RC6 算法的分析及其改进

许万鹏，周炯超，朱公澳，江一川，邹凯蓄
（中国矿业大学计算机科学与技术学院，江苏徐州 221116）

摘 要： 适合在 IC 卡上实现的加密算法 RC6 进行了详细的分析，并在此基础上对 RC6 算法本身存在的缺陷进行了改进。所得结果解决了 IC 卡管理系统中的核心安全问题，即对 IC 卡中数据的加密问题具有一定的意义。

关键词： RC6 算法；分组密码；IC 卡
中图分类号： TP311 **文献标识码：** A **文章编号：** 0372-2112（2021）08-2502-07
电子学报 URL： http://www.ejournal.org.cn **DOI：** 10.12263/j.issn.0372-2112.2021.10.007

Analysis and Improvement of RC6 Algorithm

XU Wan-peng,ZHOU Jiong-chao,ZHU Gong-ao,Jiang Yi-chuan,Zou Kai-xu

(*School of Computer Science and Technology, China University of Mining and Technology, Xuzhou,Jiangsu 221116,China*)

Abstract: The encryption algorithm RC6 suitable for IC card is analyzed in detail, and the defects of RC6 algorithm are improved. The result solves the core security problem in IC card management system, that is, it is of certain significance to the data encryption in IC card.

Key words: RC6 algorithm;block cipher;IC card

1 引言

校园 IC 卡管理系统（简称校园卡系统）是一个多功能 IC 卡应用系统. 所谓“多功能”是指以一张 IC 卡同时支持多种不同的应用子系统，各个应用子系统本身是在统一规划的前提下分别开发的包括学生上机管理、食堂收费管理、图书馆管理、校内小额消费、学生选课、校医院医疗等. 这些管理项目用 IC 卡应用系统都能很好地实现. 随着我国对教育的加大投资及各大学校园网的建成投入使用为校园 IC 卡管理系统的建立提供了可能，校园内实现 IC 卡管理已成为校园管理发展的必然趋势[1]。

IC 卡管理系统中的安全问题，主要是 IC 卡中数据的安全问题. 对于 IC 卡的攻击者，其最主要的目的是攫取卡中的信息或者对卡进行非法复制，或者篡改卡中内容，或者冒用他人的卡等，以达到破坏系统的目的。IC 卡管理系统安全中的关键问题就是如何选取一个好的加密算法对 IC 卡内存放的数据进行加密，使得即使口令保护失效时攻击者也很难获取卡中信息[2]。

以下对适合在 IC 卡上实现的加密算法 RC6 进行了详细的分析，在此基础上对 RC6 算法本身存在的主要缺陷进行了改进，提出了一种新的加密算法——三重 RC6－I 加密算法。

2 诞生背景

在 1972-1974 年间 美国国家标准局公开征求对密码体制的联邦注册，这最终导致了 DES 的出现，它已成为世界上最广泛使用的密码体制。尽管针对 DES 的争论相当激烈，但对 DES 最中肯也是最致命的批评是其秘钥长度太短。事实上，当穷举秘钥成为可能时，DES 已走到了生命的尽头。

针对 DES 的缺点，人们提出了三重 DES 算法。尽管它在安全性上有很大提高 但是三 DES 太慢了而且它的明文和密文长度也太短不能适应将来的发展所以三重 DES 只能是一个过渡性算法。

这样 1997 年 NIST 公开征求 AES 作为 2001 年以后的数据加密标准。同时，NIST 通告了对 AES

收稿日期：2021年12月18日；修回日期：2022年1月4日；责任编辑：孙瑶
基金项目：国家自然科学基金(No.61802194,No.61902190)；江苏省高等学校自然科学研究(No.17KJB520015,No.19KJB520040)；审计信息与工程协同创新中心资助项目(No.18CICA06)

的几点要求如下[3,4]：

1）分组长是 128 比特，秘钥长是 128 比特，192 比特和 256 比特。

2）加密速度比三重 DES 快。

3）在 Intel Pentium Pro 和其他硬件及软件上均能高效运行。

4）算法设计的灵活性较好。如能接受其他的密码长度，能在各种平台上实现。

5）无 32 位处理器不易实现的操作。

6）算法的设计思想尽可能简单。

AES 的征集通告发出后，许多国家、企业和个人都提交了自己的方案。1998 年 8 月 AES 召开第一次候选会议，结果有 15 个算法入围。1999 年 3 月 AES 召开第二次候选会议，5 个算法入围。2001 年 AES 将从这 5 个算法中最后选出 1 个成为新的数据加密标准。RC6 算法就是其中的一个算法

3 算法实现

我们可以先用一张流程图简单地概括 RC6 算法。

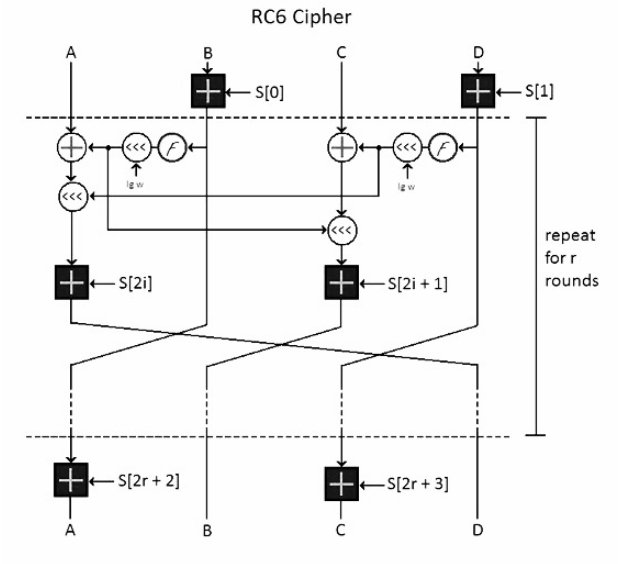

图 1 RC6 算法加密流程

接下来，对图中的各个部件进行说明。

首先，RC6 算法需要 4 个 32 位寄存器 A, B, C, D。

其次，RC6 算法需要四种基本运算，分别是：

1）整数模2^w加/减，记为$+/-$

2）整数模2^w乘，记为$\times$

3）按位模2加，记为$\oplus$

4）循环左移/循环右移，记作ROL/ROR

这四种基本运算中，前两种可以视为与运算，第三种实质上是异或运算，由此可见，算法的效率不会很低。

下面给出 RC6 算法的具体流程。

算法 1 加密

1: 把 128bit 明文放入 4 个 32bit 的寄存器 A、B、C、D 中
2: B = B + S[0] （S 为密钥）
3: D = D + S[1]
4: **FOR** i = 1 **TO** r **DO**
5: t = ROL(B * (2B+1), $log_2 w$)
6: u = ROL(D * (2D+1), $log_2 w$)
7: A = ROL(A ⊕t, u) + S[2i]
8: C = ROL(C ⊕u, t) + S[2i + 1]
9: (A, B, C, D) = (B, C, D, A)
10: **END FOR**
11: A = A + S[2r + 2]
12: C = C + S[2r + 3]
13: A, B, C, D 即为密文

算法 2 解密

1: 把 128bit 密文放入 4 个 32bit 的寄存器 A、B、C、D 中
2: C = C - S[2r + 3]
3: A = A - S[2r + 2]
4: **FOR** i = r **DOWN TO** 1 **DO**
5: (A, B, C, D) = (D, A, B, C)
6: u = ROL(D * (2D+1) , $log_2 w$)
7: t = ROL(B * (2B+1) , $log_2 w$)
8: C = ROR(C - S[2i + 1] , t) ⊕ u
9: A = ROR(A - S[2i], u) ⊕ t
10: **END FOR**
11: D = D - S[1]
12: B = B - S[0]
13: A, B, C, D 即为明文。

算法 3 密钥扩展

1: 首先，将种子密钥 K 输入到 c 个 w 比特字的 L[0],...,L[c-1]阵列，若密钥长度不够，用 0 字节填充，其中 c 为$\left\lceil \frac{8b}{w} \right\rceil$

2: P_w = 0xB7E15163

3: Q_w = 0x9E3779B9

4: S[0] = P

5: **FOR** i=1 **TO** 2r+3 **DO**

6: S[i] = S[i - 1] + Q

7: **END FOR**

8: 用户密钥混合到 S 中：

9: A = B = i = j = 0

10: v = 3 * max{c, 2r + 4}

11: **FOR** s=1 **TO** v **DO**

12: A = S[i] = ROL(S[i] + A + B, 3)

13: B = L[j] = ROL(L[j] + A + B, A + B)

14: i = (i + 1) mod(2r + 4)

15: j = (j + 1) mod c

16: **END FOR**

17: 输出 S[0],S[1],...,S[2r+3]即为子密钥

关于算法 3 中的魔法值P_w和Q_w。他们的定义为[5]

$$P_w = Odd(e-2)2^w$$

$$Q_w = Odd(\phi-2)2^w$$

其中，

$$e = 2.718281828459$$

$$\phi = 1.618033988749$$

$Odd(x)$表示大于等于$\lfloor x \rfloor$的最小奇数。

经计算得

$$P_{16} = 0xB7E1$$

$$Q_{16} = 0x9E37$$

$$P_{32} = 0xB7E15163$$

$$Q_{32} = 0x9E3779B9$$

代码实现详见附录。事实上，附录中的代码只是 RC6 的简单实现，读者可自行对其加入 base64 编码或创建寄存器类，重载几种基本运算。

4 安全性分析

首先我们知道分组密码的本质是混淆和扩散。混淆是指打乱密文、明文、密钥之间的依赖关系。扩散是指明文的统计特性消散在密文中，每个明文比特尽可能的影响多个密文，密文每个比特受多个明文比特影响。

作为 RC5 强化版的 RC6 通过引入乘法运算来决定循环移位次数的方法，对 RC5 进行改进，弥补了 RC5 在扩散速度上的不足，并且 RC6 中的非线性部分是由多个部件共同实现的，这都大大增强了 RC6 的安全性。

接着，我们考虑弱密钥的可能性。RC6 的扩展算法将用户密钥用一个伪随机过程加以扩展，尽管还没有证明两个密钥不会产生相同的循环密钥表，但是这种可能微乎其微。

当$w/r/b$三者为典型值，即$w=32$，$r=20$，$b=32$时，密钥为 256 位，两个 256 位的密钥产生一个相同的 44 个 32 位循环密钥的概率，根据密钥长度的全部可能性对应的集合大小进行比较运算，约为$\frac{2^{2*256}}{2^{44*32}} = 2^{2*256-44*32} = 2^{-896} \approx 10^{-270}$，可见概率很低。

至今 RC6 还未发现类似 DES 中的“弱密钥”。

最后，我们探讨一下常见密码破解方法对 RC6 算法的效果。

穷举攻击：穷举b字节的用户密钥或扩展密钥，密钥长度为2^{8b}位，当$w=32$，$r=20$，$b=32$时,扩展密钥长度为2^{8*4*44}位，即2^{1408}位，这种穷举法需要$min\{2^{8b}, 2^{1408}\}$次操作，可行性较低[1] 。

中间相遇攻击：RC6 多密钥重复加密时候，如果对其进行中间相遇攻击，则需要2^{700}次计算，这样要恢复扩展密钥最少需要$min\{2^{86}, 2^{704}\}$次操作[6] 。

差分攻击和线性攻击：对 RC6 的差分分析和线性分析只有在迭代轮数较少时有效，对 20 轮循环的 RC6，用线性分析法至少需要2^{155}个明文，用差分分析法至少需要2^{238}个明文[1] 。

时间攻击：因为 RC6 的加解密中只有移位和异或运算，运算过程比较固定，都与数据无关，这样可以有效地避免“时间攻击”。

5 改进算法

RC6 算法本身也存在一些缺陷，其中之一是非线性函数“f”。尽管f的设计被列举出一些优点，但f的比特扩散是单向的，扩散速度不够快。并且由于使用了乘法，f的平均计算量为$\frac{w}{2}$ =16个加法，因此非线性函数是运行速度的瓶颈。RC6 的另一个缺陷

是加解密算法差别很大。针对以上缺陷，对 RC6 进行如下一系列改进[7]。

改进一：不改变 RC6 的网络结构，仅将轮函数中的非线性函数f变为如下的g：

$$g(x)=I(I(I(I(x)+a)+a)+a).$$

其中$I(x)$为比特逆序置换，a为常数$a=a^0+2^2+2^4+\ldots+2^{n-2}$.

改进后，$g(x)$的计算量至多相当于 8 个加法，因此由图 1 RC6 算法加密流程可看出，该方法的计算量至多相当于原 RC6 算法的$\frac{14}{22}\approx 0.7$倍．该方法保留了 RC6 算法的所有简洁性和透明性. 由于$g(x)$是迭代生成的，故软件实现复杂度还可以进一步降低。$f(x)$是比特单向扩展的，而$g(x)$则是双向扩展的,平均扩展速度高于$g(x)$。

改进二： 基于 RC6，为了进一步提高 RC6 算法的强度，我们决定采用三重 DES 算法的思想，提出三重 RC6 算法。

三重 RC6 加密算法是 RC6 算法的强化形式之一，它对每一个 128 位分组明文用 3 个密钥进行加密来提高算法的强度。加/解密过程如下：

设用M表示 128 位明文，用C表示密文，$E_K(M)$表示用密钥k运行 RC6 加密算法加密明文M，$D_K(C)$表示用密钥k运行 RC6 解密算法解密密文C，则三重加密强化算法的加/解密过程可表示如下：

加密过程：$C=E_{k3}(D_{k2}(E_{k1}(M)))$

解密过程：$M=D_{k1}(E_{k2}(D_{k3}(C)))$

三重 RC6 进一步提高了 RC6 算法的强度，虽然时间上可能需要进行 3 次 RC6 运算，但是因为 RC6 本身就比 DES 算法快很多，随着 CPU 速度的不断提高。三重 RC6 的执行速度还是可以接受的。

6 CTF 竞赛中的 RC6

RC6 算法在 CTF 竞赛中出现的频率也并不低，譬如 RCTF2017 中的 CrackMe，我们阅读一下反编译后的代码。

```
v2 = (int *)(a1 + 4);
v14 = 4;
do
{
  *(v2 - 1) -= 0x7FAF076D;
  v2[1] += 0x642805B4;
  v3 = v2[2];
  result = v2[1];
  v5 = *v2;
  v6 = 0174;
  v15 = 0174;
  while ( 1 )
  {
    v7 = *(v2 - 1);
    v2[2] = result;
    *(v2 - 1) = v3;
    v13 = result;
    v12 = __ROL4__(result * (2 * result + 1), 8);
    *v2 = v7;
    v8 = v6 - 4;
    v9 = *(v2 - 1);
    v2[1] = v5;
    v10 = __ROL4__(v7 * (2 * v7 + 1), 8);
    *(v2 - 1) = v10 ^ __ROL4__(v9 - *(_DWORD *)(a2 + 4 * (v8 / 4)), 32 - (v12 & 0x1F));
    result = v12 ^ __ROL4__(v5 - *(_DWORD *)(a2 + 4 * (v15 >> 2)), 32 - (v10 & 0x1F));
    v6 = v15 - 8;
    v11 = (int)(v15 - 8) < 0;
    v2[1] = result;
    v15 -= 8;
    if ( v11 )
      break;
    v5 = v7;
    v3 = v13;
  }
  *v2 -= 0x5BF76637;
  v2[2] -= 0x4748DA7A;
  v2 += 4;
  --v14;
}
while ( v14 );
return result;
```

题目特征显然，首先可以观察到程序的两层循环结构，接着，程序对两个魔法值进行了运算，但该魔法值并不是典型值，略读至程序中部，发现了非线性函数f，至此可以确定此为修改后的 RC6 算法。

7 总结

本文从 RC6 算法在生活中的实际应用讲起，介绍了 RC6 算法的诞生背景、算法实现的流程图及伪代码部分、安全性分析、两种基于 RC6 的改进算法和 CTF 竞赛中的 RC6，最后在附录中给出 RC6 算法的 python 实现。

附录

rc6.py

```python
from math import e
from scipy.constants import golden
from math import ceil, log2

w = 32
r = 20
b = 16
MOD = pow(2, 32)


def ROL(x, s):
```

```
    x &= 0xFFFFFFFF  # F^8==2^32
    s %= w
    res = ((x << s | x >> (w - s))) & 0xFFFFFFFF
    return res


def ROR(x, s):
    return ROL(x, w - s)


def odd(x):
    x = int(x)
    while x % 2 == 0:
        x += 1
    return x



def extend_key(key):
    P_w = odd((e - 2) * pow(2, w))
    Q_w = odd((golden - 1) * pow(2, w))
    u = w // 8  # 1word=8byte
    c = ceil(b / u)

    k = [ord(ch) for ch in key] + [0] * (b - len(key))
    L = []
    for i in range(0, len(k), u):
        L.append((k[i] << 24) + (k[i + 1] << 16) + (k[i + 2] << 8) + k[i + 3])

    S = [0] * (2 * r + 4)
    S[0] = P_w
    for i in range(1, 2 * r + 3 + 1):
        S[i] = (S[i - 1] + Q_w) % MOD
    A = B = i = j = 0
    v = 3 * max(c, 2 * r + 4)
    while v := v - 1:
        A = S[i] = ROL(S[i] + A + B, 3)
        B = L[j] = ROL(L[j] + A + B, A + B)
        i = (i + 1) % (2 * r + 4)
        j = (j + 1) % (c)
    return S


def encrypt(plaintext, k):
    ciphertext = ''
    S = extend_key(k)
    bin_plaintext = ''.join([bin(ord(ch))[2:].zfill(8) for ch in plaintext])
    bin_plaintext += '0' * (128 - len(bin_plaintext) % 128)
    for l in range(0, len(bin_plaintext), 128):
        A = int(bin_plaintext[l + 00:l + 32], 2)
        B = int(bin_plaintext[l + 32:l + 64], 2)
        C = int(bin_plaintext[l + 64:l + 96], 2)
        D = int(bin_plaintext[l + 96:l + 128], 2)

        B = (B + S[0]) % MOD
        D = (D + S[1]) % MOD
        for i in range(1, r + 1):
            t = ROL(B * (2 * B + 1), int(log2(w)))
            u = ROL(D * (2 * D + 1), int(log2(w)))
            A = (ROL(A ^ t, u) + S[2 * i]) % MOD
            C = (ROL(C ^ u, t) + S[2 * i + 1]) % MOD
            (A, B, C, D) = (B, C, D, A)
        A = (A + S[2 * r + 2]) % MOD
        C = (C + S[2 * r + 3]) % MOD

        ciphertext += f'{A:032b}{B:032b}{C:032b}{D:032b}'
    res = hex(int(ciphertext, 2))[2:]
    print(res)


def decrypt(ciphertext, k):
    S = extend_key(k)
    plaintext = ''
```

```
    for l in range(0, len(ciphertext), 128):
        A = int(ciphertext[l + 00:l + 32], 2)
        B = int(ciphertext[l + 32:l + 64], 2)
        C = int(ciphertext[l + 64:l + 96], 2)
        D = int(ciphertext[l + 96:l + 128],
2)
        C = (C - S[2 * r + 3]) % MOD
        A = (A - S[2 * r + 2]) % MOD
        for i in range(r, 1 - 1, -1):
            (A, B, C, D) = (D, A, B, C)
            u = ROL(D * (2 * D + 1), 5)
            t = ROL(B * (2 * B + 1), 5)
            C = ROR(C - S[2 * i + 1], t) ^ u
            A = ROR(A - S[2 * i], u) ^ t
        D = (D - S[1]) % MOD
        B = (B - S[0]) % MOD
        plaintext += f'{A:032b}{B:032b}{C:032
b}{D:032b}'

    res = ''
    for i in range(0, len(plaintext), 8):
        res += chr(int(plaintext[i:i + 8],
2))
    print(res)


while True:
    type = input()
    if type == 'E':
        plaintext = input("请输入明文:\n")
        k = input("请输入密钥(小于32字节):\n
")
        encrypt(plaintext, k)
    elif type == 'D':
        ciphertext = input("请输入密文:\n")
        ciphertext = ''.join([f'{int(ch, 16):
04b}' for ch in ciphertext])
        k = input("请输入密钥(小于32字节):\n
")
        decrypt(ciphertext, k)
```

参考文献

[1] 王镭,陈克非.MARS 算法和 RC6 算法分析[J].计算机工程,2001(04):132-134.

[2] 杨南海,王秀坤.一种安全的 IC 卡数据加密算法及其应用[J].计算机工程与应用,2002(13):249-250+253.

[3] Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard[OL].https://csrc.nist.gov/News/1997/Announcing-Development-of-FIPS-for-Advanced-Encryp.

[4] Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard[OL].https://csrc.nist.gov/news/1997/requesting-candidate-algorithm-nominations-for-aes.

[5] Paje R E J, Sison A M, Medina R P. Multidimensional key RC6 algorithm[C]//Proceedings of the 3rd International Conference on Cryptography, Security and Privacy. 2019: 33-38.

[6] 何文才,牛晓蕾,刘培鹤,杜鹏,张媛媛.密码算法 RC5 和 RC6 的分析和比较[J].网络安全技术与应用,2007(02):28-30.

[7] 张晶,岳爱丽.基于 RC6 的改进加密算法[J].山东师范大学学报(自然科学版),2005(02):19-21.