# RC4算法的密码分析与改进

胡 亮, 迟 令, 袁 巍, 李宏图, 初剑锋
(吉林大学 计算机科学与技术学院, 长春 130012)

**摘要**: 针对RC4算法的安全隐患, 提出一种在RC4算法中增加自我检错步骤从而有效抵御错误引入攻击的改进算法, 该算法增加了字节变换, 可有效抵御状态猜测攻击. 对改进算法的安全性分析表明, 改进算法可有效抵御错误引入攻击和状态猜测攻击, 增加了RC4算法的安全性.

**关键词**: RC4算法; 错误引入攻击; 状态猜测攻击; 自我检错; 字节变换
**中图分类号**: TP309 **文献标志码**: A **文章编号**: 1671-5489(2012)03-0511-06

# Cryptanalysis and Improvements of RC4 Algorithm

HU Liang, CHI Ling, YUAN Wei, LI Hong-tu, CHU Jian-feng
(*College of Computer Science and Technology*, *Jilin University*, *Changchun* 130012, *China*)

**Abstract**: Based on security risks for RC4 algorithm, we presented some improvements. The improved algorithm can resist the fault induction attack via adding the self-error detection phase, and resist Knudsen's attack via adding the byte conversion phase. After analyzing the security of the improved algorithm, we proved that the improved algorithm can resist the fault induction attack and Knudsen's attack validly, the security of the RC4 algorithm is hence enhanced.

**Key words**: RC4 algorithm; fault induction attack; Knudsen's attack; self-error detection; byte conversion

RC4算法[1]是为了实现RSA数据的安全[2], 由Rivest设计的序列密码算法, 该算法广泛应用于多种数据传输和网络协议中, 如SSL/TLS和IEEE802.11WLAN标准. 但随着科技的进步, 该算法的安全隐患越来越多. Hoch等[3]提出了错误引入攻击, 该攻击的目标是通过在密钥加密的中间结果中引入"暂时随机"的字节错误, 利用错误的加密结果, 找出隐藏在密码设备的加密信息, 主要也是针对密钥生成阶段PGRA, 由于错误引入攻击可利用正确的和错误的加密结果进行分析比较, 因此也被称为查分错误分析; Knudsen等[4]提出的状态猜测攻击, 主要是针对密钥生成阶段PGRA实施攻击, 在获得部分初始状态的前提下, 通过对初始状态剩余字节赋值并与已经获得的密钥字符串比较, 逐步分析和判断, 最终得到完整的初始状态, 因此, 不需要种子密钥, RG4算法也可认为被攻破. 本文研究RC4算法的结构及状态猜测攻击和错误引入攻击的分析方法, 针对错误引入攻击在不影响算法主体本身的情况下加入了一个自我检错的步骤; 针对状态猜测攻击, 则通过对输出的部分字节与种子密钥相关联进行字节代换, 增加了状态猜测攻击的运算复杂度. 本文提出的两种改进方法并未改变RC4原有主体算法的结构, 只针对RC4算法的隐患进行了适当的改进, 增加了算法的安全性.

**收稿日期**: 2011-04-19.

**作者简介**: 胡 亮(1968—), 男, 汉族, 博士, 教授, 博士生导师, 从事网格计算与网络安全的研究, E-mail: hul@mail.jlu.edu.cn. 通讯作者: 初剑锋(1978—), 男, 汉族, 博士, 讲师, 从事计算机系统结构的研究, E-mail: aa11@21cn.com.

**基金项目**: 国家自然科学基金(批准号: 60873235; 60473099)、国家重点基础研究发展计划973项目基金(批准号: 2009CB320706)、教育部新世纪优秀人才支持计划项目基金(批准号: NCET-06-0300)、吉林省重点科研项目(批准号: 20080318)和吉林大学研究生创新项目(批准号: 20080224).

# 1 RC4 算法

RC4 算法由美国麻省理工学院的 Rivest[1] 开发，他也是 RSA 的开发者之一. RC4 算法是典型基于非线性数组变换的序列密码. 它以一个足够大的数组为基础，对其进行非线性变换，产生非线性的密钥序列，该大数组称为 S 盒. 实现 RC4 算法，需要两个处理过程：1）密钥调度算法(key-scheduling algorithm，KSA)：用于设置 S 盒的初始排列；2）伪随机生成算法(pseudo random-generation algorithm，PRGA)：用于选取随机元素，并修改 S 盒的原始排列顺序. RC4 算法如下.

RC4_Run($N$,$K$)：

KSA：$S_0[i]=i(i=0,1,\cdots,2^n-1)$，$i_0=0$，$j_0=0$；

$i_t=i_{t-1}+1$；

$j_t=j_{t-1}+S_{t-1}[i_{t-1}]+K[i_{t-1}\bmod l]$；

$S_t[i_t]=S_{t-1}[j_t]$，$S_t[j_t]=S_{t-1}[i_t]$，$t=1,2,\cdots,N-1$.

PRGA：$i_0=0$，$j_0=0$；

$i_t=i_{t-1}+1$；$j_t=j_{t-1}+S_{t-1}[i_t]$；

$S_t[i_t]=S_{t-1}[j_t]$，$S_t[j_t]=S_{t-1}[i_t]$；

$Z_t=S_t[S_t[i_t]+S_t[j_t]]$.

其中：$n$ 表示一个算法中使用的一个字节长度；$N$ 表示长度为 $n$ 的一个字节能显示值的总量，即 $N=2^n$；$i_t$，$j_t$ 表示两个参数；$K$ 表示种子密钥，$l$ 为其长度，$l=K$ 的比特数/$n$；$Z_t$ 表示 $t$ 时刻的输出值. 加密时，将 $Z_t$ 与明文异或；解密时，将 $Z_t$ 与密文异或.

# 2 攻击 RC4 算法分析

目前，研究者对 RC4 算法的分析主要集中于 RC4 算法本身的弱密钥性及部分字节的重复出现问题[5-7]，这些研究并不能对 RC4 算法的安全性构成威胁. 之后，Hoch 等[3] 针对序列密码的错误引入攻击方法，并对 RC4 算法进行分析，缩短了破解时间，可有效地攻击 RC4 算法.

## 2.1 错误引入攻击

假设攻击者控制了密码设备，可正确地使用密码设备进行加密，还可向密码设备中引入错误，影响加密过程，使密码设备输出错误的加密结果. 攻击者不知道隐藏在密码设备中的秘密信息，其目标是利用错误的加密结果找出秘密信息. 攻击者会同时利用正确的和错误的加密结果进行分析比较，从而揭示隐藏的秘密，这样的错误引入攻击也被称为差分错误分析. 衡量这些分析方法的参数是攻击过程中所需的密钥字和引入的错误次数. 分析方法步骤如下[8-9].

Normal_Run：正常运行 RC4 算法一次，记录足够多的输出密钥字；

Fault_Run $T(t)$：重新运行 RC4 算法，在 $i_t$ 和 $j_t$ 更新为 $i_{t+1}$ 和 $j_{t+1}$，且 Swap($S_t[i_{t+1}]$,$S_t[j_{t+1}]$)未执行前，向 S 盒的第 $t$ 个位置引入错误，记录第一个错误的输出时刻 $T'$；

Fault_Run $T(t,n)$：重新运行 RC4 算法，在 $i_{t-1}$ 和 $j_{t-1}$ 更新为 $i_t$ 和 $j_t$，且 Swap($S_{t-1}[i_t]$,$S_{t-1}[j_t]$)未执行前，向 S 盒的第 $n$ 个位置引入错误，记引入错误后得到的第 $t$ 个输出密钥字为 $Z_t'$ 和正确的第 $t$ 个输出密钥字为 $Z_t$；

Judge：通过判断错误引入攻击后产生的错误密钥字类型(type_A 和 type_B)，即可成功地找出初始状态值. 算法描述如下：

1）Normal_Run；

2）令 $t=1$；$J=0$；

3）对每个 $i\in[0,255]$，令 is_det$[i]=0$；

4）Fault_Run $T(t)$；

5）令 $n=0$；

6）if ($n=t$) then $n=n+1$，goto 7)；else goto 8)；

7）Fault_Run $T(n)$；

8）if（$Z_t = Z_t'$）then｛if（$n<256$）then｛$n=n+1$；goto 6）｝ else goto 9）｝；else

① 继续运行 7），记录下一个错误的输出时刻 $T$；

② if（$T=T'$）then ｛temp1 =（$n-J$）；$J=n$；type_A = 1；if（is_det[$t$] = 0）then $S_0[t]$ =（$n-J$）｝；

③ if（$T\neq T'$）then ｛temp2 = $n$；type_B = 1；if（is_det[$n$] = 0）then $S_0[n]=Z_t$｝；

④ if（type_B = 1&&type_A = 1&&is_det[$J$] = 0）then $S_0[J]$ =（temp2 − temp1）；goto 9）｝；

⑤ if（type_B = 0&&type_A = 1&&is_det[$J$] = 0&&$n$ = 256）

｛if（temp1 = $Z_t$）then ｛$S_0[J]$ =（$J$ − temp1）；goto 9）｝ else｛$S_0[J]=Z_t$；goto 9）｝｝；

⑥ if（type_B = 1&&typeA = 0&&is_det[$t$] = 0&&$n$ = 256）then ｛$S_0[t]=(t-J)$；goto 9）｝；

⑦ if（$n<256$）then ｛$n=n+1$；goto 6）｝；

9）if（$t<256$）then ｛$t=t+1$；goto 4）｝ else END.

Hoch 等[3]对 RC4 实施错误引入攻击的结果表明，需要 $2^{26}$个密钥字和 $2^{16}$次错误引入可恢复 RC4 的整个初始状态. Biham 等[10]在研究了 RC4 的错误引入分析攻击后，得到了需要 $2^{16}$个密钥字和 $2^{16}$次错误引入可以恢复 RC4 的整个初始状态结果.

### 2.2　Knudsen 的状态猜测攻击

该状态猜测攻击方法是针对 RC4 算法中的 PRGA，由 Knudsen 等[4]提出. 基本思想是假设攻击者可获得一段足够多的 PRGA 输出值，如果攻击者能根据这些输出值计算出 PRGA 的初始状态，则不需要密钥即可继续产生任何输出值，RC4 算法被成功破解. 攻击者通过对内部状态中未知位置赋值的方法，可根据已知的输出值和其他信息判断是否出现矛盾，如果出现了矛盾，则说明在前面的赋值过程中出现了错误，攻击者将指针返回赋值过程重新赋值；如果整个赋值过程中不出现矛盾，则说明所有的赋值都是正确的，攻击者即得到了一个正确的内部状态. 算法步骤如下.

1）检查 $S_{t-1}[i_t]$是否已被赋值过：

① 如果有值，转 2）；

② 如果没有值，用 $2^n-a_t$ 个未出现的值对 $S_t[i_t]$赋值，更新 $a_t$ 并得到 $j_t=j_{t-1}+S_{t-1}[i_t]$，然后转 2）；

2）检查 $Z_t$ 是否等于 $a_t$ 中某个值，即是否在内部状态已知值之中：

① 若等于，计算 $S_{t-1}[j_t]=S_t^{-1}[Z_t]-S_{t-1}[j_t]$，检查 $S_{t-1}[j_t]$是否已被赋值过：如果有，则观察它是否等于刚计算出的 $S_{t-1}[j_t]$：若等于，则令 $t+1$，转 1），继续；若不等于，则出现矛盾，返回 1）中的②，重新对 $S_{t-1}[j_t]$赋值；如果没有，则观察刚计算出的 $S_{t-1}[j_t]$是否在 $a_t$ 中：若不在，则用刚计算出的 $S_{t-1}[j_t]$赋值，更新 $a_t$，返回 1）；若在，则出现矛盾，返回 1）中的②，重新对$S_{t-1}[i_t]$赋值；

② 若不等于，则转 3）.

3）检查 $S_{t-1}[j_t]$是否已经被赋值过：

① 如果没有，用 $2^n-a_t$ 个未出现的值对 $S_{t-1}[j_t]$赋值，更新 $a_t$，计算 $i_t=S_{t-1}[i_t]+S_{t-1}[j_t]$，观察 $S_t[i_t]$是否被赋值：如果有值，则判断 $S_t[i_t]$是否等于 $Z_t$：如果相等，则令 $t+1$，返回 1）；如果不相等，则出现矛盾后，重新对 $S_{t-1}[j_t]$赋值；如果没有值，则观察 $Z_t$ 是否在 $a_t$ 中：如果在，则出现矛盾，重新对 $S_{t-1}[j_t]$赋值；如果不在，则把 $Z_t$ 赋值给 $S_t[i_t]$，更新 $a_t$，令 $t+1$，返回 1）；如果用 $2^n-a_t$ 中的所有值对 $S_{t-1}[j_t]$赋值均得到矛盾，则返回 1）中的①，重新对 $S_{t-1}[i_t]$赋值；

② 如果有，计算 $i_t=S_{t-1}[i_t]+S_{t-1}[j_t]$，观察 $S_t[i_t]$是否被赋值：如果没有，则把 $Z_t$ 赋值给 $S_t[i_t]$，更新 $a_t$，令 $t+1$，返回 1）；如果有值，则出现矛盾后，返回 1）中的②，重新对 $S_{t-1}[i_t]$赋值.

其中：$S_t^{-1}[Z_t]$表示输出值 $Z_t$ 在内部状态 $S$ 中对应的位置；$j_t$，$S_t[i_t]$，$S_t[j_t]$分别为 $t$ 时刻 $j$，$S[i]$，$S[j]$的值；$a_t$ 为已猜测出的字节. 根据该攻击方法，本文得到了 $n=8$ 时随机状态的复杂度[9]，列于表 1. 由表 1 可见，获得已知初始状态的值越少，复杂度越高. 当完全未知初始状态中值的情形

下，其复杂度为 $2^{779}$.

**表1 $n=8$ 时随机状态的复杂度**

**Table 1 Complexity of the random state when $n=8$**

| 已知值数量 | 随机状态复杂度 | 已知值数量 | 随机状态复杂度 | 已知值数量 | 随机状态复杂度 |
|---|---|---|---|---|---|
| ≤142 | $\geqslant 2^{152.14}$ | 163 | $2^{90.92}$ | 184 | $2^{42.94}$ |
| 143 | $2^{148.94}$ | 164 | $2^{88.31}$ | 185 | $2^{41.01}$ |
| 144 | $2^{145.79}$ | 165 | $2^{85.73}$ | 186 | $2^{39.15}$ |
| 145 | $2^{142.67}$ | 166 | $2^{83.19}$ | 187 | $2^{37.33}$ |
| 146 | $2^{139.57}$ | 167 | $2^{80.67}$ | 188 | $2^{35.56}$ |
| 147 | $2^{136.50}$ | 168 | $2^{78.48}$ | 189 | $2^{33.82}$ |
| 148 | $2^{133.45}$ | 169 | $2^{75.72}$ | 190 | $2^{32.12}$ |
| 149 | $2^{130.43}$ | 170 | $2^{73.30}$ | 191 | $2^{30.47}$ |
| 150 | $2^{127.43}$ | 171 | $2^{70.90}$ | 192 | $2^{28.85}$ |
| 151 | $2^{124.16}$ | 172 | $2^{68.54}$ | 193 | $2^{27.28}$ |
| 152 | $2^{121.31}$ | 173 | $2^{66.22}$ | 194 | $2^{25.76}$ |
| 153 | $2^{118.60}$ | 174 | $2^{63.92}$ | 195 | $2^{24.28}$ |
| 154 | $2^{115.70}$ | 175 | $2^{61.66}$ | 196 | $2^{22.85}$ |
| 155 | $2^{112.84}$ | 176 | $2^{59.44}$ | 197 | $2^{21.46}$ |
| 156 | $2^{110.00}$ | 177 | $2^{57.25}$ | 198 | $2^{20.12}$ |
| 157 | $2^{107.19}$ | 178 | $2^{55.09}$ | 199 | $2^{18.82}$ |
| 158 | $2^{104.41}$ | 179 | $2^{52.97}$ | 200 | $2^{17.68}$ |
| 159 | $2^{101.65}$ | 180 | $2^{50.88}$ | 201 | $2^{16.39}$ |
| 160 | $2^{98.93}$ | 181 | $2^{48.84}$ | 202 | $2^{15.24}$ |
| 161 | $2^{96.23}$ | 182 | $2^{46.82}$ | 203 | $2^{14.13}$ |
| 162 | $2^{93.56}$ | 183 | $2^{44.85}$ | 204 | $2^{13.11}$ |

## 3 对 RC4 算法的改进

由于错误引入攻击需要在 RC4 算法过程中通过引入“暂时随机”的错误，再利用错误的与正确的密钥输出序列对比，通过对二者对比结果的判断，最终得到部分初始状态字节. 针对其中的引入“暂时随机”错误，可进行如下改进.

**算法1** PRGA.

1) $i_0=0$, $j_0=0$;

2) $i_t=i_{t-1}+1$; $j_t=j_{t-1}+S_{t-1}[i_t]$;

3) 令 $A_{t-1}=S_{t-1}[i_t]-S_{t-1}[j_t]$，且记此时的总状态序列为 $S_{t-1}$;

4) $S_t[i_t]=S_{t-1}[j_t]$, $S_t[j_t]=S_{t-1}[i_t]$;

5) 令 $A_t=S_t[j_t]-S_t[i_t]$，且记此时的总状态序列为 $S_t$;

6) 将 $S_{t-1}$ 与 $S_t$ 按对应字节依次相减，count($S_{t-1}[i_t]-S_t[i_t]\neq 0$);

7) if (count($S_{t-1}[i_t]-S_t[i_t]\neq 0$)$=3$), then close RC4_Run;

8) if (count($S_{t-1}[i_t]-S_t[i_t]\neq 0$)$=2$), then {if ($A_t\neq B_t$) then close $\mathrm{RC4_{Run}}$; if ($A_t=B_t$) then goto 9)};

9) $Z_t=S_t[S_t[i_t]+S_t[j_t]]$.

由于攻击者控制了密码设备，并且在 $i_{t-1}$ 和 $j_{t-1}$ 更新为 $i_t$ 和 $j_t$ 后，Swap($S_{t-1}[i_t]$,$S_{t-1}[j_t]$)未执行前，在其中引入一个错误的字节得到新的密钥序列，并与正常运行 RC4 算法得到的密钥序列进行对比，通过判断错误产生的方式和类型，逐步得到正确的 RC4 算法的初始状态.

本文在运算过程中添加一个自我检查的步骤以防止错误引入攻击的实施. 主要思想是通过对比交换前后状态 $S_t$ 的变化判断 RC4 算法运行期间是否有攻击者实施了错误引入攻击. 如果通过自我检查步骤的判断确定了 RC4 算法运行期间存在攻击者实施的错误引入攻击，则终止 RC4 算法的运行；如果通过自我检查步骤的判断确定一切运行正常，没有攻击者实施错误引入攻击，则继续运行 RC4 算法生

成密钥序列. 算法描述如下：

1）在 $i_{t-1}$和 $j_{t-1}$更新为 $i_t$ 和 $j_t$ 后，计算 $i_{t-1}$和 $j_{t-1}$两个指针对应的 $S_{t-1}[i_t]$和 $S_{t-1}[j_t]$差值，记为 $A_{t-1}$，总状态记为 $S_{t-1}$；

2）在交换步骤 Swap($S_{t-1}[i_t]$,$S_{t-1}[j_t]$)执行后，计算 $S_t[i_t]$和 $S_t[j_t]$的差值，记为 $A_t$，总状态记为 $S_t$；

3）将 $S_{t-1}$与 $S_t$ 按对应节依次相减，计算其中($S_{t-1}[i_t]-S_t[i_t]\neq 0$)的数目；

4）如果检查出的数目为 3，则终止算法；

5）如果检查出的数目为 2，则判断 $A_t$ 与 $A_{t-1}$是否相等，不相等则终止算法.

由于交换后 $S_t[i_t]=S_{t-1}[j_t]$，$S_t[j_t]=S_{t-1}[i_t]$，所以步骤 6）中对应的字节 $S_{t-1}[i_t]$与 $S_t[i_t]$的差值即为 $S_t[i_t]$和 $S_t[j_t]$的差值或 $S_{t-1}[i_t]$和 $S_{t-1}[j_t]$的差值. 执行步骤 6）后，只是对 $S_{t-1}[i_t]$和 $S_{t-1}[j_t]$进行了交换，其他位置的值并未改变，所以对 $S_{t-1}$与 $S_t$ 按字节进行相减，除了对换位置的两对值不为 0 外，其余所有字节的值都应等于 0. 因此，在引入错误后会有两种情况出现：

1）执行步骤 6）后有 3 个值不等于 0，则说明攻击者引入错误了，且引入错误的位置不是交换位置的 $S_{t-1}[i_t]$和 $S_{t-1}[j_t]$二者之一，由于自检步骤的存在，算法 1 会终止运行；

2）攻击者引入的错误正好是交换位置的 $S_{t-1}[i_t]$和 $S_{t-1}[j_t]$两者中之一，步骤 3）中记录了差值 $A_{t-1}$和总状态 $S_{t-1}$，且引入错误是在交换步骤 Swap($S_{t-1}[i_t]$,$S_{t-1}[j_t]$)执行后，即是步骤 3）执行后，记录了差值 $A_t$ 和此时的总状态 $S_t$. 如果正常运行 RC4 算法，则两个差值 $A_{t-1}$和 $A_t$ 的值相等. 但攻击者在其中引入了错误的字节，导致两个差值必然不相等，根据自检步骤，算法 1 会终止运行.

状态猜测攻击只针对 PGRA 的攻击，攻击者首先需要获得足够多的输出值，并已对初始状态的部分字节赋值，才能通过分析及对剩余的字节赋值得到整个初始状态. 其中的关键是对初始状态剩余字节的赋值，在得到新密钥输出序列后与已获得正确的密钥输出序列对比，然后分析直到赋值正确. 因此，可在输出密钥字节序列的环节进行字节代换，使攻击者的分析复杂化以增加运算复杂度. 于是，在算法 1 的基础上再做如下改进，称为算法 2.

假设 $n=8$，种子密钥为 128 位，即 16 字节，按字节对 $2^8=256$ 取 mod，分别记为 $k_0,k_1,\cdots,k_{15}$.

**算法 2**　PRGA.

1）$i_0=0$，$j_0=0$；

2）$i_t=i_{t-1}+1$；$j_t=j_{t-1}+S_{t-1}[i_t]$；

3）令 $A_{t-1}=S_{t-1}[i_t]-S_{t-1}[j_t]$，且记此时的总状态序列为 $S_{t-1}$；

4）$S_t[i_t]=S_{t-1}[j_t]$，$S_t[j_t]=S_{t-1}[i_t]$；

5）令 $A_t=S_t[j_t]-S_t[i_t]$，且记此时的总状态序列为 $S_t$；

6）将 $S_{t-1}$与 $S_t$ 按对应字节依次相减，count($S_{t-1}[i_t]-S_t[i_t]\neq 0$)；

7）if (count($S_{t-1}[i_t]-S_t[i_t]\neq 0$)=3), then close RC4_Rum；

8）if (count($S_{t-1}[i_t]-S_t[i_t]\neq 0$)=2), then {if ($A_t\neq B_t$) then close $RC4_{Rum}$; if ($A_t=B_t$) then goto 9)}；

9）if (($S_t[i_t]+S_t[j_t]$) mod 256$\neq k_m$&&($S_t[i_t]+S_t[j_t]$) mod 256$\neq 255-k_m$，其中 $m=0,1,\cdots,15$)，then {$Z_t=S_t[S_t[i_t]+S_t[j_t]]$}；

10）if (($S_t[i_t]+S_t[j_t]$) mod 256 $=k_m$ ‖ ($S_t[i_t]+S_t[j_t]$) mod 256 $=255-k_m$，其中 $m=0,1,\cdots,15$)，then {$Z_t=\{255-S_t[S_t[i_t]+S_t[j_t]]\}$ mod 256}.

状态猜测攻击主要针对 PGRA 阶段，并且前提条件为获得足够多的密钥输出字节及已赋值正确的部分初始状态字节. 于是，可以对初始状态中的部分字节只简单地进行依次对 256 取模并与 255 做差，而字节的选取则根据通信双方协商约定的种子密钥 $k_m$. 如果输出值对 256 取模后与 $k_m$ 不相等，则继续运行 RC4 算法，即运行步骤 9），正常输出密钥序列；如果输出值等于 $k_m$ 或 $255-k_m$，则运行步骤 10），进行字节代换运算.

## 4　安全性分析

由于本文的两个改进算法并未对 RC4 算法的主体进行改动，只针对两个对 RC4 算法有效地攻击方法

进行改进，所以改进后的RC4算法安全性依然依赖于其自身所具有的安全性优势.

对于错误引入攻击，由于其攻击形式的特殊性，算法1只增加了自我检错功能，当攻击者进行错误引入攻击时，攻击者只控制了密码设备，并不能修改算法程序，所以算法会因为自我检错语句的运行而终止，从而有效防止了错误引入攻击. 但也要考虑到改进算法本身会不会出现错误导致RC4算法在正常运行的情况下因自我检错步骤而终止. 算法1在原RC4算法的基础上首先将$i_{t-1}$和$j_{t-1}$前后的差值存储在另外的寄存器中，然后通过输出密钥字节序列前的判断步骤，对比$i_{t-1}$和$j_{t-1}$更新前后状态的差异，判断RC4算法运行过程中是否存在错误引入攻击所引入的“暂时随机”错误字节. 改进算法并未对原RC4算法结构、语句及任何字节进行改变，因此不存在改进算法会在RC4算法正常运行中出现异常字节导致RC4算法终止的情况.

对于状态猜测攻击，由于其前提条件的限定，使用了算法2的改进. 算法2的改进是在算法1的基础上继续针对状态猜测攻击的关键步骤进行改进，它复杂化了密钥字节序列输出步骤. 由于存在字节代换步骤，攻击者若想获得全部的初始状态，则必须知道究竟是哪16个字节进行了运算(另外16个字节和它们直接相关，所以不包括在内)，而这16个字节的选取却完全取决于密码通信双方协商约定的种子密钥，攻击者不可能获知. 而对进行字节变换运算的模与$k_m$和$(255-k_m)$相同的字节，它们的输出概率相差无几，所以在进行字节变换运算后并不影响RC4算法输出密钥序列的随机性. 这种改进方法增加了攻击者进行状态猜测攻击的复杂度. 根据状态猜测攻击的分析原理，在分析前，攻击者必须知道哪16个字节进行了变换，然后才能对具体字节进行赋值. 这样，运算复杂度就增加了$C_{256}^{16}$，大于$2^{83}$的复杂度，即使是赋值了203个字节，最终依然有$2^{96}$的复杂度. 如果选取更大的种子密钥序列，则攻击者计算的复杂度也会相应增加. 算法2改进了密钥字节序列输出阶段，对密钥字节先进行判断再输出，并未对算法本身任何时刻下状态中的字节做改变，因此，它不会因为出现意外的错误字节导致算法1的运行使算法终止. 根据种子密钥中字节进行字节代换运算的字节只是与255做差，依然是原状态中存在的字节，所以并不会使输出产生原状态没有的异常字节，从而导致算法出现漏洞.

## 参 考 文 献


[1] Rivest R L. The RC4 Encryption Algorithm [J/OL]. [2011-06-23]. http://www.vocal.com/data.sheets/RC4.pdf.

[2] Forouzan B A. Cryptography and Network Security [M]. New York: McFraw-Hill, 2008: 219-222.

[3] Hoch J J, Shamir A. Fault Analysis of Stream Ciphers [C]//Lecture Notes in Computer Science. Berlin: Springer-Velag, 2004: 240-253.

[4] Knudsen L R, Meier W, Preneel B, et al. Analysis Methods for (Alleged) RC4 [C]//Asiacrypt'98. Berlin: Springer-Velag, 1998: 327-341.

[5] CHEN Jia-geng, Miyaji A. Generalized RC4 Key Collisions and Hash Collisions [C]//Lecture Notes in Computer Science. Berlin: Springer-Velag, 2010: 73-87.

[6] CHEN Jia-geng, Miyaji A. A New Class of RC4 Colliding Key Pairs with Greater Hamming Distance [C]//Lecture Notes in Computer Science. Berlin: Springer-Velag, 2010: 30-44.

[7] Sichani M H, Movaghar A. A New Analysis of RC4 a Data Mining Approach (J48) [C]//Proceedings of the International Conference on Security and Cryptography. Setubal, Portugal: Insticc Press, 2009: 213-218.

[8] DU Yu-song, SHEN Jing. Research on Fault Induction Attack on RC4 Algorithm [J]. Journal of University of Electronic Science and Technology of China, 2009, 38(2): 253-257. (杜育松, 沈静. 对RC4算法的错误引入攻击研究 [J]. 电子科技大学学报, 2009, 38(2): 253-257.)

[9] Matsui M. Key Collisions of the RC4 Stream Cipher [C]//Lecture Notes in Computer Science. Berlin: Springer-Velag, 2009: 38-50.

[10] Biham E, Carmeli Y. Efficient Reconstruction of RC4 Keys from Internal States [J]. Lecture Notes in Computer Science, 2008, 5086: 270-288.


(责任编辑：韩 啸)