基于状态位索引方法的小状态流密码算法Draco-F

张润莲* 范 欣 赵 昊 武小年 韦永壮
(桂林电子科技大学计算机与信息安全学院 桂林 541000)

摘 要：Draco算法是首次基于初始向量和密钥前缀组合(CIVK)方案构造的一个流密码设计实例，其声称对于时空数据折中(TMDTO)攻击具有完全可证明的安全性。但因Draco算法的选择函数存在周期小的结构缺陷，攻击者给出了突破其安全界限的分析结果。针对Draco算法存在的安全缺陷等问题，该文提出一种基于状态位索引和动态初始化的改进算法Draco-F算法。首先，Draco-F算法通过使用状态位索引的方法增加了选择函数的周期并降低硬件成本；其次，在保障非线性反馈移位寄存器(NFSR)状态位使用均匀性的前提下，Draco-F算法通过简化输出函数进一步降低算法的硬件成本；最后，Draco-F算法引入动态初始化技术以防止密钥回溯。对Draco-F算法的安全性分析和软硬件测试结果表明：相对于Draco算法，Draco-F算法避免了Draco算法的安全漏洞，可以以128 bit的实际内部状态提供128 bit的安全级别；同时，Draco-F算法具有更高的密钥流吞吐率和更小的电路面积。

关键词：流密码；初始向量和密钥前缀组合；Draco；状态位索引；动态初始化
中图分类号：TN918.1 **文献标识码**：A **文章编号**：1009-5896(2025)01-0271-08
DOI: 10.11999/JEIT240524

1 引言

经典的流密码算法包括Grain算法[1]、SNOW算法[2]、Fruit算法[3]、Welch-Gong (WG)流密码[4]、祖冲之密码(Zhu Chongzhi Cipher, ZUC)算法[5]等。随着物联网和移动设备的普及，资源受限环境下的安全通信需求日益增长，轻量级流密码应运而生。但轻量级流密码算法容易受到时空数据折中(Time Memory Data TradeOff, TMDTO)[6]攻击的影响，使得算法的安全级别降低到其内部状态大小的1/2。为避免TMDTO攻击影响，常用方法是将密码的内部状态至少设计为所需安全级别的两倍。而在资源有限的设备中，较大的内部状态会增加硬件成本。

为解决这一矛盾，在2015年快速软件加密会议(Fast Software Encryption 2015, FSE 2015)中，小状态流密码算法[7]设计思想被提出，即将存储在非易失性存储器(Non-Volatile Memory, NVM)中的密钥或初始值作为算法内部状态的一部分，参与初始化和密钥流生成全过程，减小算法运行中使用寄存器的数量，降低硬件成本。基于该思想，一系列轻量级流密码算法被提出，包括Sprout[7], Fruit-80[3],Lizard[8],Plantlet[9],Atom[10]和Draco[11]等，这些算法整体结构高度相似，都采用线性或者非线性反馈移位寄存器(Linear/Nonlinear Feedback Shift Register, LFSR/NFSR)的串联结构进行驱动。

目前，小状态流密码有4种设计方案：(1)Lizard方案；(2)持续密钥(Continuous KEY, CKEY)方案；(3)持续向量 (Continuous Initial Value, CIV)方案；(4)初始向量和密钥前缀组合(Consisting of the Initial Value and Key-prefix, CIVK)方案。但Lizard算法针对不同攻击的安全性仍然处于生日门槛[12]；CKEY方案在状态更新期间持续使用存储在NVM中的密钥，使得基于该方案设计的密码算法在应对各种状态恢复攻击时难以维持密钥长度位的安全级别[13]；CIV方案不使用初始密钥，而是使用初始值作为非易失性部分，但目前还没有该类算法[14]；CIVK方案使用密钥前缀和初始值作为非易失性部分，由Hamann等人[11]于2022年提出，以设计的Draco算法为算法实例，并通过随机预言模型证明了使用该方案的流密码算法对通用TMDTO攻击的可证明安全性。此外，2023年，Gül等人[15]介绍了一种新的置换网络结构扩散层的构造方法，通过固定伪随机排列定义新的流密码操作模式，以新模式和扩散层结构设计小内部状态流密码，降低硬件开销。

Draco是Hamann等人[11]给出的一个基于CIVK方案的算法实例，声称对于TMDTO攻击具有完全可证明的安全性。但Draco算法的选择函数存在周期较小(仅为3 104)的问题，针对该缺陷，2022年，Banik[16]提出了两种对Draco算法的TMDTO攻击，成功地在设计者声明的安全界限内恢复了其内部状

收稿日期：2024-06-25；改回日期：2024-09-12；网络出版：2024-09-19
*通信作者： 张润莲 zhangrl@guet.edu.cn
基金项目：国家自然科学基金(62062026)，广西重点研发计划(桂科AB23026131)，广西研究生教育创新计划(YCSW2024347)
Foundation Items: The National Natural Science Foundation of China (62062026), The Key Research and Development Program of Guangxi (guike AB23026131), The Innovation Project of Guangxi Graduate Education (YCSW2024347)

态，并进一步通过该内部状态回溯得到了密钥值。此外，Draco算法还存在硬件实现成本高的问题。

为解决Draco算法存在的上述问题，本文提出一种改进的基于状态位索引的小状态流密码算法Draco-F。针对Draco算法存在的选择函数周期小且硬件成本高的问题，Draco-F算法通过NFSR状态位索引方式设计新的硬件成本更低的选择函数；针对Draco算法内部状态可逆的问题，引入动态初始化技术防止密钥回溯；同时，改进算法进一步简化Draco算法的输出函数，降低算法硬件开销。

2 Draco算法介绍

Draco[11]包括1个非易失性存储器、1个选择函数、2个NFSR和1个输出函数，如图1所示。

2.1 算法部件

(1)非易失性存储器和选择函数。除128 bit易失性内部状态外，Draco还采用129 bit非易失性内部状态，由96 bit公共初始值(**IV**)、128 bit密钥的前32 bit(即$K_0,K_1,\cdots,K_{31}$)和1个常数位0组成。在时钟周期t中，Key-IV调度位(KIS位)d_t计算为

$$d_t=\begin{cases}x_{t \bmod 97}, & 0\le t<256\\ K_{t \bmod 32}+x_{t \bmod 97}, & t\ge 256\end{cases} \tag{1}$$

其中，$x_0=0,x_i=\mathrm{IV}_{i-1},i=1,2,\cdots,96$，$K_i=k_i$，$i=0,1,\cdots,31$。如图1所示，每个时钟周期计算出的$d_t$将被馈送到NFSR2。

(2)NFSR1。Draco算法的NFSR1级数为33，t时刻其状态记作$\boldsymbol{S}_t=[s_0^t,s_1^t,\cdots,s_i^t,\cdots,s_{32}^t]$，其中$s_i^t$表示$t$时刻NFSR1的第$i$个状态位。考虑到密码易失性内部状态的大小减半(Draco为128 bit, Grain-128AEAD为256 bit)带来的安全威胁，Draco算法采用了33 bit宽的NFSR1取代Grain系列的LFSR，以加强针对代数和快速相关攻击的抵抗力。ACHTERBAHN-128/80的设计者提供了13个通过实验方法找到的具有最大周期的非线性反馈函数的列表，大小范围从21 bit(NFSR A0)到33 bit(NFSR A12)[17]，Draco算法的NFSR1选用了其中33 bit的NFSR A12，并通过添加$\overline{s_0^t}\,\overline{s_1^t}\cdots\overline{s_{32}^t}$项以避免其陷入全0状态，从而使得该NFSR真正达到最大周期2^{33}。其更新函数为

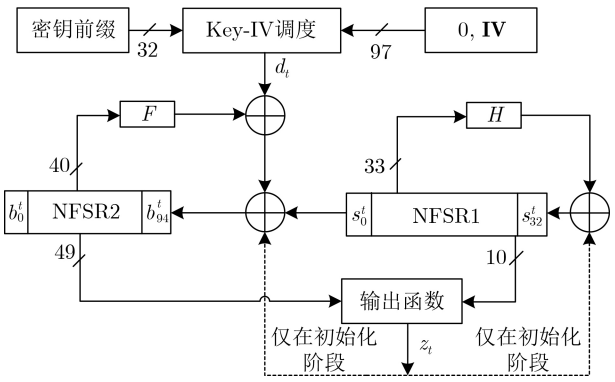

图 1 Draco算法结构图

$$\begin{aligned}s_{32}^{t+1}=&s_0^t+s_2^t+s_7^t+s_9^t+s_{10}^t+s_{15}^t+s_{23}^t+s_{25}^t+s_{30}^t\\&+s_8^ts_{15}^t+s_{12}^ts_{16}^t+s_{13}^ts_{15}^t+s_{13}^ts_{25}^t+s_8^ts_{13}^t\\&+s_{13}^ts_{15}^t+s_{13}^ts_{25}^t+s_1^ts_8^ts_{14}^t+s_1^ts_8^ts_{18}^t\\&+s_8^ts_{12}^ts_{16}^t+s_8^ts_{14}^ts_{18}^t+s_8^ts_{15}^ts_{16}^t+s_8^ts_{15}^ts_{17}^t\\&+s_{15}^ts_{17}^ts_{24}^t+s_1^ts_8^ts_{14}^ts_{17}^t+s_1^ts_8^ts_{17}^ts_{18}^t\\&+s_1^ts_{14}^ts_{17}^ts_{24}^t+s_1^ts_{17}^ts_{18}^ts_{24}^t+s_8^ts_{12}^ts_{16}^ts_{17}^t\\&+s_8^ts_{14}^ts_{17}^ts_{18}^t+s_8^ts_{15}^ts_{16}^ts_{17}^t+s_{12}^ts_{16}^ts_{17}^ts_{24}^t\\&+s_{14}^ts_{17}^ts_{18}^ts_{24}^t+s_{15}^ts_{16}^ts_{17}^ts_{24}^t+\overline{s_0^t}\,\overline{s_1^t}\cdots\overline{s_{32}^t}\\=&H(\boldsymbol{S}_t)\end{aligned} \tag{2}$$

(3)NFSR2。Draco算法的NFSR2的级数为95，t时刻NFSR2的状态记作$\boldsymbol{B}_t=[b_0^t,b_1^t,\cdots,b_i^t,\cdots,b_{94}^t]$，其中$b_i^t$表示$t$时刻NFSR2的第$i$个状态位比特。

NFSR2的反馈多项式$F(\boldsymbol{B}_t)$是Grain128a[1]中g的修改版本，通过将Grain-128a中g的6个抽头向左移动两个位置(即抽头做如下移位：86←88, 89←91, 90←92, 91←93, 93←95, 94←96)构造的95 bit寄存器的反馈函数；为提高NFSR2的非线性度，给NFSR2更新函数增加了4个2次单项式和1个3次单项式。NFSR2的更新函数为

$$\begin{aligned}b_{94}^{t+1}=&s_0^t+d_t+b_0^t+b_{26}^t+b_{56}^t+b_{89}^t+b_{94}^t\\&+b_3^tb_{67}^t+b_{11}^tb_{13}^t+b_{17}^tb_{18}^t+b_{27}^tb_{59}^t\\&+b_{36}^tb_{39}^t+b_{40}^tb_{48}^t+b_{50}^tb_{79}^t+b_{54}^tb_{71}^t\\&+b_{58}^tb_{63}^t+b_{61}^tb_{65}^t+b_{68}^tb_{84}^t\\&+b_8^tb_{46}^tb_{87}^t+b_{22}^tb_{24}^tb_{25}^t+b_{70}^tb_{78}^tb_{82}^t\\&+b_{86}^tb_{90}^tb_{91}^tb_{93}^t\\=&s_0^t+d_t+F(\boldsymbol{B}_t)\end{aligned} \tag{3}$$

(4)输出函数。为了消除内部状态较小所造成的安全隐患，Draco算法的输出函数有更多的输入比特(53个)和更大的代数次数，以将2个NFSR的状态进一步混淆。记输出函数在t时刻的输出为z_t，则

$$z_t=L_t+Q_t+\mathrm{T1}_t+\mathrm{T2}_t+\mathrm{T3}_t \tag{4}$$

其中

$$\left.\begin{aligned}L_t=&b_7^t+b_{15}^t+b_{32}^t+b_{47}^t+b_{66}^t+b_{80}^t+b_{92}^t\\Q_t=&b_5^tb_{85}^t+b_{12}^tb_{74}^t+b_{20}^tb_{69}^t+b_{34}^tb_{57}^t\\\mathrm{T1}_t=&b_{53}^t+b_{38}^tb_{44}^t+b_{23}^tb_{49}^tb_{83}^t+b_6^tb_{33}^tb_{51}^tb_{73}^t\\&+b_4^tb_{29}^tb_{43}^tb_{60}^tb_{81}^t+b_9^tb_{14}^tb_{35}^tb_{42}^tb_{55}^tb_{77}^t\\&+b_1^tb_{16}^tb_{28}^tb_{45}^tb_{64}^tb_{75}^tb_{88}^t\\\mathrm{T2}_t=&s_{26}^t+s_5^ts_{19}^t+s_{11}^ts_{22}^ts_{31}^t\\\mathrm{T3}_t=&b_{76}^t+s_3^tb_{10}^t+s_{20}^tb_{21}^tb_{30}^t+s_6^ts_{29}^tb_{62}^tb_{72}^t\end{aligned}\right\} \tag{5}$$

2.2 初始化过程

(1)加载密钥。两个NFSR按如式(6)的初始化

$$\left.\begin{aligned} b_j^0 &= \begin{cases} K_j + 1, j = 0 \\ K_j, \quad\ j = 1,2,\cdots,94 \end{cases} \\ s_i^0 &= K_{i+95}, i = 0,1,\cdots,32 \end{aligned}\right\} \tag{6}$$

(2)Grain式混合。如图1所示，对算法计时512次，期间不产生实际的密钥流，输出函数输出反馈到NFSR中参与其内部状态的更新。期间NFSR更新为

$$\left.\begin{aligned} &\text{NFSR2}: \begin{cases} b_j^{t+1} = b_{j+1}^t, j = 0,1,\cdots,93 \\ b_{94}^{t+1} = z_t + s_0^t + d_t + F(\boldsymbol{B}_t) \end{cases} \\ &\text{NFSR1}: \begin{cases} s_i^{t+1} = s_{i+1}^t, i = 0,1,\cdots,31 \\ s_{32}^{t+1} = z_t + H(\boldsymbol{S}_t) \end{cases} \end{aligned}\right\} \tag{7}$$

2.3 密钥流生成过程

初始化过程结束后，NFSR1的状态为$[s_0^{512}, s_2^{512},\cdots,s_{32}^{512}]$，NFSR2的状态为$[b_0^{512}, b_1^{512},\cdots,b_{94}^{512}]$。当$t \geq 512$时，NFSR1和NFSR2每一轮迭代分别向左移位1位，其最高位分别按照式(2)和式(3)更新，算法每一轮迭代产生1位密钥流比特，z_t为t时刻输出的密钥流比特，第1个用来加密明文的密钥流比特为z_{512}。生成和待加密明文等长的密钥流后，再与明文逐位异或生成密文。

3 Draco算法的改进算法Draco-F算法

针对Draco算法选择函数周期较小且硬件实现成本较高，密钥可回溯的问题[16]，对Draco算法的选择函数、输出函数、非易失性状态和初始化过程进行改进，改进算法称为Draco-F算法。和Draco算法类似，Draco-F算法也是基于CIVK方案的小状态流密码算法，同样由1个非易失性存储器、1个选择函数、2个NFSR和1个输出函数组成，该算法具体结构如图2所示。

3.1 算法部件

Draco-F的初始密钥K长度为128 bit，记作$\boldsymbol{K} = [k_0, k_1,\cdots,k_{127}]$；初始向量**IV**长度为96 bit，记作$\mathbf{IV} = [\text{iv}_0, \text{iv}_1,\cdots,\text{iv}_{95}]$。NFSR1的级数为33，$t$时刻NFSR1的状态记作$\boldsymbol{S}_t = [s_0^t, s_1^t,\cdots,s_i^t,\cdots,s_{32}^t]$，其中$s_i^t$表示$t$时刻NFSR1的第$i$个状态位；NFSR2的级数为95，$t$时刻NFSR2的状态记作$\boldsymbol{B}_t = [b_0^t, b_1^t,\cdots, b_i^t,\cdots,b_{94}^t]$，其中$b_i^t$表示$t$时刻NFSR2的第$i$个状态位bit。

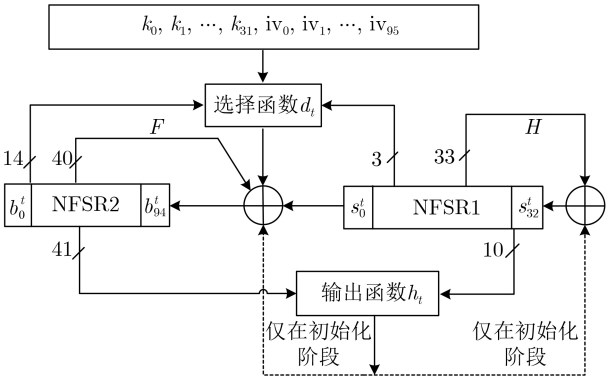

图 2 Draco-F算法结构图

(1)非易失性存储器与基于状态位索引的选择函数：非易失性存储器有128 bit，存储初始密钥$\boldsymbol{K}$的前32 bit和96 bit初始向量，其状态表示为

$$[X_0, X_1,\cdots,X_{127}] = [k_0, k_1,\cdots,k_{31}, \text{iv}_0, \text{iv}_1,\cdots,\text{iv}_{95}] \tag{8}$$

为避免选择函数周期小所造成的安全问题，并降低算法的硬件开销，本文设计了一个基于状态位索引的选择函数：以NFSR1,NFSR2中的17个状态位生成3个索引值r, p, q；以这3个索引值分别作为下标选择存储于非易失性存储器中的3个bit数据；最后，这3个bit数据通过指定的非线性运算生成选择函数的输出位。具体为

$$\left.\begin{aligned} r &= b_9^t \parallel b_{43}^t \parallel b_{37}^t \parallel b_{52}^t \parallel s_{31}^t \\ p &= b_{62}^t \parallel s_3^t \parallel b_{12}^t \parallel b_{23}^t \parallel b_{83}^t \parallel b_{41}^t \\ q &= b_{88}^t \parallel b_{19}^t \parallel b_{73}^t \parallel s_{17}^t \parallel b_{31}^t \parallel b_2^t \end{aligned}\right\} \tag{9}$$

其中，r由NFSR1和NFSR2中的5个bit决定，取值范围为$[0,31]$；p, q则分别由6个bit确定，p, q的取值范围都为$[0,63]$；s_i^t表示t时刻NFSR1的第i个状态位bit，b_i^t表示t时刻NFSR2的第i个状态位bit。

在时钟周期t中，基于状态位索引值r, p, q选择非易失性存储器中的密钥和初始向量bit，选择函数的输出d_t计算为

$$d_t = X_r + X_p + X_{q+64} + X_p X_{q+64} \tag{10}$$

在内部状态为随机的情况下，通过r索引，每一轮可以随机选择1位密钥前缀参与到d_t的运算；通过p索引，可在非易失性存储器前64 bit中随机选取1位密钥前缀或初始向量参与运算，p的取值范围使得X_p来自于密钥前缀和初始向量的概率是相同的；通过$q+64$索引，保证在非易失性存储器后64 bit中随机选取1位初始向量参与运算。这确保每一轮选择均有来自于密钥前缀和初始向量的比特位被选中参与运算。

由于3个索引值r, p, q由NFSR1, NFSR2中的17个状态位生成，随着NFSR1, NFSR2状态的更新，这3个索引值会随之改变，再通过非线性函数充分混合，选择函数的选择位d_t也会动态改变。事实上，由于d_t也参与到NFSR2的状态更新过程，所以这17个用于索引的状态位的更新周期是不确定

的，在最糟糕的情况下即在非易失性存储器为全0状态，d_t不再干预NFSR2的更新时，则NFSR1和NFSR2独立运行，而两者的状态总共为128 bit，因此，选择函数周期最小也可以达到2^{128}。通过状态位索引方法可以保证选择函数具有较大的周期，从而避免针对Draco算法的选择函数周期小而提出的相应攻击威胁。

同时，由于Draco算法的选择函数在硬件实现时需要2个单独的定时器共12 bit来实现其定义的模运算操作，且逻辑实现需要多个多路复用器，硬件开销较大。改进Draco-F算法的选择函数在硬件实现时仅需通过简单的移位、异或、与操作来实现，硬件开销显著降低。

(2)NFSR1：Draco-F的NFSR1沿用了Draco算法的NFSR1，这两者的反馈函数相同，但此时反馈函数中的变量均是Draco-F算法中的变量。该反馈函数的非线性度为114 688，相关免疫阶为6，扩散参数为54，周期为2^{33}。此外，该反馈函数是平衡的，弹性度为6，代数次数为4。

(3)NFSR2：Draco-F的NFSR2沿用了Draco算法的NFSR2，这两者的反馈函数相同，但此时反馈函数中的变量均是Draco-F算法中的变量。该反馈函数的非线性度为549 656 723 456($\approx 2^{39}$)，代数次数为4。因Draco-F算法选择函数的输出d_t始终会参与NFSR2的内部状态更新，因此NFSR2的反馈函数的周期是未知的。

(4)输出函数：为了消除内部状态较小所造成的安全隐患，Draco算法的输出函数使用了NFSR中更多的状态位作为输入bit(53个)和更大的代数次数进行状态混淆，相对于其他的小状态流密码算法的输出函数过于复杂，增加了硬件开销。

Draco-F算法在设计选择函数时，使用了17个NFSR的比特位，为避免NFSR中状态位的重复使用并简化Draco算法的输出函数，Draco-F算法的输出函数由45 bit输入组成，其中，10 bit输入来自于NFSR1，来自NFSR2的35个输入位选择NFSR2的更新函数和选择函数中未用过的状态比特位。该输出函数由具有7个单项的线性函数L_t、3项的三角函数$\mathrm{T1}_t$、7项的三角函数$\mathrm{T2}_t$和4项的三角函数$\mathrm{T3}_t$组成

$$\left.\begin{aligned}
L_t &= b_7^t + b_{15}^t + b_{32}^t + b_{47}^t + b_{66}^t + b_{80}^t + b_{92}^t \\
\mathrm{T1}_t &= s_{26}^t + s_5^t s_{19}^t + s_{11}^t s_{22}^t s_{31}^t \\
\mathrm{T2}_t &= b_{53}^t + b_{38}^t b_{44}^t + b_{20}^t b_{49}^t b_{74}^t + b_6^t b_{33}^t b_{51}^t b_{69}^t \\
&\quad + b_4^t b_{29}^t b_{34}^t b_{60}^t b_{81}^t + b_5^t b_{14}^t b_{35}^t b_{42}^t b_{55}^t b_{77}^t \\
&\quad + b_1^t b_{16}^t b_{28}^t b_{45}^t b_{64}^t b_{75}^t b_{85}^t \\
\mathrm{T3}_t &= b_{76}^t + s_3^t b_{10}^t + s_{20}^t b_{21}^t b_{30}^t + s_6^t s_{29}^t b_{57}^t b_{72}^t
\end{aligned}\right\} \tag{11}$$

记z_t为t时刻输出的密钥流比特，则

$$z_t = L_t + \mathrm{T1}_t + \mathrm{T2}_t + \mathrm{T3}_t \tag{12}$$

Draco-F算法的输出函数较Draco算法的输出函数减少了8个状态位，且通过选用不同于上述函数所用过的比特位作为其输入，可保持NFSR中比特位使用的均匀性，并进一步降低硬件成本。

3.2 初始化过程

针对Draco算法存在的密钥可回溯问题，改进算法Draco-F使用一种改进的动态初始化技术进行解决。动态初始化技术最早由Ghafari在Fruit-80算法中提出[3]，其在算法初始化一定轮数后，将FSR中部分状态位赋值给一个外部定时器中的指定位置，算法每迭代1轮，定时器的值加1，在定时器的值为指定的值(如全1)时，结束动态初始化过程。由于所取的状态位是不确定，更新到指定状态的轮数也不确定，从而防止算法内部状态的回溯。但是使用外部定时器会增加寄存器的位数，从而增加硬件成本。

为避免额外增加外部定时器硬件成本，在Draco-F算法的密钥初始化过程中，以NFSR2后五位的状态变化确定是否结束动态初始化过程。初始化过程包括密钥和初始向量装载、Grain式混合、动态初始化过程、部分暴露状态位异或密钥、最终混淆五个步骤，算法初始化过程运行方式如图2所示，具体如下：

(1)密钥和初始向量装载：读取128 bit初始密钥$\boldsymbol{K} = [k_0, k_1, \cdots, k_{127}]$和96 bit初始向量$\mathbf{IV} = [\mathrm{iv}_0, \mathrm{iv}_1, \cdots, \mathrm{iv}_{95}]$；将128 bit的初始密钥分别加载到2个NFSR中，低位加载到NFSR2，高位加载到NFSR1

$$\left.\begin{aligned}
b_0^0 b_1^0 \cdots b_{95}^0 &= k_0, k_1, \cdots, k_{95} \\
s_1^0 s_2^0 \cdots s_{31}^0 &= k_{96}, k_{97}, \cdots k_{127}
\end{aligned}\right\} \tag{13}$$

(2)Grain式混合：空转128轮，将输出函数的输出比特位与2个NFSR更新函数的输出比特位异或，所得值分别参与两个NFSR新状态位的更新，使得128位NFSR状态完全更新一次。NFSR2与NFSR1更新为

$$\mathrm{NFSR2}: \begin{cases} b_i^{t+1} = b_{i+1}^t, 0 \le i \le 93 \\ b_{94}^{t+1} = s_0^t + z_t + d_t + F(\boldsymbol{B}_t) \end{cases} \tag{14}$$

$$\mathrm{NFSR1}: \begin{cases} s_i^{t+1} = s_{i+1}^t, 0 \le i \le 31 \\ s_{32}^{t+1} = z_t + H(\boldsymbol{S}_t) \end{cases} \tag{15}$$

(3)动态初始化过程：为防止算法内部状态的回溯，并避免额外增加外部定时器硬件成本，以NFSR2的后5位的状态变化来确定是否结束动态初始化过程，即判断$(b_{90}^{128} b_{91}^{128} b_{92}^{128} b_{93}^{128} b_{94}^{128})$的值是否为01011以确定是否结束动态初始化过程。若这5个状态位的值不是01011，则算法继续迭代，每迭代

1次，t的值加1；否则停止迭代。由于这5个状态位的初始值是未知的，因此，动态初始化结束时，t也是未知的。通过动态初始化，有效避免了通过算法某一时刻的内部状态回溯其初始状态得到密钥值的可能。

(4)部分暴露状态位异或密钥：在动态初始化过程中，NFSR2暴露了5 bit。为了保护这些内部状态，防止攻击者利用这一问题进行攻击，对这5 bit进行更新。取非易失性寄存器X中的初始密钥的前5 bit $k_0k_1k_2k_3k_4$分别异或在动态初始化过程中暴露的NFSR2的后五位$b_{90}^t b_{91}^t b_{92}^t b_{93}^t b_{94}^t$，将得到的值依然赋值到NFSR2的后5 bit，即

$$b_i^t = b_i^t + X_{i-90} = b_i^t + k_{i-90}, 90 \le i \le 94 \tag{16}$$

通过该操作，使得从此时开始直到加密结束，算法的内部状态对于敌手来说是不可知的。

(5)最终混淆：算法再次迭代256轮，期间输出比特位仍然参与2个NFSR的状态更新，至此初始化过程结束。这样即使动态初始化过程的轮数为0，算法初始化阶段依然可以最少迭代384轮，保证初始化阶段算法内部可以充分混淆。

3.3 密钥流生成过程

Draco-F是以数据包模式运行的流密码算法。由于密钥前缀为32 bit且NFSR1只有33个状态位，在密钥流生成阶段，为保证安全性，限制在同一个密钥-初始向量对下生成的密钥流长度最大为2^{32} bit。密钥流生成阶段算法以如下方式运行：

假设明文长度为p，t时刻的密钥流比特基于输出函数z_t计算，而后更新NFSR2和NFSR1。

以当前状态下NFSR1的第1个比特s_0^t, d_t和NFSR2的更新函数的输出值进行异或，更新NFSR2的最高位，NFSR2的更新函数为

$$\left.\begin{aligned} b_i^{t+1} &= b_{i+1}^t, 0 \le i \le 93 \\ b_{94}^{t+1} &= s_0^t + d_t + F(\boldsymbol{B}_t) \end{aligned}\right\} \tag{17}$$

更新NFSR1，其中NFSR1的更新函数为

$$\left.\begin{aligned} s_i^{t+1} &= s_{i+1}^t, 0 \le i \le 31 \\ s_{32}^{t+1} &= H(\boldsymbol{S}_t) \end{aligned}\right\} \tag{18}$$

令$t = t + 1$, $p = p$–1，若$p \ne 0$，继续迭代；若$p = 0$，结束迭代，将这一过程中z_t按生成的时间先后排序得到密钥流z。

4 算法安全性分析

本节对Draco-F算法进行安全性分析，包括通用TMDTO攻击、攻破Draco算法的零d流攻击和选择IV攻击、猜测确定攻击、由状态恢复攻击导致的密钥恢复攻击和随机性测试。

4.1 通用TMDTO攻击

通用TMDTO攻击可分为两种情况：不考虑内部状态特殊结构和考虑内部状态特殊结构。

为方便描述，以$\ell_{\rm nv}$表示非易失性内部状态长度，$\ell_{\rm v}$为易失性内部状态长度，$\ell_{\rm s}$为总的内部状态大小，$\ell_{\rm k}$为密钥长度，$\ell_{\rm p}$为单个密钥流数据包最大长度，$\ell_{\rm IV}$为初始值长度。CIVK方案使用长度为$\log_2\ell_{\rm p}$的密钥前缀和初始值作为非易失性状态，使用密钥初始化易失性存储器，$\ell_{\rm v} = \ell_{\rm k} = \ell_{\rm nv}$。**IV**长度$\ell_{\rm IV}$由密钥和数据包长度决定

$$\ell_{\rm p} \cdot 2^{\ell_{\rm IV}} = 2^{\log_2(\ell_{\rm p}) + \ell_{\rm IV}} = 2^{\ell_{\rm nv}} \tag{19}$$

Draco-F算法中$\ell_{\rm nv}$为128，$\ell_{\rm v}$为128，$\ell_{\rm s}$为256，$\ell_{\rm k}$为128，$\ell_{\rm p}$为2^{32}，$\ell_{\rm IV}$为96。在不考虑内部状态的特殊结构的情况下，发动对Draco-F算法的攻击，则该攻击的权衡曲线为

$$T \cdot D = 2^{\ell_{\rm s}} = 2^{\ell_{\rm nv} + \ell_{\rm v}} \tag{20}$$

可获得的最大数据量D为

$$\ell_{\rm p} \cdot 2^{\ell_{\rm IV}} = 2^{\log_2(\ell_{\rm p}) + \ell_{\rm IV}} = 2^{\ell_{\rm nv}} \tag{21}$$

则权衡曲线产生的时间复杂度$T = 2^{\ell_{\rm v}} = 2^{\ell_{\rm k}}$，对于Draco-F算法该复杂度为$2^{128}$，因此Draco-F算法可以抵抗这种形式的TMDTO攻击。

第2种攻击基于对手已知对应数据包初始值。假设攻击者获得p个长度为$\ell_{\rm p}$的密钥流数据包，对应初始值$\mathbf{IV}_1, \mathbf{IV}_2, \cdots, \mathbf{IV}_p$，数据复杂度$D = p \cdot \ell_{\rm p}$；对每个$\mathbf{IV}_i$，攻击者随机生成$s$次密钥前缀$k_i^{\rm pre}$和随机易失性内部状态$z_i \in \{0,1\}^{\ell_{\rm v}}$。攻击者据此计算长度为$\ell_{\rm s}$的输出密钥流块$S$。如果$D \cdot s = 2^{\ell_{\rm v}}$，则易失性内部状态中的碰撞很有可能发生。

此外，对手需要正确的非易失性内部状态。若已知**IV**，且以${\ell_{\rm p}}^{-1}$的概率正确猜测密钥前缀。则每个$\mathbf{IV}_i$生成的s个密钥前缀中的一个是正确的概率为$s/\ell_{\rm p}$。若下式成立，则认为攻击成功

$$D \cdot \frac{s}{\ell_{\rm p}} = p \cdot \ell_{\rm p} \cdot \frac{s}{\ell_{\rm p}} = p \cdot s = 2^{\ell_{\rm v}} \tag{22}$$

不难发现，无论观察到多少个密钥流数据包，攻击者的时间复杂度T始终为$p \cdot s = 2^{\ell_{\rm v}}$，对于Draco-F算法该复杂度为$2^{128}$，因此Draco-F算法也可以抵抗这种形式的TMDTO攻击。

4.2 零d流攻击

这是一种利用选择函数的输出值d_t在某段时间或从始至终一直为0的TMDTO攻击。具体地，若没有添加这个0位，则Draco算法非易失性状态A为$(K_{\rm pre}, \mathbf{IV})$，此时若存在$(K_{\rm pre}||K_{\rm pre}||K_{\rm pre})$形式的**IV**，使得$A = (K_{\rm pre}, K_{\rm pre}||K_{\rm pre}||K_{\rm pre})$，此时通过Draco选择函数计算出来的$d_t$在任何时刻都为0，这意味

着，非易失性状态完全独立于算法，算法有效状态位仅为128位。这种情形下进行的TMDTO攻击的时间复杂度和空间复杂度都仅为2^{60}，对整个算法进行TMDTO攻击的时间复杂度和空间复杂度也仅为2^{80}。为了使算法达到128位的安全级别，算法设计者们在非易失性状态中额外添加一个0位，以试图避免这种攻击的威胁。但是在密码算法发布后，Banik还是利用了Draco算法的结构漏洞成功的实施了一种特殊形式的零d流攻击，攻击的具体细节可见于文献[16]中的第1种攻击。

为抵御零d流攻击，在Draco-F算法中没有通过添加0位来解决这一问题，而是通过状态位索引方法设计了大周期(至少为2^{128})的选择函数，一方面该方法使得选择函数的输入以近似随机的状态呈现；另一方面该选择函数也被设计为更加复杂的非线性函数，从而消除了Draco算法的这一结构缺陷。在这种情况下，如果攻击者想要继续寻找有序的零d流将会变得非常困难，如果坚持使用零d流攻击的话，其成本将远远大于密钥穷搜。

4.3 选择IV攻击

对于**IV**前96−E位连续为0的特殊情况，Draco存在新的漏洞：考虑某个$\lambda \equiv 0 \bmod 97$，可以看出，(97–E)位序列$d_\lambda, d_{\lambda+1}, \cdots, d_{\lambda+96-\mathrm{E}}$不依赖于**IV**的前96−E位。E表示活跃的比特位数。基于此，文献[16]提出突破Draco安全声明界限的选择**IV**攻击(第2种攻击)。不难发现该漏洞仍然是由Draco的选择函数造成的：选择函数2个输入位中的1位一直以小周期(97)顺序遍历**IV**。

而Draco-F的选择函数有3个输入，且都在指定范围内近似随机地选择非易失性存储器中的比特位，并通过一个非线性选择函数将密钥和初始值进行混合，因此Draco-F不存在上述漏洞，并且可以很好的规避选择**IV**攻击的风险。

4.4 猜测确定攻击

注意到Draco-F在动态初始化过程的最后一轮中暴露了5个状态位，为防止猜测确定攻击的潜在威胁，在设计时把暴露的5个状态位选在NFSR2最新生成的5个位置，并随即用密钥值的前5 bit异或暴露的5 bit状态值，所得值仍然赋值给这5个状态位，通过这种方式可有效避免猜测确定攻击的威胁，但这种方式也相应增加了部分硬件成本。

4.5 内部状态的不可逆

所有形式的TMDTO攻击和猜测确定攻击等攻击都需要先确定密码算法的某一时刻内部状态，得到该内部状态后，想要进一步确定密钥值，需要算法的内部状态可逆，即在初始化阶段和密钥流生成阶段的状态更新函数都是有效可逆的。Draco-F算法使用动态初始化技术模糊初始化过程的轮数并引入部分密钥再异或的操作，使得密钥不可回溯。

4.6 随机性测试

本文使用NIST 2014年发布的随机性检测组件NIST sts-2_1_2对Draco-F生成的密钥流进行随机性检验。NIST随机性检测的两个准则如下：

(1)每个检测项计算出来的P值不小于显著性水平α为通过，α建议取值范围为0.001～0.01；通过的样本总量有上下界，上下界计算为

$$\left(1-\alpha-3\sqrt{\alpha(1-\alpha)/s}, 1-\alpha+3\sqrt{\alpha(1-\alpha)/s}\right) \tag{23}$$

本次测试设置100组数据，设置α为0.01，则计算的上下界为(0.960 150 4, 1.019 849 6)。

(2)P值的统计一致性需满足要求。

设置100组数据，每组数据长度为10^6 bit，共10^8 bit数据，设$\alpha = 0.01$，测试结果如表1所示。

表1结果表明，Draco-F算法通过了NIST随机性测试。

5 算法软硬件测试及结果分析

5.1 软件测试

使用C++分别实现Draco算法和Draco-F算法，并测试算法的密钥流吞吐率。实验计算机的CPU为11th Gen Intel(R) Core(TM) i7-11800H @ 2.30 GHz，内存16 GB，操作系统为Windows 11家庭中文版。

在测试过程中，使用timeit()函数定位密钥流生成开始和结束阶段，计算算法生成10^8 bit密钥流

表 1 Draco-F算法随机性检验结果

编号	测试统计项	P-value值	通过率	检测结果
1	Frequency	0.048 716	0.99	Pass
2	BlockFrequency	0.851 383	0.99	Pass
3	CumulativeSums	0.488 509	0.99	Pass
4	Runs	0.383 827	0.98	Pass
5	LongestRun	0.798 139	1.00	Pass
6	Rank	0.955 835	1.00	Pass
7	FFT	0.275 709	1.00	Pass
8	NonOverlapingTemplate	0.543 258	0.989	Pass
9	OverlappinTemplate	0.122 325	1.00	Pass
10	Universal	0.419 021	0.99	Pass
11	ApproximteEntropy	0.514 124	1.00	Pass
12	RandomExcursions	0.531 523	0.996	Pass
13	RandomExcursionsVariant	0.454 231	0.996	Pass
14	Serial	0.498 609	0.995	Pass
15	LinearComplexity	0.236 810	1.00	Pass

所需时间t，并重复测试10次取平均值。测得Draco[11]和Draco-F的密钥流吞吐率如表2所示。

从表2可以看出，两个算法在初始化轮数、非易失性状态长度、密钥流吞吐率存在差异，Draco-F算法比Draco算法在数据结构上显得更加整齐，密钥流吞吐率比Draco算法高约3.9%。

5.2 硬件测试

使用Verilog语言实现Draco-F算法，并进行时序仿真和功能验证。进一步使用Synopsys Design Compiler工具，基于SMIC 0.13 μm逻辑工艺的标准单元库，使用50 MHz的频率进行综合设计，测试算法的面积和功耗，并在相同环境下与Grain-128a[1], Atom[10], Draco[11]进行对比。测试结果如表3所示。

表3结果显示，Draco-F和Draco的硬件面积和功耗都远低于Grain-128a和Atom。与Draco相比，尽管Draco-F为了防止内部状态可逆造成的密钥恢复攻击，在初始化阶段更复杂，但硬件实现面积仍然低于Draco，功耗稍高于Draco。

6　结论

本文针对Draco算法存在的安全缺陷，提出基于状态位索引和动态初始化的改进算法Draco-F算法。该算法通过状态位索引的方法增加了选择函数的周期并降低了其硬件成本；其次，在保障NFSR状态位使用均匀性的前提下，简化了输出函数，进一步降低了算法的硬件成本；最后，通过动态初始化技术有效阻止对密钥的回溯。对Draco-F算法的安全性分析和软硬件测试结果表明，相对于Draco算法，Draco-F算法解决了Draco算法面临的安全缺陷，具有更高的安全性，且Draco-F算法具有更高的密钥流吞吐率和更小的电路面积。在将来的工作中，进一步优化Draco-F算法结构，在保证算法安全的前提下提高算法执行效率并降低硬件面积。

表 2　两种算法的软件实现性能

算法	初始化轮数	非易失性内部状态长度(bit)	密钥流吞吐率(kbit/s)
Draco	512	129	308
Draco-F	动态变化	128	320

表 3　不同算法的硬件指标结果

算法	面积		功耗
	(μm^2)	(GE)	(mW)
Grain-128a[1]	13214.51	2911.33	0.479
Atom[10]	14070.26	3099.86	0.383
Draco[11]	10127.22	2231.15	0.309
Draco-F	10083.37	2221.49	0.315

参 考 文 献

[1] ÅGREN M, HELL M, JOHANSSON T, *et al*. Grain-128a: A new version of Grain-128 with optional authentication[J]. *International Journal of Wireless and Mobile Computing*, 2011, 5(1): 48–59. doi: 10.1504/IJWMC.2011.044106.

[2] EKDAHL P, JOHANSSON T, MAXIMOV A, *et al*. A new SNOW stream cipher called SNOW-V[J]. *IACR Transactions on Symmetric Cryptology*, 2019, 2019(3): 1–42. doi: 10.13154/tosc.v2019.i3.1-42.

[3] AMIN GHAFARI V and HU Honggang. Fruit-80: A secure ultra-lightweight stream cipher for constrained environments[J]. *Entropy*, 2018, 20(3): 180. doi: 10.3390/e20030180.

[4] ZIDARIČ N, MANDAL K, GONG G, *et al*. The welch-gong stream cipher-evolutionary path[J]. *Cryptography and Communications*, 2024, 16(1): 129–165. doi: 10.1007/s12095-023-00656-0.

[5] 冯秀涛. 3GPP LTE国际加密标准ZUC算法[J]. 信息安全与通信保密, 2011, 9(12): 45–46. doi: 10.3969/j.issn.1009-8054.2011.12.033.
FENG Xiutao. ZUC algorithm: 3GPP LTE international encryption standard[J]. *Information Security and Communications Privacy*, 2011, 9(12): 45–46. doi: 10.3969/j.issn.1009-8054.2011.12.033.

[6] KUMAR S and SARKAR S. Conditional TMDTO as a MILP instance[J]. *IEEE Transactions on Information Theory*, 2023, 69(5): 3330–3346. doi: 10.1109/TIT.2022.3230910.

[7] ARMKNECHT F and MIKHALEV V. On lightweight stream ciphers with shorter internal states[C]. The 22nd International Workshop on Fast Software Encryption, Istanbul, Turkey, 2015: 451–470. doi: 10.1007/978-3-662-48116-5_22.

[8] HAMANN M, KRAUSE M, and MEIER W. LIZARD-A lightweight stream cipher for power-constrained devices[J]. *IACR Transactions on Symmetric Cryptology*, 2017, 2017(1): 45–79. doi: 10.13154/tosc.v2017.i1.45-79.

[9] MIKHALEV V, ARMKNECHT F, and MÜLLER C. On ciphers that continuously access the non-volatile key[J]. *IACR Transactions on Symmetric Cryptology*, 2017, 2017(2): 52–79. doi: 10.13154/tosc.v2016.i2.52-79.

[10] BANIK S, CAFORIO A, ISOBE T, *et al*. Atom: A stream cipher with double key filter[J]. *IACR Transactions on Symmetric Cryptology*, 2021, 2021(1): 5–36. doi: 10.46586/tosc.v2021.i1.5-36.

[11] HAMANN M, MOCH A, KRAUSE M, *et al*. The DRACO stream cipher: A power-efficient small-state stream cipher with full provable security against TMDTO attacks[J]. *IACR Transactions on Symmetric Cryptology*, 2022, 2022(2): 1–42. doi: 10.46586/tosc.v2022.i2.1-42.

[12] HAMANN M and KRAUSE M. On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks[J]. *Cryptography and Communications*, 2018, 10(5): 959–1012. doi: 10.1007/

s12095-018-0294-5.

[13] HAMANN M, KRAUSE M, MEIER W, *et al*. Design and analysis of small-state grain-like stream ciphers[J]. *Cryptography and Communications*, 2018, 10(5): 803–834. doi: 10.1007/s12095-017-0261-6.

[14] HAMANN M, KRAUSE M, and MOCH A. Tight security bounds for generic stream cipher constructions[C]. The Selected Areas in Cryptography–SAC 2019: 26th International Conference, Waterloo, Canada, 2020: 335–364. doi: 10.1007/978-3-030-38471-5_14.

[15] GÜL Ç and KARA O. A new construction method for keystream generators[J]. *IEEE Transactions on Information Forensics and Security*, 2023, 18: 3735–3744. doi: 10.1109/TIFS.2023.3287412.

[16] BANIK S. Cryptanalysis of Draco[J]. *IACR Transactions on Symmetric Cryptology*, 2022, 2022(4): 92–104. doi: 10.46586/tosc.v2022.i4.92-104.

[17] GAMMEL B, GÖTTFERT R, and KNIFFLER O. Achterbahn-128/80: Design and analysis[C]. ECRYPT Network of Excellence-SASC Workshop Record, Bochum, Germany, 2007: 152–165.

张润莲：女，副教授，研究方向为信息安全与分布式计算.
范　欣：男，硕士生，研究方向为信息安全.
赵　昊：男，硕士生，研究方向为信息安全.
武小年：男，教授，研究方向为信息安全与分布式计算.
韦永壮：男，教授，研究方向为分组密码算法设计与分析.

责任编辑：余　蓉

The Small-state Stream Cipher Algorithm Draco-F Based on State-bit Indexing Method

ZHANG Runlian　FAN Xin　ZHAO Hao　WU Xiaonian　WEI Yongzhuang

(*School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin* 541000, *China*)

Abstract:

Objective　The Draco algorithm is a stream cipher based on the Consisting of the Initial Value and Key-prefix (CIVK) scheme. It claims to provide security against Time Memory Data TradeOff (TMDTO) attacks. However, its selection function has structural flaws that attackers can exploit. These weaknesses can compromise its security. To address these vulnerabilities and lower the hardware costs associated with the Draco algorithm, this paper proposes an improved version called Draco-F. This new algorithm utilizes state bit indexing and dynamic initialization.

Methods　Firstly, to address the small cycle problems of the selection function and the high hardware costs in the Draco algorithm, the Draco-F algorithm introduces a new selection function. This function employs state bit indexing to extend the selection function's period and reduce hardware costs. Specifically, the algorithm generates three index values based on 17 status bits from two Nonlinear Feedback Shift Registers (NFSRs). These index values serve as subscripts to select three bit of data stored in non-volatile memory. The output bit of the selection function is produced through specified nonlinear operations on these three bit of data. Secondly, while ensuring uniform usage of NFSR state bits, the Draco-F algorithm further minimizes hardware costs by simplifying the output function. Finally, Draco-F incorporates dynamic initialization techniques to prevent key backtracking.

Results and Discussions　Security analysis of the Draco-F algorithm, including evaluations against universal TMDTO attacks, zero stream attacks, selective IV attacks, guessing and determining attacks, key recovery attacks, and randomness testing, demonstrates that Draco-F effectively avoids the security vulnerabilities encountered by the original Draco algorithm, thereby offering enhanced security. Software testing results indicate that the Draco-F algorithm achieves a 128-bit security level with an actual 128-bit internal state and higher key stream throughput compared to the Draco algorithm. Additionally, hardware testing results reveal that the circuit area of the Draco-F algorithm is smaller than that of the Draco algorithm.

Conclusions　In comparison to the Draco algorithm, the Draco-F algorithm significantly enhances security by addressing its vulnerabilities. It also offers higher key stream throughput and a reduced circuit area.

Key words: Stream cipher; Consisting of the Initial Value and Key-prefix (CIVK); Draco; State bit indexing; Dynamic initialization