为 7. 经过后面的 3 个线性运算，可得第 2 轮输出状态的每一个比特的代数次数也都不高于 7.

(4)第 3 轮. 同样，我们可以用相似的证明过程证明 $a_{0,0}^{(1)}$ 到第 3 轮的 S 盒层输出状态的每一个字节也都是置换. 则此状态的每个比特的代数次数都不超过 7. 由此推得，第 3 轮结束时每个比特的代数次数都至多为 7.

这样，取定起始状态中 $a_{0,0}^{(1)}$ 之外字节的值，遍历 $a_{0,0}^{(1)}$ 所有值，会在第 3 轮结尾得到平衡状态.

5.2 解释 Rijndael-256 的 4 轮积分区分器

在 2008 年的非密会上，Galice 和 Minier 给出了 Rijndael-256 的 4 轮积分区分器[6]，如图 3 所示，利用此区分器，最多可以攻击 9 轮的 Rijndael-256 算法. Rijndael-256 的 4 轮积分区分器应用 3 阶积分，即在起始状态选择 3 个字节，取遍这 3 个字节的所有 2^{24} 个值，其它字节任取为定值. 可以推得第 4 轮的输出状态的第 3 列和第 7 列共 8 个字节是平衡的. 路径的推导细节请参看文献[6].

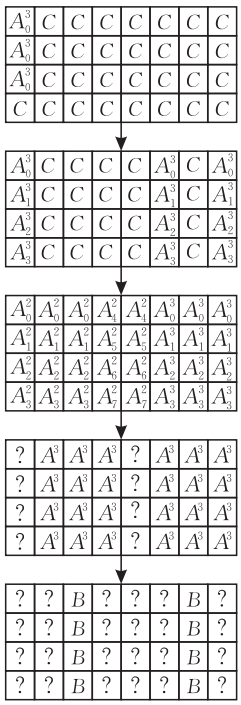


图 3 Rijndael-256 的 4 轮积分区分器

同样，我们可以用高阶差分的视角来解释这条路径. 我们首先对单字节 8 个比特变量代数次数增加情况进行分析，然后利用单字节代数次数的增加性质，推导出 3 个字节 24 个比特作为变量，在 4 轮结束时，状态中所有字节的代数次数的上界. 图 4 为选择字节 $a_{0,0}^{(1)}$ 中 8 个比特为变量，代数次数的增加情况. 方框中数字代表此字节中所有比特的代数次

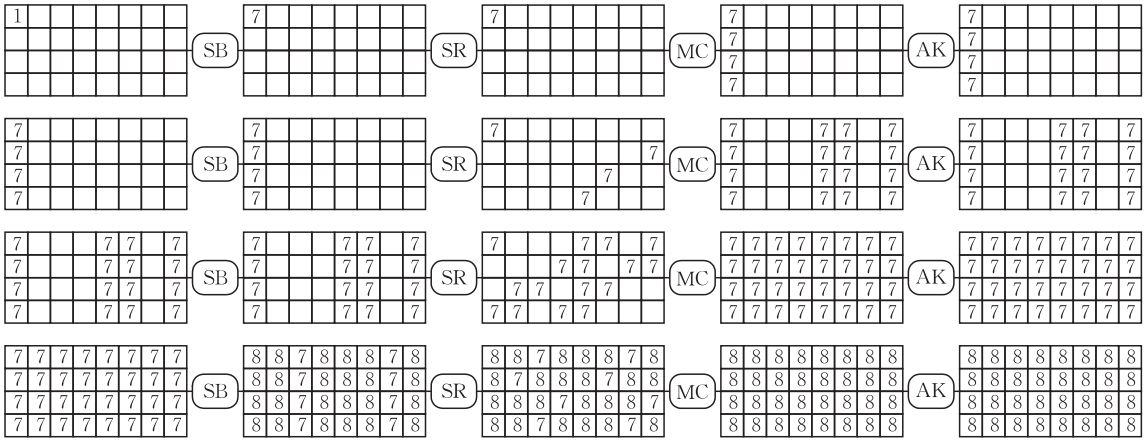


图 4 Rijndael-256 的 4 轮单字节起始代数次数增加情况

数上界，空白方框代表每个比特代数次数均为零的字节.

首先我们给出单字节 8 个比特变量的 4 轮高阶差分路径：

(1) 起始状态. 取 $a_{0,0}^{(1)}$ 的 8 个比特为变量，其它字节任取为常数. 起始状态中此字节的每个比特代数次数为 1，其它常数比特的代数次数为 0.

(2) 第 1 轮和第 2 轮. 前两轮的分析和 AES 类似. $a_{0,0}^{(1)}$ 的 8 个比特变量在第 2 轮结束时，使得状态的第 1、5、6、8 列比特的代数次数上界为 7，而其它字节的所有比特均为常值.

(3) 第 3 轮. 类似地，我们可以推知 $a_{0,0}^{(1)}$ 到第 3 轮 S 盒层的输出状态的第 1、5、6、8 列的每个字节都是置换. 由定理 1，这些字节所有比特的代数次数都不超过 7. 我们注意到，在第 3 轮的列混淆层输入中，第 3 列和第 7 列都只有一个字节含有变量，所以 $a_{0,0}^{(1)}$ 到第 3 轮输出状态的第 3 列和第 7 列中每个字节都是置换，这个性质可以延伸到第 4 轮的 S 盒层之后.

(4) 第 4 轮. 由上面第 3 轮的推导可以得出，$a_{0,0}^{(1)}$ 到第 4 轮中 S 盒层输出状态的第 3 列和第 7 列的每个字节都是置换，所以这 8 个字节的所有比特的代数次数都不超过 7，而其余的字节所含比特没有此性质，但是由于只有 8 个变量，所以其余字节所含比特的代数次数上界为 8. 由图 4 我们可以看到，从一个活跃字节出发，无法得出第 4 轮输出状态的某些字节平衡，但是我们得到第 4 轮的 S 盒层之后，有两列所含比特的代数次数都没有达到最大的 8 次.

由上面给出的单字节 4 轮高阶差分路径，如果我们选起始状态中 $a_{0,0}^{(1)}$ 为活跃字节，则可以得到一个 3.25 轮的区分器. 利用此性质我们来构造 3 个字节 24 个比特变量的 4 轮高阶差分路径：

(1) 与上述单字节路径类似，如果在起始状态中选择 $a_{1,0}^{(1)}$ 或者 $a_{2,0}^{(1)}$ 的 8 个比特为变量，则会在第 4 轮 S 盒层之后得到第 4 列和第 8 列，或者第 2 列和第 6 列所有比特的代数次数达不到最大的 8 次.

(2) 如果我们同时选择这 3 个字节所含的 24 个比特为变量，对于图 4 所示的单字节起始路径来说，相当于将 $a_{1,0}^{(1)}$，$a_{2,0}^{(1)}$ 中的 16 个常数比特重新变为了变量. 由定理 2 可得，第 4 轮中 S 盒层输出状态的第 3 列和第 7 列，所有比特的代数次数都不会超过 23. 分别由 $a_{1,0}^{(1)}$ 和 $a_{2,0}^{(1)}$ 对应的单字节路径出发，我们可以用相似的过程得到此状态的第 2、4、6、8 列所有比特的代数次数也至多为 23.

(3) 容易知道，经过第 4 轮的行移位、列混淆和轮密钥加运算，可得第 4 轮的输出状态的第 3 列和第 7 列的所有比特的代数次数都不超过 23. 如图 5 所示.

这样，取定起始状态中 $a_{0,0}^{(1)}, a_{1,0}^{(1)}, a_{2,0}^{(1)}$ 之外字节的值，并遍历这 3 个字节的所有值，会在第 4 轮结尾得到平衡状态.

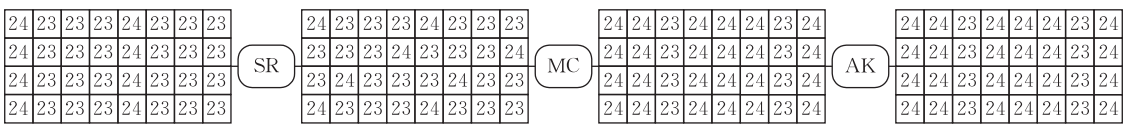


图 5　Rijndael-256 的 4 轮区分器的最后 3 个运算

根据上述结论，我们可以同时选择 4 个字节，使得第 4 轮输出状态全部比特的代数次数都达不到最大的 32. 这样我们可以以 2^{32} 的复杂度得到 4 轮的“完全”平衡状态，而不是只有两列平衡. 事实上，我们选择 $a_{0,0}^{(1)}, a_{1,0}^{(1)}, a_{2,0}^{(1)}, a_{2,1}^{(1)}$ 这 4 个字节作为活跃字节即可.

6　用高阶差分改进积分区分器

2011 年，Wei 等人利用比特级积分改进了 Rijndael-256 基于字节的积分区分器[8]，将其 3 轮积分区分器的数据复杂度从 2^8 降至 2^5. 此区分器首先选择 $a_{0,0}^{(1)}, a_{1,0}^{(1)}, a_{2,0}^{(1)}, a_{3,0}^{(1)}, a_{0,1}^{(1)}$ 的最低位比特作为活跃比特，取定其它比特的值，遍历这 5 个比特的所有 32 个值，则可以在 3 轮之后得到一个全平衡状态.

6.1　解释 Rijndael-256 的 3 轮比特积分区分器

我们可以在高阶差分的视角下，利用代数次数的增加情况来解释这条路径. 如图 6 所示，设字节 $a_{0,0}^{(1)}, a_{1,0}^{(1)}, a_{2,0}^{(1)}, a_{3,0}^{(1)}, a_{0,1}^{(1)}$ 的最低位比特分别为 x_1, x_2, x_3, x_4, x_5，它们在区分器中是要遍历的活跃比特，所以在研究代数次数的时候以变量形式出现. 图中数字代表该字节所有比特代数次数上界，下标代表该字节所有比特的代数正规型中至多包含的变量. 例如 1_{145} 表示该字节的所有比特的代数次数至多为 1，并且代数正规型中至多可能出现 x_1, x_4, x_5，换句话说，一定不会出现 x_2, x_3. 此外，为了记法的方便，我们采用带负号的下标，例如 1_{-5} 表示，代数正规型中一定不会出现 x_5.

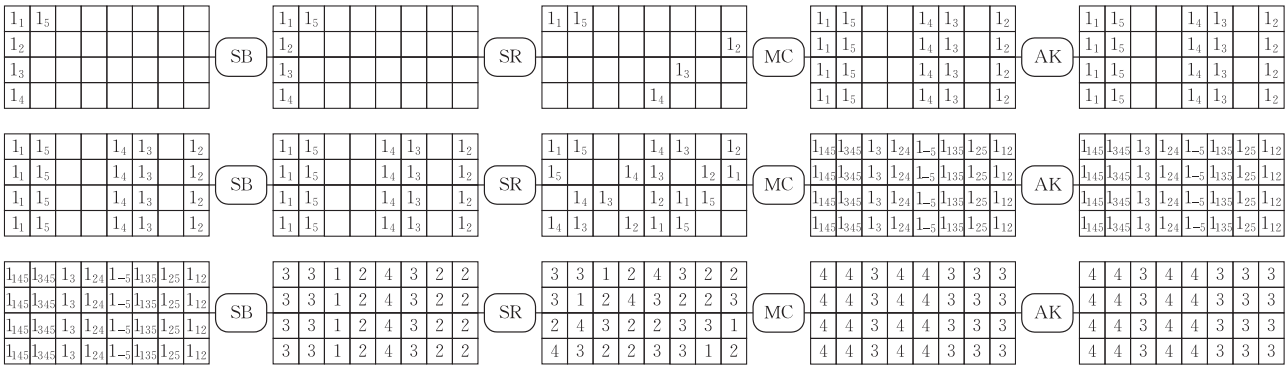


图 6 Rijndael-256 的 3 轮 5 阶高阶差分路径

我们下面给出 Rijndael-256 的 3 轮比特起始区分器的代数次数增加情况说明：

(1)第 1 轮. 起始状态中含有变量的字节都只有一个变量，这样第 1 轮的 S 盒实际上可以看作是一个线性运算，也就是说，在第 1 轮 S 盒的输出状态中，这 5 个字节的所有比特的代数次数至多是 1. 经过后面的几个线性扩散运算，在第 1 轮的末端，有 5 列都含有变量，但是显然代数次数仍然至多为 1，并且含变量的字节仍然只含有一个变量.

(2)第 2 轮. 由上面的分析，第 2 轮的输入状态中含有变量的字节都只有一个变量参与运算，这样其输出状态中的代数次数仍然至多为 1. 经过后面的线性扩散运算，会使得状态的每个比特都可能含有变量，代数次数的上界仍然为 1，但是，有些比特的代数正规型中，可能会含有多个变量，例如第 1 列中的某些比特可能会同时含有 x_1, x_4, x_5.

(3)第 3 轮. 我们知道，含有多个变量的字节，经过 S 盒层之后，代数次数一定不可能大于其所含变量的个数. 而第 3 轮的输入中，每个字节至多可能含有 4 个变量，这样在第 3 轮的 S 盒层的输出状态中，所有比特的代数次数不可能大于 4，而后面都是线性运算，所以，在第 3 轮的末端，状态中所有比特的代数次数都不会超过 4 次.

这样，在其它比特取定值，$a_{0,0}^{(1)}, a_{1,0}^{(1)}, a_{2,0}^{(1)}, a_{3,0}^{(1)}, a_{0,1}^{(1)}$ 的最低位比特取遍所有 32 个值，可以在 3 轮之后得到平衡状态.

6.2 改进 Rijndael-256 的 3 轮比特积分区分器

我们利用代数次数的增加，解释了 Rijndael-256 的 3 轮比特积分区分器. 但是事实上，此区分器并没有将数据复杂度降低到最小，如果利用代数次数进行分析，可以找到涉及更少明文的“零和”区分器. 图 7 为 Rijndael-256 的 3 轮 3 阶高阶差分区分器，图中符号代表的意义与图 6 相同.

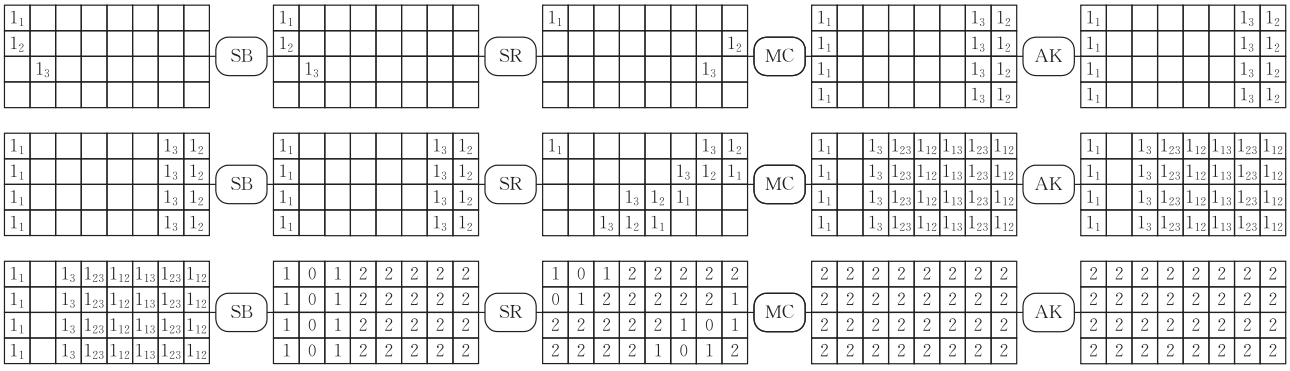


图 7 Rijndael-256 的 3 轮 3 阶高阶差分路径

我们下面给出 Rijndael-256 的 3 轮 3 阶高阶差分路径的说明：

(1)起始状态. 我们在 $a_{0,0}^{(1)}, a_{1,0}^{(1)}, a_{2,1}^{(1)}$ 这 3 个字节中分别任取一个比特作为活跃比特，记为 x_1, x_2, x_3，其它比特取为定值.

(2)第 1 轮. 起始状态中含有变量的字节都只有一个变量，所以在第 1 轮 S 盒的输出状态中，这 3 个字节的所有比特的代数次数至多是 1. 经过后面的几个线性扩散运算，在第 1 轮的末端有 3 列含变量，而且代数次数仍然至多为 1，含变量的字节仍然只含有一个变量.

(3)第 2 轮. 第 2 轮的输入状态中含有变量的

字节同样只有一个变量，这样其输出状态中相关比特的代数次数仍然至多为 1. 经过后面的线性扩散运算，除第 2 列外，状态的每个比特都可能含有变量，其中第 1 列和第 3 列比特至多含有一个变量，其余 5 列至多含有两个变量，代数次数的上界仍然为 1.

(4) 第 3 轮. 由上一轮的分析，我们得到所有的字节所含的变量数都不大于 2，这样在第 3 轮的 S 盒层的输出状态中，所有比特的代数次数不可能大于 2，而后面都是线性运算，所以，在第 3 轮的末端，状态中所有比特的代数次数都不会超过 2 次.

这样，在其它比特取定值，选定的 3 个比特取遍所有 8 个值，可以在 3 轮之后得到平衡状态.

如果我们放弃最后状态所有比特都要平衡的要求，我们还可以以更少的明文个数得到 3 轮部分平衡的高阶差分区分器，如图 8 所示.

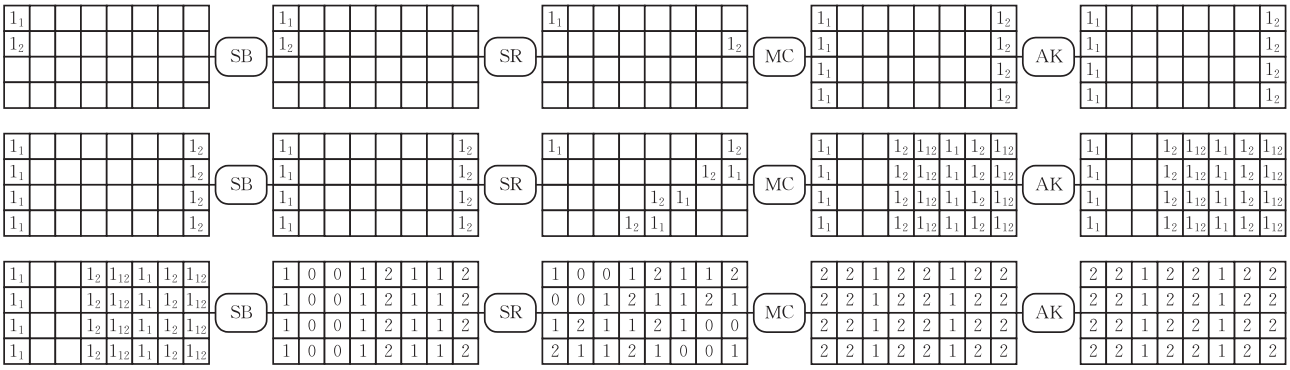


图 8　Rijndael-256 的 3 轮 2 阶高阶差分路径

我们在 $a_{0,0}^{(1)}, a_{1,0}^{(1)}$ 两个字节中分别任取一个比特作为活跃比特，记为 x_1, x_2，其它比特取为定值. 则可以在第 1 轮后得到代数次数至多为 1，且每个字节至多含一个变量的两列，其它列为常数. 同样，第 2 轮之后每个比特的代数次数都不超过 1，并且我们得到两个常数列，此外有第 1、4、6、7 列中每个字节只含有 1 个变量，只有第 5 列和第 8 列中字节可能含有两个变量. 这样在第 3 轮的 S 盒层之后除第 5 列和第 8 列外的所有比特的代数次数都不超过 1. 再经过行移位、列混淆和轮常数加运算，我们发现第 3 列和第 6 列中所有比特代数次数都不超过 1. 这样，在其它比特取定值，选定的两个比特取遍所有 4 个值，可以在 3 轮之后得到局部平衡状态.

7　构造 SPONGENT 的零和区分器

2009 年，Aumasson 等人利用高阶差分的思想，提出了“零和区分器”的概念. 本节我们利用 SPONGENT 函数代数次数的增加情况，对其最轻量级版本 SPONGENT-88/80/8 的内部置换构造零和区分器.

SPONGENT 的内部置换采用的是 PRESENT 类的置换类型，对于高阶差分分析来说，其 S 盒层不但提供代数次数的增加，而且提供各个比特变量的混淆. 但是，由于其轮函数中比特置换层的对称性，加上其 4 比特 S 盒中有一个比特的代数次数没有达到最高的 3 次，导致其轮函数迭代对应的代数次数增加比期望的要慢. 我们便由其 S 盒中每个比特的代数次数出发，来研究轮函数的代数次数增加情况，进而构造零和区分器.

SPONGENT 采用的 4×4 的 S 盒可以记为 $S[16]=\{E,D,B,0,2,1,4,F,7,A,8,5,9,C,3,6\}$. 如果将其 4 比特输入，由高位向低位依次记为 a_1, a_2, a_3, a_4，4 比特输出由高位向低位依次记为 b_1, b_2, b_3, b_4，则其代数正规型可以写为

$$b_1=a_1a_2a_4+a_1a_3a_4+a_1a_3+a_1a_4+a_3a_4+a_1+a_2+1,$$
$$b_2=a_1a_2a_3+a_1a_4+a_2+a_3+1,$$
$$b_3=a_1a_2a_3+a_1a_2+a_1a_3+a_1a_4+a_2a_3+a_4+1,$$
$$b_4=a_2a_3+a_1+a_3+a_4.$$

这样，即使 S 盒的 4 个输入比特都取为变量，其输出的 4 个比特中也会有一个为 2 次，其它的为 3 次. 此外，我们还将用到此 S 盒的逆的代数正规型. 将 S 盒的逆的 4 比特输入，由高位向低位依次记为 a_1, a_2, a_3, a_4，4 比特输出由高位向低位依次记为 b_1, b_2, b_3, b_4，则其代数正规型可以写为

$$b_1=a_2a_3+a_2a_4+a_3a_4+a_1,$$
$$b_2=a_2a_3a_4+a_1a_3+a_2a_3+a_3a_4+a_2+a_3+a_4,$$
$$b_3=a_2a_3a_4+a_1a_2+a_2a_3+a_2a_4+a_3+a_4+1,$$
$$b_4=a_1a_2a_4+a_1a_3a_4+a_2a_4+a_1+a_2+a_3+1.$$

其输出的 4 个比特中也会有一个为 2 次，其它的为 3 次. 我们根据此 S 盒代数次数的性质，加上比特置

换层的对称性，给出 SPONGENT-88/80/8 函数置换的 14 轮零和区分器.

SPONGENT-88/80/8 的置换函数状态大小为 88 比特，采用 4×4 的 S 盒，所以每轮含有 22 个并行的 S 盒作为 S 盒层，我们从左至右记为第 1，2，…，22 个 S 盒. S 盒层的输出通过一个比特置换层交换比特的位置. 而轮常数加运算因为不会给代数次数上界带来影响，在我们的分析中将其忽略. 我们所给的 14 轮零和区分器包含两个部分，8 轮的正向部分，我们记为第 7～14 轮；6 轮的逆向部分，我们记为第 1～6 轮. 而起始的中间状态，为第 6 轮的输出状态，选择了 21 个 S 盒的 84 个比特作为活跃比特.

7.1 区分器的第 9～14 轮高阶差分路径

首先我们给出正向部分中第 9～14 轮的高阶差分路径.

(1) 起始状态. 第 9 轮的输入状态有 4 个 S 盒作为活跃 S 盒，即包含 16 个活跃比特，这 4 个活跃 S 盒分别为第 1，8，15，19 个 S 盒. 由于轮函数中的 S 盒是一个置换，所以第 9 轮 S 盒层中上述 4 个 S 盒输出的 16 个比特仍然是活跃且独立的. 我们从第 9 轮 S 盒层的输出状态出发，将活跃的 16 个比特记为变量 $x_1, x_2, \cdots, x_{16}$，其它比特为常数，计算第 9～14 轮的每个比特的代数次数的上界.

(2) 第 9 轮. 16 个活跃比特通过比特置换层后，位置发生了变化，使得第 10 轮的 S 盒输入中，至多有一个变量进入同一个 S 盒，还有 6 个 S 盒没有变量进入. 表 1 的第 2 行，给出了第 9 轮输出状态中每个比特的代数次数的上界，同一个小括号中的数字，代表要进入下一轮同一个 S 盒的 4 个比特的代数次数的上界.

(3) 第 10 轮. 我们知道，一个变量比特进入 S 盒之后，S 盒的所有输出比特都可能和这个变量相关，则它们的代数次数都可能会从 0 变为 1. 所以，16 个变量通过第 10 轮的 S 盒层后，会使得输出状态中，与它们相关的 64 个比特的代数次数上界变为 1. 但是仍然有 22－16＝6 个 S 盒中没有变量参与运算，即 S 盒层输出比特中有 24 个，代数次数仍为 0. 经过比特置换交换位置，此轮输出状态的所有比特代数次数上界见表 1 第 3 行.

(4) 第 11 轮. 从第 11 轮开始，代数次数的计算开始利用 S 盒的代数正规型，我们由 S 盒 4 个输入比特的代数次数，来估计可能产生的最高次项的次数. 以第 11 轮的第 1 个 S 盒为例，由第 10 轮的输出我们知道，此 S 盒的 4 个输入比特代数次数上界为 (1,1,0,1)，又其输出的第 1 个比特含有 3 次项 $a_1a_2a_4$，由输入比特的代数次数上界知道 a_1, a_2, a_4 都可能含有变量，所以这一比特的代数次数上界为 3. 此 S 盒的第 2 个输出比特中含有 3 次项 $a_1a_2a_3$，而 a_3 代数次数为 0，即为常值不含变量，所以此 3 次项的代数次数至多为 2，则这个比特的代数次数上界为 2. 同样可以推知，第 3 个和第 4 个输出比特的代数次数上界分别为 2 和 1. 利用这一方法，可以推得第 11 轮 S 盒层输出的所有比特代数次数的上界，再由比特换位运算得出第 11 轮输出状态的所有比特代数次数上界，见表 1 第 4 行.

表 1 第 9～14 轮输出比特代数次数上界

轮数	每一比特代数次数上界
9	(1,0,0,0),(0,0,0,1),(0,0,0,0),(0,0,1,0),(0,0,1,0), (0,0,1,0),(0,0,0,0),(0,1,0,0),(0,0,0,0),(1,0,0,0), (1,0,0,0),(1,0,0,0),(0,0,0,1),(0,0,0,0),(0,0,1,0), (0,0,1,0),(0,0,1,0),(0,0,0,0),(0,1,0,0),(0,0,0,0), (1,0,0,0),(1,0,0,0)
10	(1,1,0,1),(1,1,0,1),(0,1,1,1),(1,0,1,1),(1,0,1,0), (1,1,1,1),(0,1,1,1),(0,1,0,1),(1,1,1,0),(1,1,1,0), (1,0,1,1),(1,1,0,1),(1,1,0,1),(0,1,1,1),(1,0,1,1), (1,0,1,0),(1,1,1,1),(0,1,1,1),(0,1,0,1),(1,1,1,0), (1,1,1,0),(1,0,1,1)
11	(3,3,2,3),(2,3,2,2),(2,2,3,3),(3,2,3,2),(3,2,2,2), (2,3,2,2),(2,2,2,3),(2,1,3,3),(2,2,2,2),(2,2,3,2), (1,3,3,2),(2,2,2,2),(2,3,2,1),(3,3,2,2),(2,2,2,2), (3,2,1,3),(3,2,1,1),(2,1,1,2),(2,1,2,2),(1,1,1,2), (1,1,2,2),(1,2,2,1)
12	(9,7,8,8),(7,7,7,8),(6,7,6,6),(6,8,6,8),(6,5,6,4), (5,4,8,7),(7,8,7,7),(6,6,6,7),(7,6,7,8),(6,6,6,4), (5,3,4,5),(8,7,7,8),(7,7,6,6),(6,7,7,6),(7,8,6,6), (6,4,5,3),(4,5,5,5),(5,5,4,5),(4,4,4,5),(6,4,5,5), (4,3,3,2),(3,2,3,4)
13	(16,16,16,16),(16,16,16,16),(16,16,14,16), (16,16,16,14),(14,15,13,16),(9,10,16,16), (16,16,16,16),(16,16,16,16),(12,16,16,16), (16,15,14,14),(12,15,10,8),(16,16,16,16), (16,16,16,16),(16,16,12,16),(16,16,16,15), (14,14,12,15),(10,8,15,14),(13,14,11,12), (15,12,13,12),(7,14,13,14),(14,9,10,9),(8,9,6,5)
14	(16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,16),(16,16,16,16),(16,16,16,16), (16,16,16,15)

(5) 第 12 轮和第 13 轮. 按照同样的方法，我们可以向后再推两轮，在第 13 轮的输出状态中，得到所有比特的代数次数上界，值得注意的是进入第 14 轮的最后一个 S 盒的 4 个比特的代数次数上界为 (8,9,6,5).

(6) 第 14 轮. 由于共有 16 个变量参与运算，故

所有比特的代数次数至多为 16. 但是，我们发现此轮的最后一个 S 盒的最后一个比特的代数次数至多为 15. 这是因为，最后一个比特的 4 个项 a_2a_3、a_1、a_3、a_4 的代数次数上界分别为 15、8、6、5，所以此比特代数次数不超过 15.

这样，在第 9 轮的 S 盒层输出状态中，选定的 16 个活跃比特取遍所有 2^{16} 个值，其它比特取定值，可以在第 14 轮之后使得状态的最后一个比特平衡.

7.2　区分器的第 1～4 轮高阶差分路径

下面我们给出逆向部分第 1～4 轮的高阶差分路径，我们按照代数次数增加的顺序，将第 4 轮的输出状态定为起始状态，逆向至第 1 轮进行代数次数上界的计算.

(1) 起始状态. 第 4 轮的输出状态的前 16 个比特作为活跃比特. 我们从此状态出发，将活跃的 16 个比特记为变量 $x_1, x_2, \cdots, x_{16}$，其它比特为常数，计算第 4～1 轮的每个比特的代数次数的上界.

(2) 第 4 轮. 由于第 4 轮的输出状态中前 16 个比特为变量，逆向计算先通过比特置换运算，16 个变量进入 16 个逆 S 盒，分别为它们的最高位比特，最后的 6 个 S 盒没有变量进入. 在逆 S 盒的输出状态(正向看为第 4 轮的输入状态)中前 16 个有变量进入的逆 S 盒的所有 64 个比特位置都有可能含有变量，但是由于每个逆 S 盒只有一个变量进入，所以此状态中前 64 个比特的代数次数上界为 1；由于后 6 个逆 S 盒没有变量进入，所以此状态中后 24 个比特的代数次数为 0. 见表 2 第 2 行.

(3) 第 3 轮. 第 4 轮的输入状态逆向计算通过比特置换层后，使得第 3 轮逆 S 盒层的输入状态中，前 20 个 S 盒的输入代数次数模式都是(1,1,1,0)，即前 3 个输入比特的代数次数上界都是 1，最后一个比特的代数次数上界为 0. 它们的输出比特的代数次数上界需要根据 S 盒的逆的代数正规型来计算. 首先计算逆 S 盒的第 1 个输出比特，根据代数正规型，此比特的最高次项为 2 次，并且里面有一项为 a_2a_3，而其输入中 a_2，a_3 两个比特的代数次数都可能为 1，所以第 1 个输出比特的代数次数上界为 2. 第 2 个和第 3 个输出比特中都只有 3 次项 $a_2a_3a_4$，而我们知道 a_4 的代数次数为 0，所以这两个比特的代数次数上界也为 2. 同样，第 4 个比特的两个 3 次项中都含有 a_4，则它的代数次数上界也为 2. 所以，逆 S 盒的输入代数次数模式(1,1,1,0)的输出代数次数模式为(2,2,2,2). 而第 3 轮逆 S 盒层的输入状态中，后 2 个 S 盒的输入代数次数模式为(1,1,0,0)，可以计算它的输出代数次数模式为(1,1,2,2). 见表 2 第 3 行.

(4) 第 2 轮和第 1 轮. 按照同样的方法，我们可以向前再推两轮，在第 1 轮的输入状态中，得到所有 88 个比特的代数次数上界，其中的 40 个比特的代数次数不能达到最大的 16.

表 2　第 4～1 轮输入比特代数次数上界

轮数	每一比特代数次数上界
4	(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(1,1,1,1),(0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0)
3	(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(2,2,2,2),(1,1,2,2),(1,1,2,2)
2	(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,6,6,6),(4,5,5,5),(4,5,5,5),(4,6,6,6),(4,6,6,6),(4,5,5,5),(4,5,5,5),(4,6,6,6),(4,6,6,6)
1	(12,16,16,16),(12,16,16,16),(10,14,14,16),(12,16,16,16),(12,16,16,16),(12,16,16,16),(10,14,14,16),(12,16,16,16),(11,15,15,15),(12,16,16,16),(10,14,14,16),(12,16,16,16),(11,15,15,15),(11,16,16,16),(9,13,13,15),(12,16,16,16),(12,16,16,16),(12,16,16,16),(9,13,13,15),(12,16,16,16),(12,16,16,16),(12,16,16,16)

这样，在第 4 轮的输出状态中，选定的 16 个活跃比特取遍所有 2^{16} 个值，其它比特取定值，可以在第 1 轮的输入处得到平衡状态.

7.3　区分器的第 5～8 轮高阶差分路径

本小节我们将上两个小节的高阶差分路径进行组合，构造一个 14 轮高阶差分路径，从而得到一个 14 轮的零和区分器.

我们知道在积分攻击中，对于一条积分路径来说，如果可以找到一个状态，在经过一轮计算以后，得到的活跃且独立的字节，包含原积分路径起始状态中的所有活跃字节，则可以将这一轮加在原积分路径的前面，这样就构造了一个比原来多一轮的积分路径. 如果原积分路径最后得到平衡状态，则新的多一轮的路径也可以得到平衡状态. 我们在此也利用这一技术，进行两条子高阶差分路径的组合.

如图 9 所示，从第 9 轮的 S 盒层逆向计算可以得出，若使得第 9 轮的第 1、8、15、19 个 S 盒的 16 个比特活跃且独立，需要第 8 轮的 S 盒层的第 1、2、3、4、7、8、9、10、13、14、15、16 个 S 盒的输出比特活跃且独立，而使得第 8 轮的这些 S 盒活跃且独立，需要第 7 轮的前面 20 个 S 盒的输出比特活跃且独立；另

外一个方向，若使得第 5 轮的第 1、2、3、4 个 S 盒的 16 个输入比特活跃且独立，需要第 6 轮的 S 盒层的第 1、6、7、12、17、18 个 S 盒的输出比特活跃且独立，而使得第 6 轮的这些 S 盒活跃且独立，需要第 7 轮的第 1、2、3、5、6、7、8、9、10、12、13、14、16、17、18、19、20、21 个 S 盒的输出比特活跃且独立. 综合这两个方向，取它们的并才能满足所有的条件. 所以，若要使得正向和逆向的起始状态中活跃 S 盒的个数达到前面的要求，必须选择第 7 轮的输入状态的前 21 个 S 盒的 84 个比特活跃.

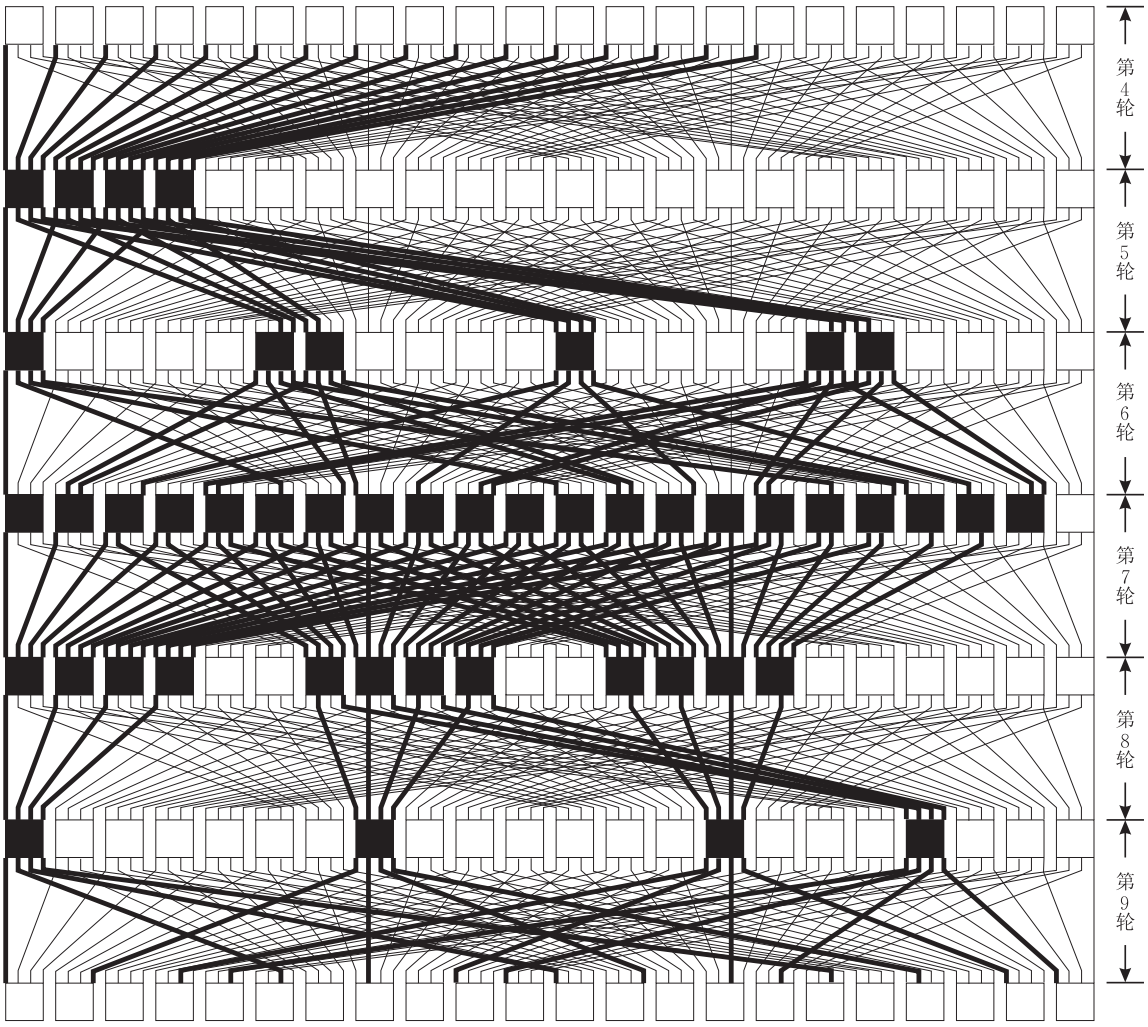


图 9 SPONGENT 零和区分器的 4～9 轮

这样，我们用了 2^{84} 的数据复杂度，得到了 SPONGENT-88/80/8 中间置换的 14 轮零和区分器. 此区分器从第 7 轮的输入出发，选择前 21 个 S 盒的 84 个输入比特作为活跃比特，分别进行正向和逆向的攻击，可以在第 1 轮的输入处得到 2^{84} 个整个 14 轮置换的输入值，使得它们的异或和为零，并且它们对应的输出的异或和也为零.

8 结束语

本文在高阶差分的视角下解释了具有代表性的 AES 和 Rijndael-256 的积分攻击. 并且改进了 Rijndael-256 的 3 轮积分区分器的数据复杂度. 此外，还利用代数次数的增加性质，对一个新的轻量级杂凑函数 SPONGENT 的内部置换，构造了 14 轮的零和区分器.

下一步的研究方向是从积分攻击和高阶差分分析的理论基础出发，进一步更深层次地发现这两种攻击的联系. 此外，进一步改进 SPONGENT 置换的零和区分器，并尝试用于此函数的其它版本中去.

参 考 文 献

[1] FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, U. S. Department of Commerce/N. I. S. T, 2001

[2] Knudsen L, Wagner D. Integral cryptanalysis//Proceedings

of the International Workshop on Fast Software Encryption (FSE 2002). Leuven, Belgium, 2002: 112-127

[3] Daemen J, Knudsen L, Rijmen V. The block cipher Square//Proceedings of the International Workshop on Fast Software Encryption (FSE 1997). Haifa, Israel, 1997: 149-165

[4] Lai Xue-Jia. Higher order derivatives and differential cryptanalysis//Proceedings of the Symposium on Communication, Coding and Cryptography. Ascona, Switzerland, 1994: 227-233

[5] Knudsen L. Truncated and higher order differentials//Proceedings of the International Workshop on Fast Software Encryption (FSE 1994). Leuven, Belgium, 1994: 196-211

[6] Galice S, Minier M. Improving integral attacks against Rijndael-256 up to 9 rounds//Proceedings of the 1st International Conference on Cryptology in Africa (AFRICACRYPT 2008). Casablanca, Morocco, 2008: 1-15

[7] Z'aba M, Raddum H, Henricksen M, Dawson E. Bit-pattern based integral attack//Proceedings of the International Workshop on Fast Software Encryption (FSE 2008). Lausanne, Switzerland, 2008: 363-381

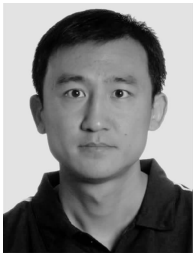
[8] Wei Yue-Chuan, Sun Bing, Li Chao. New integral attack on Rijndael-256. Acta Electronica Sinica, 2011, 39(2): 476-480 (in Chinese)
(魏悦川，孙兵，李超. 对 Rijndael-256 算法新的积分攻击. 电子学报，2011，39(2)：476-480)

[9] Bogdanov A, Knežević M, Leander G, Toz D, Varici K, Verbauwhede I. SPONGENT: A lightweight hash function//Proceedings of the International Workshop of Cryptographic Hardware and Embedded Systems (CHES 2011). Nara, Japan, 2011: 312-325

[10] Daemen J, Rijmen V. The Design of Rijndael. Heidelberg: Springer, 2002

[11] Bogdanov A, Knudsen L, Leander G, Paar C, Poschmann A, Robshaw M, Seurin Y, Vikkelsoe C. PRESENT: An ultra-lightweight block cipher//Proceedings of the International Workshop of Cryptographic Hardware and Embedded Systems (CHES 2007). Vienna, Austria, 2007: 450-466

[12] Feng Deng-Guo, Pei Ding-Yi. An Introduction to Cryptology. Beijing: Science Press, 1999(in Chinese)
(冯登国，裴定一. 密码学导引. 北京：科学出版社，1999)



DONG Le, born in 1980, Ph. D. candidate, lecturer. His research interests include the cryptanalysis of hash functions and block ciphers.

WU Wen-Ling, born in 1966, Ph.D., researcher, Ph. D. supervisor. Her research interests include the cryptanalysis and design of block ciphers.

WU Shuang, born in 1983, Ph. D., assistant professor. His research interests include the cryptanalysis and design of hash functions.

ZOU Jian, born in 1985, Ph.D. candidate. His research interests include the cryptanalysis and design of hash functions.

Background

Integral attack and higher-order differential attack are important techniques for analyzing block ciphers, and we usually apply them with similar steps. However, within our knowledge, no work elaborates the relation between the two. In this paper, we exploit the properties of balanced vectorial Boolean functions. Applying these methods, we expound the relationship of integral attack and higher-order differential attack from the distinguishers of three-round AES and four-round Rijndael-256 with balanced endings. Besides, Wei et al. firstly analyzed Rijndael-256 with bit-pattern based integral and gave a three-round integral distinguisher in a lower data complexity in 2011. We explain the distinguisher in a higher-order differential view and improve it. In 2009, Aumasson et al. presented a zero-sum distinguisher for hash functions which utilizes the theoretical foundation of higher-order differential. We give a fourteen-round zero-sum distinguisher for the inner permutation of the SPONGENT hash function by using a partial higher-order differential technique. Utilizing this technique, a part of the bits in the initial state are selected as the active bits, which is different from the normal higher-order differential attacks.

This research was supported by the National Science and Technology Major Project of China under Grant No. 2011ZX03002-005-02, and the Knowledge Innovation Project of The Chinese Academy of Sciences.