一个新型流密码体制的安全性分析*

张 斌 冯登国

(中国科学院软件研究所信息安全国家重点实验室, 北京 100080)

摘要 分析了一个新型流密码体制——COSvd(2,128)的安全性. COSvd(2,128)使用了钟控非线性反馈移位寄存器(NLFSR)和由混沌序列控制的 *S*-盒, 设计者声称 COSvd(2,128)能够抵抗已知的任何攻击. 然而, 分析表明, 该流密码体制的 *S*-盒设计存在严重漏洞, 它所产生密钥流的字节分布严重不均衡, 在一些很重要的广播加密场合, 这一漏洞会导致成功概率很高的唯密文攻击; 此外, 该流密码体制的其他构件的设计也存在许多漏洞, 根据这些漏洞提出了一个分别征服攻击, 该攻击能够从 $O(2^{26})$字节的已知明文恢复出 COSvd(2,128)的所有密钥, 其成功概率可达 93.4597%, 复杂度为 $O(2^{113})$, 显然这种攻击的复杂度大大低于穷搜索的复杂度 2^{512}.

关键词 流密码 分别征服攻击 非线性反馈移位寄存器**(NLFSR)** 钟控 混沌序列

自从 2004 年 2 月 1 日欧洲开始实施ECRYPT计划以来, 涌现出大量的新型流密码体制. 这些新的流密码体制大都不再像传统流密码体制一样依赖于线性反馈移位寄存器(LFSR)来产生驱动序列, 而是将自身的安全性建立在一些新的驱动构件之上, 如*T*-函数、非线性反馈移位寄存器、随机数组等. 这种设计的目的是阻止针对基于LFSR的流密码的快速相关攻击和代数攻击等, COSvd(2,128)[1]就是一个这样的例子. 它使用了NLFSR, 并采用钟控的方法和所谓的“交叉”机制来产生驱动序列, 这样做使得各种针对流密码的一般攻击方法都无法直接使用; 进一步, 因为COSvd(2,128)的低级版本COS(2,128)[2]并不安全, 设计者附加了一个高度非线性的*S*-盒来隐藏驱动序列的信息, 使得针对低级版本COS(2,128)的攻击失效[3~5]. 另一方面, 根据设计者的声明, COSvd(2,128)这一体制至少被一个商业标准所采纳(文件系统加密软件TURENNE的核心算法), 而对该算法的分析在

收稿日期: 2005-08-24; 接受日期: 2005-12-23
* 国家自然科学基金(批准号: 60273027, 60373047)及“九七三”(2004CB318004)资助项目

公开文献中仅见文献[6], 而且其结果极为初等. 因此, 对COSvd(2,128)进行深入的安全性分析具有很重要的意义.

我们对这一体制进行了深入的安全性分析, 得出如下主要结果: COSvd(2,128)的 *S*-盒的设计存在严重漏洞, 致使它所产生密钥流的字节分布严重不平衡, 这一漏洞会导致在广播加密场合的唯密文攻击, 只需少量的密文和计算量就可以恢复出相应的明文; *S*-盒设计的另一弱点是输出字节与 *S*-盒输入之间存在一定的相关性, 这样就可以对 *S*-盒进行单独处理, 从而剥离 *S*-盒对驱动序列的掩护, 逐个地来恢复出所有密钥; 原设计者并未指明 *S*-盒部分所用混沌序列的实现精度, 在双精度场合, 我们指出这种实现的一个弱点, 并给出这种情况下的分别征服攻击, 攻击需要 $O(2^{26})$字节的已知明文, 能够以 93.4597%的成功概率恢复出所有密钥, 其复杂度为 $O(2^{113})$. 需要强调的是, 我们的攻击适用于任意(更高)精度的场合, 此时整个复杂度的增加十分有限, 而这正是分别征服攻击的特点所在.

下面我们先给出 COSvd(2,128)这一流密码体制的详细描述; 然后具体指出这个体制存在的种种弱点, 并构造一个分别征服攻击来恢复所有密钥, 对这个攻击的各个参数也进行了详细分析; 最后给出我们的结论与建议.

我们的分析与 COSvd(2,128)的密钥装入算法无关, 所以这里略去COSvd(2,128)密钥装入阶段的描述. COSvd(2,128)的密钥流产生算法如下: 令 $L_1 = L_{10}\|L_{11}\|L_{12}\|L_{13}$, $L_2 = L_{20}\|L_{21}\|L_{22}\|L_{23}$ 表示两个长为 128 bit 的非线性反馈移位寄存器(NLFSR), 其中符号"$\|$"表示 bit 串的直接连接, 对 $i = 1,2$ 且 $j = 0,1,2,3$, L_{ij} 都是 32 bit 的字. 在每一步, 根据下述步骤产生密钥流:

1) 计算钟控值 d 和 d': 令 $clk = 2\cdot lsb(L_2) + lsb(L_1)$, 则 $d = C[clk]$, 这里 $lsb(\cdot)$ 表示自变量的最低位; 再令 $clk = 2\cdot msb(L_1) + msb(L_2)$, 则 $d' = C'[clk]$, 这里 $msb(\cdot)$ 表示自变量的最高位, $C[0,\cdots,3] = \{64,65,66,64\}$, $C'[0,\cdots,3] = \{41,43,47,51\}$;

2) 在第 i 步, 若 i 是偶数, 令 L_1 前进 d 步, L_2 前进 d' 步; 若 i 是奇数, 令 L_1 前进 d' 步, L_2 前进 d 步;

3) 根据如下方式产生一个 128 bit 的字 $B_i = B_{i0}\|B_{i1}\|B_{i2}\|B_{i3}$: $B_{i0} = L_{20}\oplus L_{12}$, $B_{i1} = L_{21}\oplus L_{13}$, $B_{i2} = L_{22}\oplus L_{10}$ 以及 $B_{i3} = L_{23}\oplus L_{11}$, 这里符号"$\oplus$"表示对应 bit 的异或, 见图 1;

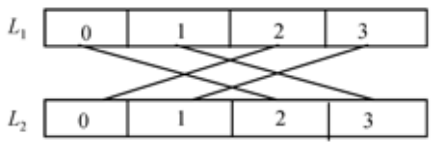

图 1 COSvd 的交叉机制

4) 将 B_i 按顺序分成 16 个字节 B_i^j, $0 \le j < 16$; 然后把 B_i 逐字节的输入下

面描述的非线性滤波器 HNLL(highly non-linear layer)中产生相应一个字节的输出.

非线性滤波器**HNLL**　设计这一模块的目的是掩藏NLFSR的内部状态, 阻止针对低版本COS(2,128)的攻击[3~5]. HNLL对每个 B_i^j 应用一段 10 bit的混沌序列来决定*S*-盒此时的输出. 设计者用henon映射[7]来构造一条混沌序列, 亦即对一个初始点 (x_{-2}, x_{-1}), 按 $x_n = 1 + 0.3x_{n-2} - 1.4x_{n-1}^2$ 计算出 x_n, 若 $x_n > 0.39912$, 则令 $z_n = 1$; 否则令 $z_n = 0$, 这样得到henon序列 $\{z_i\}$. 以相同的方式从另一初始点 (x'_{-2}, x'_{-1}) 产生另一条henon序列 $\{z'_i\}$, 令 $z_i \oplus z'_i = h_i$, 得到序列 $\{h_i\}$. 对每个 $B_i^j = b_{i7}^j, b_{i6}^j, \cdots, b_{i0}^j$, 从序列 $\{h_i\}$ 中截取两个混沌序列段

$$H_i^j = h_{10(16i+j)} \| h_{10(16i+j)+1} \| \cdots \| h_{10(16i+j)+7}$$

和

$$K_i^j = h_{10(16i+j)+8} \| h_{10(16i+j)+9},$$

由 Tabkey2 $[B_i^j \oplus H_i^j][K_i^j]$ 产生一个字节的输出, 这里的 Tabkey2 是一个 $GF(2)^8 \times GF(2)^2 \to GF(2)^8$ 的 *S*-盒, 其具体描述见附录 A.

一般来说, COSvd(2,128)的密钥包括两个NLFSR的初始状态和两个henon映射的初始点 (x_{-2}, x_{-1}) 及 (x'_{-2}, x'_{-1}). 原设计者[1]并未指明实现混沌序列的精度, 下面我们将假设以双精度来实现混沌序列, 根据IEEE 754 的浮点数标准[8], 以一个 64 bit的字来表示一个双精度实数, 从而整个COSvd(2,128)的密钥空间为 2^{512}. 事实上, 下面的攻击适用于任意精度的混沌实现, 我们这样做一方面是因为双精度实现是一种重要的实际应用, 另一方面也是要指出COSvd(2,128)在这种实现下的一个弱点. 下面关于双精度实现的结论不止适用于COSvd(2,128), 我们的结论具有普遍意义.

根据附录 A 中 Tabkey2 的描述, 我们通过计算机得到下面关于 $[0,256)$ 之间的每个整数在 Tabkey2 中的分布表 1.

表 1　$[0,256)$ 间每个整数在 Tabkey2 中的分布

出现次数	个数	具体值
2	1	{0}
3	8	{2,11,50,52,65,104,160,208}
5	8	{4,5,19,32,37,75,174,151}
6	1	{17}
4	238	其他的数

由表 1 可见, 虽然有 238 个整数在 Tabkey2 中出现了 4 次, 但其他 18 个整数出现次数均多于或少于 4 次. 举例来说, 字节 17 出现了 6 次, 是随机情况的 1.5 倍. 我们进行了计算机模拟试验来验证上述观察, 结果表明在 COSvd(2,128)产生的 2^{18} 字节的密钥流中, 17 出现的概率为 $2^{-8}+1.05\times 2^{-9}$, 这非常接近于理论值 $3/2^9$.

上述字节分布的偏差在某些广播加密(如电子邮件群发)的场合对整个体制安全性的威胁是致命的. 在这些场合, 同一个消息被不同的多个密钥段加密并分别发送给不同的多个用户[9]. 假设一个包含 l 字节的消息 $m=m_1m_2\cdots m_l$ 被 COSvd(2,128)加密了 N 次, 对每个 m_i, 相应的密文字节为 $m_i\oplus c_j$, $1\le j\le N$. 由于密钥流字节分布的不均衡性, $17\oplus m_i$ 应该是密文中出现次数最多的字节. 假设 $p=2^{-8}+2^{-9}$, 那么在 N 个密钥流字节里, 字节 17 出现的概率服从二项分布 (N,p); 而在随机情况下, 它服从二项分布 $(N,p'=2^{-8})$. 我们用正态分布来近似代替二项分布, 设 $u=Np$, $\sigma=\sqrt{Np(1-p)}$ 分别表示 17 出现的数学期望与标准差; u' 和 σ' 分别表示随机情况下的这两个量. 那么如果 $|u-u'|>2(\sigma+\sigma')$, 亦即 $N>2^{16.31}$, 那么字节序列 $m_i\oplus c_j$ 就能以 0.9772 的概率与随机字节序列 $m_i'\oplus c_j$ 区分开来. 这样我们就可以逐字节的恢复出消息 m. 计算机模拟试验的结果表明, 如果我们把一个消息用COSvd(2,128)密钥流的不同字段加密 2^{17} 次, 那么就可以成功地通过把 17 和密文里出现频率最高的字节相异或来恢复出对应的消息字节.

COSvd(2,128)的 S-盒除了存在上述漏洞之外, 还有一些其他形式的信息泄漏. 举例来说, 如果字节 50 出现在密钥流中, 那么我们就知道 3 个产生这个字节的候选位置为 $(68,0)$, $(108,2)$ 和 $(115,3)$, 把这些位置信息转化成二进制形式, 我们就得到

$$\begin{aligned} &68\rightarrow 01000100, \qquad &&0\rightarrow 00,\\ &108\rightarrow 01101100, \qquad &&2\rightarrow 10,\\ &115\rightarrow 01110011, \qquad &&3\rightarrow 11. \end{aligned}$$

仔细观察二进制表示的每一列, 我们得到

$$b_{i7}^j\oplus h_{10(16i+j)+7}=b_{i6}^j\oplus h_{10(16i+j)+6}\oplus 1=0,$$

$$b_{i0}^j\oplus h_{10(16i+j)}=b_{i1}^j\oplus h_{10(16i+j)+1}=h_{10(16i+j)+9},$$

$$b_{i0}^j\oplus h_{10(16i+j)}=b_{i4}^j\oplus h_{10(16i+j)+4},$$

$$b_{i2}^j\oplus h_{10(16i+j)+2}=b_{i4}^j\oplus h_{10(16i+j)+4}\oplus 1,$$

$$b_{i5}^j\oplus h_{10(16i+j)+5}=h_{10(16i+j)+8},$$

$$b_{i4}^{j} \oplus h_{10(16i+j)+4} \oplus b_{i5}^{j} \oplus h_{10(16i+j)+5} = b_{i3}^{j} \oplus h_{10(16i+j)+3},$$

$$b_{i2}^{j} \oplus h_{10(16i+j)+2} \oplus b_{i3}^{j} \oplus h_{10(16i+j)+3} = 1 \oplus b_{i5}^{j} \oplus h_{10(16i+j)+5},$$

以及

$$P\left(h_{10(16i+j)+8} = 1\right) = 2/3 \text{ 和 } P\left(h_{10(16i+j)+9} = 0\right) = 2/3.$$

从上述等式可以看出, *S*-盒的输出与输入之间相关性很强. 事实上, 我们用计算机试验了 $[0,256)$ 之间的所有整数, 结果表明当任一整数作为输出字节时, 我们都可以得到类似上面的线性等式组, 唯一的区别是等式个数的多少. 因此, 从密钥流我们可以得到大量关于 $B_i^j \oplus H_i^j$ 及 K_i^j 的线性等式, 这时容易想到的是能否使用已知明文攻击彻底破解这个体制, 但注意到混沌序列的存在, 这个想法很难实现.

下面我们给出一个分别征服攻击来剥掉 HNLL 这层“皮”, 然后再恢复出两个 NLFSR 的初始内部状态.

首先, 我们给出一个关于henon序列重要的观察. 注意到由于 0.39912 这个值的选取, henon序列的单bit分布是均衡的, 但它的两bit串分布却是严重不平 衡的[10]. 通过计算机随机选取由 $(-1.33,0.42)$, $(1.32,0.133)$, $(1.245,-0.14)$ 及 $(-1.06,-0.5)$ 定义的henon序列收敛四边形Q内的初始点, 然后产生 2^{30} bit的混沌序列, 我们得到 (z_i, z_{i+1}) 为 $(0,0)$, $(0,1)$, $(1,0)$ 及 $(1,1)$ 的概率分别为 $1/4-2^{-3.2811}$, $1/4+2^{-3.36745}$, $1/4+2^{-3.36745}$ 及 $1/4-2^{-3.45931}$. 这些严重背离 1/4 的偏差构成了我们攻击的基础. 另一方面, 在双精度实现的假设下, 混沌序列还有另一方面的弱点. 根据IEEE 754 的浮点数标准[8], 用一个 64 bit的字来表示一个双精度浮点数. 具体来说, 1 位表示符号; 11 位表示指数部分; 52 位表示数值部分. 设 $e = e_{10}e_9\cdots e_0$ 为 11 位的指数, $d_1d_2\cdots d_{52}$ 表示 52 位的数值, 这里 e_i, $d_i \in GF(2)$. 根据IEEE 754 的浮点数标准, 一个双精度实数的绝对值为

$$\left(1+\sum_{i=1}^{52} d_i \cdot 2^{-i}\right) \cdot 2^{e^{-1023}},$$

我们的观察在于: 如果我们只搜索低三位 $e_2e_1e_0$, 而令 $(e_{10}, e_9, \cdots, e_3) = (0,1,\cdots,1)$, 那么我们没有搜索到的点只占 Q 的很小一部分, 见图 2.

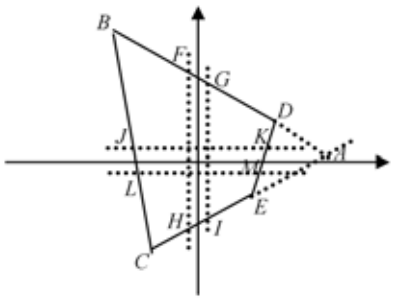

图 2　猜测的区域

图 2 中, 横、竖两个梯形(*JLMK* 和 *FHIG*)所覆盖的面积中只包含至少一个坐标绝对值小于 $2^{-7}+2^{-7}\cdot(1-2^{-52}) < 2^{-6} = 0.015625$ 的点,

我们使用 Helen 公式计算出图 2 中大(ΔABC)、小(ΔADE)两个三角形的面积, 它们数值之差就是 Q 的面积; 而两个梯形($JLMK$ 和 $FHIG$)面积之和再减去中间小正方形(相交部分)的面积就是猜测不到的点所覆盖的面积, 两部分之比为

$$(0.00953758+0.0389093-0.000244141)/1.48139=3.25389\%.$$

这样, 如果我们用这种方法穷搜一条 henon 序列的初始点 (x_{-2},x_{-1}), 那么复杂度仅为 $O(2^{112})$, 而成功概率却达到 96.7461%.

现在来具体描述我们的分别征服攻击. 基本思想是: 如果我们猜测一条 henon 序列的初始点 (x_{-2},x_{-1}), 那么通过密钥流和这个初始点所产生的序列 z_i, 我们可以得到 K_i^j 的部分信息; 然后求出 K_i^j 与 z_i 相异或得出序列的两 bit 串分布, 如果这个分布在意料之中, 那么就接受 (x_{-2},x_{-1}) 的猜测, 否则就拒绝它.

首先根据上面单条 henon 序列的两 bit 串分布来计算出两条相异或的 henon 序列的两 bit 串分布. 对单条 henon 序列, (采用四进制符号)记 $P(0)=1/4-2^{-3.2811}$ 为 $(0,0)$ 出现的概率, $P(1)=1/4+2^{-3.36745}$ 为 $(0,1)$ 出现的概率, $P(2)=1/4+2^{-3.36745}$ 为 $(1,0)$ 出现的概率, $P(3)=1/4-2^{-3.45931}$ 为 $(1,1)$ 出现的概率, 那么两条 henon 序列异或后, 相应的分布为

$$(0,0):\ \sum_{i=0}^{3}P(0)^2=0.287625;\qquad (0,1):\ \sum_{i=0}^{3}P(i\oplus 1)\cdot P(i)=0.212447;$$

$$(1,0):\ \sum_{i=0}^{3}P(i\oplus 2)\cdot P(i)=0.212447;\quad (1,1):\ \sum_{i=0}^{3}P(i\oplus 3)\cdot P(i)=0.287482.$$

现在摆在我们面前的困难在于: 即使在某些情况下可以得到一些单个 bit 的分布, 但从密钥流我们无法唯一确定出相应的 K_i^j 值. 然而我们注意到这个问题在相应的输出字节为 0 时可以解决. 此时相对应的 K_i^j 要么为 $(1,0)$, 要么为 $(0,1)$, 这就意味着如果我们猜测一条 henon 序列的初始点 (x_{-2},x_{-1}), 产生一定长度的 z_i, 然后把输出字节 0 相应的 $(z_{10(16i+j)+8},z_{10(16i+j)+9})$ 和 $(0,1)$ 与 $(1,0)$ 分别相异或, 那么我们事实上是把 $(z_{10(16i+j)+8},z_{10(16i+j)+9})$ 分别与 K_i^j 和 $K_i^j\oplus(1,1)$ 相异或, 图 3

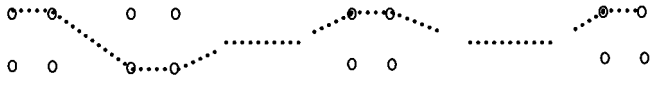

图 3 猜测的过程

表明了这个过程.

图 3 中有虚线相连的点表示实际的 K_i^j, 其相对应的两点就是 $K_i^j \oplus (1,1)$, 这样我们就可以计算异或后得到序列的两 bit 串分布.

当输出的字节 0 对应于 $K_i^j = (1,0)$ 时, 相应的 z_i' 中两 bit 串的分布为

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (0,0) \middle| K_i^j = (1,0)\right)$$

$$= \frac{P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (0,0), (z_{10(16i+j)+8}, z_{10(16i+j)+9}) = (1,0)\right)}{P\left(K_i^j = (1,0)\right)}$$

$$= \frac{\left(1/4 - 2^{-3.2811}\right)\left(1/4 + 2^{-.36745}\right)}{0.212447} = 0.24024;$$

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (0,1) \middle| K_i^j = (1,0)\right)$$

$$= \frac{P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (0,1), (z_{10(16i+j)+8}, z_{10(16i+j)+9}) = (1,1)\right)}{P\left(K_i^j = (1,0)\right)}$$

$$= \frac{\left(1/4 - 2^{-3.45931}\right)\left(1/4 + 2^{-.36745}\right)}{0.212447} = 0.259759.$$

类似地,

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (1,0) \middle| K_i^j = (1,0)\right) = 0.24024;$$

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (1,1) \middle| K_i^j = (1,0)\right) = 0.259759.$$

相应地, 当输出的字节 0 对应于 $K_i^j = (0,1)$ 时, 相应的 z_i' 中两 bit 串的分布为

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (0,0) \middle| K_i^j = (0,1)\right) = 0.24024;$$

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (0,1) \middle| K_i^j = (0,1)\right) = 0.24024;$$

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (1,0) \middle| K_i^j = (0,1)\right) = 0.259759;$$

$$P\left((z'_{10(16i+j)+8}, z'_{10(16i+j)+9}) = (1,1) \middle| K_i^j = (0,1)\right) = 0.259759.$$

这样, 当 (x_{-2}, x_{-1}) 猜对时, 得到的两 bit 串分布为

$$\frac{0.24024 \cdot n/2 + 0.24024 \cdot n/2 + 0.259759 \cdot n/2 + 0.259759 \cdot n/2}{2n}$$

$$= \frac{(0.24024 + 0.259759)}{2} = 0.25, \tag{1}$$

亦即, 此时应为均匀分布, (1)式中 n 为密钥流中字节 0 出现的次数. 以两 bit 串

$(0,0)$ 为例, 此时图 3 中上下两条序列的总计数个数为 $2n$, 字节 0 对应于 $K_i^j=(1,0)$ 和 $K_i^j=(0,1)$ 的概率均为 0.5. 当 $K_i^j=(1,0)$ 时, z_i' 中串 $(0,0)$ 出现的概率为 0.24024, z_i' 中出现 $(1,1)$ 方能使得另一半结果序列中出现 $(0,0)$, 而 z_i' 中出现 $(1,1)$ 的概率为 0.259759; 对于 $K_i^j=(0,1)$ 时的情况可以类似处理, 这样就得到了(1)式. 这里的分析也说明了当 (x_{-2},x_{-1}) 猜对时不能按统计独立的情况来处理, 而应该按条件分布来处理, 后面列举的实验结果也证明了这一点.

当 (x_{-2},x_{-1}) 猜错时, 假设产生混沌序列 z_i'', 此时可以把 z_i'' 看作与 z_i 和 z_i' 统计独立, 这样就有

$$\begin{aligned}&\left(z''_{10(16i+j)+8},z''_{10(16i+j)+9}\right)\oplus\left(z_{10(16i+j)+8},z_{10(16i+j)+9}\right)\oplus\left(z'_{10(16i+j)+8},z'_{10(16i+j)+9}\right)\\&=\left(z''_{10(16i+j)+8},z''_{10(16i+j)+9}\right)\oplus(1,0)\text{或}\left(z''_{10(16i+j)+8},z''_{10(16i+j)+9}\right)\oplus(0,1),\end{aligned}$$

亦即, 如果 $K_i^j=(1,0)$, 序列 z_i'' 的 $(0,0)$ 映射到 $(1,0)$, $(0,1)$ 映射到 $(1,1)$, $(1,0)$ 映射到 $(0,0)$ 及 $(1,1)$ 映射到 $(0,1)$; 如果 $K_i^j=(0,1)$, 序列 z_i'' 的 $(0,0)$ 映射到 $(0,1)$, $(0,1)$ 映射到 $(0,0)$, $(1,0)$ 映射到 $(1,1)$ 及 $(1,1)$ 映射到 $(1,0)$. 这样, 此时的两 bit 串分布为

$$(0,0):\ \frac{\frac{n}{2}\left(\frac14+2^{-3.36745}+\frac14+2^{-3.36745}+\frac14+2^{-3.36745}+\frac14+2^{-3.36745}\right)}{2n}=\frac14+2^{-3.36745};$$

$$(0,1):\ \frac{\frac{n}{2}\left(\frac14-2^{-3.45931}+\frac14-2^{-3.2811}+\frac14-2^{-3.2811}+\frac14-2^{-3.45931}\right)}{2n}=\frac14-2^{-3.36746};$$

$$(1,0):\ \frac{\frac{n}{2}\left(\frac14-2^{-3.45931}+\frac14-2^{-3.2811}+\frac14-2^{-3.2811}+\frac14-2^{-3.45931}\right)}{2n}=\frac14-2^{-3.36746};$$

$$(1,1):\ \frac{\frac{n}{2}\left(\frac14+2^{-3.36745}+\frac14+2^{-3.36745}+\frac14+2^{-3.36745}+\frac14+2^{-3.36745}\right)}{2n}=\frac14+2^{-3.36745}.$$

还是以串 $(0,0)$ 为例来说明这些表达式推导, 此时图 3 中上下两条序列的总计数个数仍为 $2n$, 字节 0 对应于 $K_i^j=(1,0)$ 和 $K_i^j=(0,1)$ 的概率均为 0.5. 当 $K_i^j=(1,0)$ 时, z_i'' 中出现 $(1,0)$ 方能使得一半结果序列中出现 $(0,0)$, 而 z_i' 中出现 $(0,1)$ 方能使得另一半结果序列出现 $(0,0)$; $K_i^j=(0,1)$ 时的情况可以类似处理.

很明显, 猜错时的分布与(1)式中的分布明显不同, 这样就可以用作淘汰错误猜测的依据. 我们进行了计算机模拟实验来验证上述结论. 计算机产生了 2^{30}bit 的混沌序列 $\{z_i \oplus z'_i\}$, 一旦 $(z_i, z_{i+1}) \oplus (z'_i, z'_{i+1}) = (0,1)$ 或 $(1,0)$, 当初始点 (x_{-2}, x_{-1}) 猜测正确时, 由 $(z_i, z_{i+1}) \oplus (0,1)$ 与 $(z_i, z_{i+1}) \oplus (1,0)$ 所得序列中 $(0,0)$ 出现的概率为 0.25016676; 而当猜测错误时, 这个概率为 $1/4 + 2^{-3.36799}$. 这就进一步验证了我们的分析的正确性.

假设已知 $O(2^8 \cdot n)$ 字节的密钥流, 那么就可以得到 $O(n/2)$ 个非连续的 0 字节, 这是因为根据表 1, 字节 0 出现的概率为 2^{-9}, 另一方面这样假设也是为了简化下面的推导. 随机猜测一条 henon 序列的初始点 (x_{-2}, x_{-1}), 产生一定长度的序列 z_i (n 和这里所谓的一定长度下面会逐步确定), 对密钥流中出现的每个 0 字节, 将相对应的 $(z_{10(16i+j)+8}, z_{10(16i+j)+9})$ 与 $(0,1)$ 和 $(1,0)$ 分别异或, 记数组序列

$$G = \begin{Bmatrix} (z_{10(16i+j)+8}, z_{10(16i+j)+9}) \oplus (0,1) \\ (z_{10(16i+j)+8}, z_{10(16i+j)+9}) \oplus (1,0) \end{Bmatrix}_{i,j}$$

中串 $(0,0)$, $(0,1)$, $(1,0)$ 及 $(1,1)$ 的个数分别为 a_0, a_1, a_2 及 a_3. 如果 (x_{-2}, x_{-1}) 猜得对, 那么串 $(0,0)$, $(0,1)$, $(1,0)$ 及 $(1,1)$ 将均匀分布. 设

$$A = \left\{ (a_0, a_1, a_2, a_3) \middle| n \cdot \left(\frac{1}{4} - x\right) \le a_0, a_1, a_2, a_3 \le n \cdot \left(\frac{1}{4} + x\right) \text{且} a_0 + a_1 + a_2 + a_3 = n \right\},$$

这里的 x 和 n 一样是留在后面要决定的参数值. 如果猜测正确, 那么我们要求 $(a_0, a_1, a_2, a_3) \in A$ 的概率 P_1 非常接近于 1. P_1 和猜测错误时 $(a_0, a_1, a_2, a_3) \in A$ 的概率 P_2 如下:

$$P_1 = \sum_{(a_0,a_1,a_2,a_3) \in A} \frac{n!}{a_0!a_1!a_2!a_3!} \cdot \left(\frac{1}{4}\right)^n;$$

$$P_2 = \sum_{(a_0,a_1,a_2,a_3) \in A} \frac{n!}{a_0!a_1!a_2!a_3!} \cdot \left(\frac{1}{4} + 2^{-3.36745}\right)^{a_0+a_3} \cdot \left(\frac{1}{4} - 2^{-3.36746}\right)^{a_1+a_2}.$$

由于多项系数的存在, P_1 和 P_2 并不便于实际计算, 下面我们用多元正态分布来逼近 P_1 和 P_2:

$$P_1 \to \int_{n\left(\frac{1}{4}-x\right)-0.5}^{n\left(\frac{1}{4}+x\right)+0.5} \int_{n\left(\frac{1}{4}-x\right)-0.5}^{n\left(\frac{1}{4}+x\right)+0.5} \int_{n\left(\frac{1}{4}-x\right)-0.5}^{n\left(\frac{1}{4}+x\right)+0.5} f(y_1, y_2, y_3) \mathrm{d}y_1 \mathrm{d}y_2 \mathrm{d}y_3,$$

$$P_2 \to \int_{n\left(\frac{1}{4}-x\right)-0.5}^{n\left(\frac{1}{4}+x\right)+0.5}\int_{n\left(\frac{1}{4}-x\right)-0.5}^{n\left(\frac{1}{4}+x\right)+0.5}\int_{n\left(\frac{1}{4}-x\right)-0.5}^{n\left(\frac{1}{4}+x\right)+0.5} g\left(y_1,y_2,y_3\right)\mathrm{d}y_1\mathrm{d}y_2\mathrm{d}y_3,$$

其中

$$f\left(y_1,y_2,y_3\right)=\frac{1}{\sqrt{\left(2\pi\right)^3\cdot|B|}}\cdot e^{-(y-u)^{\mathrm{T}}B^{-1}(y-u)/2};$$

$$g\left(y_1,y_2,y_3\right)=\frac{1}{\sqrt{\left(2\pi\right)^3\cdot|B'|}}\cdot e^{-(y-u')^{\mathrm{T}}B'^{-1}(y-u')/2};$$

$$y=\left(y_1,y_2,y_3\right)^{\mathrm{T}};\quad u=\left(\frac{n}{4},\frac{n}{4},\frac{n}{4}\right)^{\mathrm{T}};$$

$$u'=\left(\left(\frac{1}{4}-2^{-3.36746}\right)n,\ \left(\frac{1}{4}+2^{-3.36745}\right)n,\ \left(\frac{1}{4}+2^{-3.36745}\right)n\right)^{\mathrm{T}};$$

$$B=\begin{pmatrix}\frac{3}{16}n & -\frac{1}{16}n & -\frac{1}{16}n\\ -\frac{1}{16}n & \frac{3}{16}n & -\frac{1}{16}n\\ -\frac{1}{16}n & -\frac{1}{16}n & \frac{3}{16}n\end{pmatrix};$$

$$B'=\begin{pmatrix}0.129665n & -0.0531118n & -0.0531118n\\ -0.0531118n & 0.226559n & -0.120335n\\ -0.0531118n & -0.120335n & 0.226559n\end{pmatrix}.$$

对不同的 n 和 x 的值, 我们用 Mathematica 4.0 计算出 P_1 与 P_2 的值. 我们发现在 $n=2^{12}$ 和 $x=96/2^{12}$ 时, $P_1=0.998521$, 而 $P_2=2^{-193.725}<\frac{1}{2^{112}}$. 此时获得的关于初始点 $\left(x_{-2},x_{-1}\right)$ 的信息为

$$\Delta=112-\left(-\sum_{i=1}^{|H|}\frac{P_1}{|H|}\cdot\log_2\frac{P_1}{|H|}-\sum_{i=1}^{|\overline{H}|}\frac{1-P_1}{|\overline{H}|}\cdot\log_2\frac{1-P_1}{|\overline{H}|}\right)$$

$$=P_1\cdot\log_2\frac{P_1}{P_2+2^{-112}}+\left(1-P_1\right)\cdot\log_2\frac{1-P_1}{1-P_2-2^{-112}}=111.818,$$

其中

$$H=\left\{\left(x_{-2},x_{-1}\right)\mid\text{对 }G,\left(a_0,a_1,a_2,a_3\right)\in A\right\},$$

$$\overline{H}=\left\{\left(x_{-2},x_{-1}\right)\mid\text{对 }G,\left(a_0,a_1,a_2,a_3\right)\notin A\right\}.$$

至此, 我们从 $O(2^8\times 2^{12})=O(2^{20})$ 字节的密钥流, 以 $0.967461\times 0.998521=96.603\%$ 的成功概率恢复出一条 henon 序列的初始点 (x_{-2},x_{-1}), 复杂度为 $O(2^{112})$. 下一步, 我们将着手来恢复出另一条 henon 序列的初始点 (x'_{-2},x'_{-1}).

注意到此时已知 (x_{-2},x_{-1}), 如果我们穷搜 (x'_{-2},x'_{-1}), 产生出一定长度的混沌序列 z'_i, 那么当输出字节为 0 时, 有 $h_{10(16i+j)+8}=1\oplus h_{10(16i+j)+9}$. 因此, 如果有 2^{16} 字节长的密钥流, 那么就有 2^7 个输出 0 字节, 这样, 一个错误的 (x'_{-2},x'_{-1}) 猜测能连续通过 2^7 次检查 $h_{10(16i+j)+8}=1\oplus h_{10(16i+j)+9}$ 是否成立的概率为 $2^{-128}<2^{-112}$. 于是我们就淘汰掉了错误猜测, 恢复出 (x'_{-2},x'_{-1}). 至此, 我们攻击的复杂度为 $O(2^{112}+2^{112})=O(2^{113})$, 所需密钥流长度仍为 $O(2^{20})$ 字节, 成功概率为 $0.967461^2\times 0.998521=93.4597\%$.

一旦我们知道了 (x_{-2},x_{-1}) 与 (x'_{-2},x'_{-1}), 我们就得到每个 K_i^j 的值, 这样从相对应的输出字节就可以确定出 S-盒的另一输入 $B_i^j\oplus H_i^j$ (绝大多数情况下, 可以唯一确定出 $B_i^j\oplus H_i^j$; 例外的情况是输出为 Tabkey2 中出现≥5 次的 9 个字节时, 此时每种情况下, 也只有 1 或 2 个 K_i^j 的值引起很小的不确定性, 这时整个攻击的复杂度只增加一个常数量级, 因此我们可以安全地假定唯一知道 $B_i^j\oplus H_i^j$ 的值, 进而知道 B_i^j 的值).

下面考察如图 4 所示的从一奇数步开始的连续 7 步. L_1 在第 2, 4, 6 步都前进 d 步, 在其余各步都前进 d' 步; L_2 则正相反. 在图 4, a,b,c,x,w,y,S,U,k_i

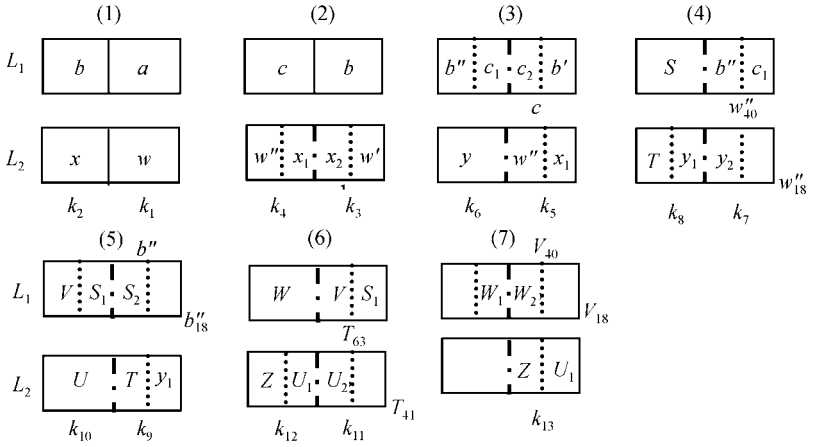

图 4　连续的 7 步

$(1 \le i \le 13)$, W 都是 64 bit 的字; 不失一般性, 我们假设在这 7 步里, $d = 64$, $d' = 41$. 这种情况发生的概率是 $(2^{-3})^6 = 2^{-18}$, 其他情况都可类似处理. 注意这里假设从某一奇数步开始, 而上一步的内部状态决定了下一步产生时前进的步数, 我们的考察到第 7 步为止, 因此无需再乘一个 2^{-3}. 这样图 4 中 bit 串 $w', w'', x_1, x_2, b', b'', c_1, c_2, y_1, y_2, T, V$ 以及 $S_1, S_2, Z, U_1, U_2, W_1, W_2$ 的长度都可根据 d 和 d' 的值确定. 我们用 $k_i\ (1 \le i \le 13)$ 表示两个 NLFSR 的内部状态经过交叉异或之后得出的 64 bit 字, 这样, 在第 1 步就有

$$\begin{cases} b_{63} \oplus w_{63} = k_{1,63}, \\ \cdots \\ b_0 \oplus w_0 = k_{1,0}, \\ a_{63} \oplus x_{63} = k_{2,63}, \\ \cdots \\ a_0 \oplus x_0 = k_{2,0}, \\ b_{63} = 0, \\ x_{63} = 0, \\ a_0 \oplus w_0 = 0, \end{cases}$$

其中 $b = b_{63} \cdots b_0$, $w = w_{63} \cdots w_0$, $a = a_{63} \cdots a_0$, $x = x_{63} \cdots x_0$ 及 $k_i = k_{i,63} \cdots k_{i,0}$ $(1 \le i \le 13)$. a, b, x 及 w 共有 256 个未知 bit, 而通过第 1 步, 我们已经得到 131 个线性方程. 这正是交叉体制的一个致命缺点——太多的内部状态暴露了出来. 如果我们多前进几步, 我们完全能够得到更多关于 a, b, x 及 w 的 256 个 bit 的方程. 通过第 2 步, 我们有

$$\begin{cases} c_{63} \oplus x_{40} = k_{3,63}, \\ \cdots \\ c_{23} \oplus x_0 = k_{3,23}, \\ c_{22} \oplus w_{63} = k_{3,22}, \\ \cdots \\ c_0 \oplus w_{41} = k_{3,0}, \\ b_{63} \oplus w''_{40} = k_{4,63}, \\ \cdots \\ b_{23} \oplus w''_0 = k_{4,23}, \\ c_{63} = 0, \\ w''_{40} = 0, \\ b_0 \oplus w_{41} = 0. \end{cases}$$

特别地, 还有

$$\begin{cases} b_{22} \oplus x_{63} = k_{4,22}, \\ \cdots \\ b_0 \oplus x_{41} = k_{4,0}. \end{cases}$$

亦即, 我们又得到了 23 个关于 a, b, x, w 的线性方程. 此外, 我们注意到在这一步新加进来的两个变量 c 和 w'' 的各个 bit 能够表示成 b, x, w 中的 bit 和密钥流字节 bit 的线性组合. 随着步数增加, 所有新加进来变量当中的 bit 都可以表示成 b, x, w 中的 bit 和密钥流字节 bit 的线性组合, 而且, 每多一步就多得到 23 个关于 a, b, x, w 的线性方程, 例如, 在第 3 步, 我们得到

$$\begin{cases} c_{63} \oplus x_{63} = k_{3,63} \oplus x_{40} \oplus x_{63} = k_{5,22}, \\ \cdots \\ c_{41} \oplus x_{41} = k_{3,41} \oplus x_{18} \oplus x_{41} = k_{5,0}, \end{cases}$$

等等.

在第 7 步, 我们有

$$\begin{cases} W_{41} \oplus U_{41} = k_{11,41} \oplus k_{10,18} \oplus k_{5,59} \oplus k_{4,59} \oplus b_{59} \oplus k_{12,0} \oplus k_{9,0} \oplus k_{8,0} \oplus c_{41} = k_{13,0}, \\ \cdots \\ W_{63} \oplus U_{63} = k_{11,63} \oplus k_{10,40} \oplus k_{7,17} \oplus k_{4,58} \oplus b_{58} \oplus k_{12,22} \oplus k_{9,22} \oplus k_{8,22} \oplus c_{63} = k_{13,22}. \end{cases}$$

这样, 在 7 步完了以后, 我们至少得到 $128 + 23 \cdot 6 = 266$ 个关于 a, b, x, w 包含 bit 的线性方程. 检查这个线性方程组的一致性可以淘汰掉错误的位置; 当一致性满足时解这个方程组就可以恢复出 a, b, x, w. 这一步需要的密钥流长度为 $7 \times 2^{18} \times 2 \times 2^4 \to O(2^{26})$ 字节的密钥流, 复杂度为 $O\left(2^{24} \cdot (2^{25} - 1) + 2^{24}\right) = O\left(2^{49}\right)$. 这里密钥流长度的计算过程如下: 我们需要连续的 7 步, 而每步包含 2^4 个字节, 我们所考察的情况发生的概率为 2^{-18}, 又因为从奇数步开始考察, 所以需要再乘一个因子 2.

至此, 我们就从 $O\left(2^{26}\right)$ 字节的密钥流, 以 93.4597%的概率恢复出了 COSvd(2,128)的所有密钥, 我们攻击的复杂度为 $O\left(2^{113} + 2^{49}\right) = O\left(2^{113}\right)$. 通过分析可以看出, COSvd(2,128)的设计并不成功, 这个密码的安全性主要在于混沌序列的存在, 其他部分在密码学意义下都很弱. 虽然我们攻击的各个参数都是在双精度假设下得出的, 它实际上适用于任意精度的情况, 唯一的区别就是复杂度增加一个很有限的量级, 而这主要是由穷搜一条 henon 序列的初始点造成的, 攻击的其他部分不变. 注意到任意更高精度的实现只不过是由长一些的计算机字来完成, 设对应于某一实现的计算机字长度为 $l > 64$ bit, 那么我们攻击的复杂度就

变为 $O(2\cdot 2^{2l})$, 而此时的成功概率为 100%.

本文分析了 COSvd(2,128)的安全性. 结果表明, 虽然 COSvd(2,128)不仅被一个商业标准采纳, 但它的设计并不成功. 我们给出的攻击要远远快于穷搜索的复杂度, 因此, 我们建议 COSvd(2,128)应至少在以下两个方面进行修改:

1) 采用 henon 算法之外的混沌序列发生器, 要求该发生器产生的序列最好具有理想的游程分布;

2) 采用一个更好的 *S*-盒, 要求使得输出密钥流序列具有均匀的字节分布和游程分布, 并对差分和线性密码分析免疫.

参 考 文 献

1 Filiol E, Fontaine C, Josse S. The COSvd ciphers. The State of the Art of Stream Ciphers: Workshop Record, Belgium, 2004, 45~59

2 Filiol E, Fontaine C. A new ultrafast stream ciphers design: COS ciphers. In: The 8th IMA Conference on Cryptography and Coding, LNCS, Springer-Verlag, 2001. 2260: 85~98

3 Wu H, Bao F. Cryptanalysis of stream cipher COS (2,128) mode I. In: Australian Conference on Information Security and Privacy-ACISP'2002. Springer-Verlag: LNCS, 2002. 2384: 154~158

4 Babbage S. The COS stream ciphers are extremely weak. http://eprint.iacr.org/2001/078/

5 Babbage S. Cryptanalysis of the COS (2,128) stream ciphers. http://eprint.iacr.org/2001/106/

6 Orumiehchi M, Mirghadri R. A distinguish attack on COSvd cipher. http://eprint.iacr.org/2005/164/

7 Henon M. A two-dimensional mapping with a strange attactor. Communications in Mathematical Physics, 1976, 50: 69~77

8 Goldberg D, Priest D. What every computer scientist should know about floating-point arithmetic. ACM Comp Surv, 1991, 23(1): 5~48[DOI]

9 Mantin I, Shamir A. A practical attack on broadcast RC4. In: Fast Software Encryption-FSE'2001. Springer-Verlag: LNCS, 2002. 2355: 152~164

10 Erdmann D, Murphy S. Henon stream cipher. Electronic Letters, 1992, 28(9): 893~895

附录 A

```
unsigned char Tabkey2[256][4] = {
{17,198,0,37}, {210,240,183,153}, {228,85,222,214}, {130,147,36,201},
{217,169,187,173}, {80,238,162,101}, {135,12,178,219}, {6,13,48,93},
{132,8,152,196}, {227,86,199,249}, {161,220,69,202}, {147,53,185,244},
{241,226,101,158}, {151,25,145,8}, {13,27,26,186}, {174,10,133,55},
{200,26,139,241}, {12,17,24,180}, {138,22,170,226}, {213,209,166,178},
{186,247,22,193}, {2,5,241,17}, {167,213,104,236}, {187,245,72,61},
{221,105,236,91}, {29,49,33,16}, {75,17,34,17}, {126,156,76,74},
{26,54,52,28}, {88,249,193,246}, {115,187,214,110}, {232,218,119,146},
{156,42,59,154}, {16,33,51,24}, {25,35,49,30}, {207,76,217,167},
{168,103,83,212}, {175,91,126,253}, {211,254,77,59}, {74,202,229,40},
{199,211,45,184}, {134,196,23,4}, {5,5,227,5}, {1,3,118,159},
```

{155,36,15,224}, {76,214,100,230}, {106,136,254,122}, {94,246,54,68},
{165,192,74,183}, {202,31,150,123}, {58,99,67,33}, {154,38,97,216},
{35,100,224,34}, {64,193,28,116}, {252,57,153,149}, {216,24,146,240},
{52,108,105,57}, {85,177,252,232}, {177,243,130,223}, {51,102,155,44},
{230,119,175,221}, {21,43,17,137}, {142,183,30,25}, {40,114,80,121},
{180,124,109,166}, {54,106,110,43}, {33,66,102,49}, {60,120,198,39},
{50,70,98,60}, {128,1,60,211}, {55,90,8,22}, {198,140,159,189},
{39,74,78,38}, {42,89,84,53}, {49,116,186,46}, {38,73,3,56},
{41,112,82,119}, {215,175,71,203}, {149,148,161,81}, {113,172,107,251},
{45,95,91,41}, {235,64,212,78}, {9,9,19,9}, {10,28,255,82},
{11,23,201,10}, {247,185,85,237}, {3,7,237,42}, {87,227,249,213},
{15,30,31,168}, {159,231,163,215}, {153,173,200,252,}, {231,94,203,139},
{212,16,253,245}, {239,215,32,58}, {188,237,108,136}, {14,20,29,85},
{61,122,125,54,}, {253,166,225,160}, {218,34,148,247}, {117,210,61,66},
{116,199,134,67}, {124,184,223,64}, {89,163,195,248}, {169,82,68,197},
{70,200,192,69}, {102,207,138,199}, {129,131,86,233}, {190,127,238,90},
{95,252,50,6}, {183,123,189,47}, {246,109,124,152}, {179,101,209,135},
{104,217,210,114}, {32,32,65,140}, {170,98,248,195}, {91,165,14,50},
{146,133,38,235}, {152,32,129,148}, {103,205,42,89}, {18,18,37,18},
{254,60,177,208}, {242,96,115,32}, {43,87,35,62}, {140,135,47,228},
{56,83,116,51}, {143,21,220,72}, {81,228,160,243}, {201,152,44,182},
{244,232,245,217}, {114,170,239,107}, {108,212,221,86}, {67,134,89,109},
{66,132,204,98}, {192,206,142,128}, {120,241,140,79}, {150,62,13,225},
{100,141,197,120}, {69,139,213,113}, {93,251,11,96}, {83,234,180,118},
{110,180,246,45}, {144,48,234,157}, {203,44,63,14}, {57,93,235,162},
{78,149,157,77}, {160,157,79,15}, {84,179,169,106}, {105,255,81,29},
{99,233,117,92}, {189,110,251,187}, {77,146,7,112}, {53,68,127,142},
{82,224,165,238}, {219,46,55,144}, {214,51,112,177}, {173,250,111,192},
{136,189,16,163}, {166,72,158,111}, {226,88,215,210}, {71,219,143,12},
{90,190,182,83}, {176,115,99,227}, {145,186,53,156}, {27,45,132,11},
{19,19,39,19}, {24,58,93,23}, {20,56,41,164}, {225,195,208,155},
{22,47,88,20}, {4,4,9,133}, {193,2,228,161}, {250,203,66,21},
{7,15,219,84}, {248,253,10,126}, {174,133,242,198}, {240,225,90,220},
{30,61,62,27}, {109,145,202,206}, {243,204,87,175}, {44,81,173,124},
{195,55,144,188}, {223,239,43,194}, {47,126,181,3}, {123,182,190,76},
{151,8,46,151}, {185,181,240,97}, {73,194,147,117}, {194,19,149,172},
{127,158,216,104}, {204,151,106,31}, {28,41,58,170}, {157,40,167,234},
{122,244,250,108}, {148,59,40,222}, {184,229,113,190}, {237,6,137,130},
{131,4,6,254}, {46,125,5,48}, {234,164,123,132}, {101,150,179,7},
{233,143,12,134}, {171,137,95,191}, {249,113,191,129}, {220,171,25,185},
{178,71,135,200}, {107,153,56,88}, {68,222,136,231}, {209,176,4,105},

{141,144,27,138}, {72,221,154,242}, {205,159,21,141}, {181,107,20,255},
{251,80,172,147}, {96,129,114,80}, {197,121,232,181}, {255,117,194,99},
{191,248,120,13}, {121,230,171,87}, {97,155,205,94}, {23,63,218,1},
{75,0,151,75}, {36,97,73,165}, {63,79,92,52}, {163,69,247,204},
{208,178,164,229}, {92,242,184,95}, {65,130,131,127}, {164,235,122,239},
{236,216,231,131}, {229,128,207,174}, {182,75,196,100}, {34,111,103,115},
{172,92,141,250}, {196,14,128,70}, {125,168,2,73}, {98,188,244,209},
{206,154,188,179}, {48,77,230,176}, {37,37,75,37}, {31,39,174,26},
{224,78,211,205}, {139,223,1,63}, {118,236,243,65}, {245,191,226,150},
{86,174,70,125}, {62,84,168,36}, {158,138,94,207}, {137,162,18,169},
{112,167,233,102}, {59,118,121,171}, {238,67,96,145}, {79,197,57,103},
{162,201,64,143}, {119,161,176,218}, {222,29,156,71}, {111,142,206,35}}