基于Camellia算法的快速流密码算法设计与特性研究

丁 杰 石 会 龚 晶 邓元庆
(解放军理工大学通信工程学院 南京 210007)

摘 要 Camellia算法作为欧洲分组密码加密标准,与AES算法具有一致的安全性与适用性。以Camellia算法为核心部件,从部分轮函数F中提取4个字节的中间状态作为密钥流输出,设计了一种新的快速流密码算法,并分析了它的相关特性。分析结果表明,该算法的密钥流生成速度和密钥流随机性与同类型的LEX算法大致相当,但由于每个Camellia模块的输入与密钥均发生了改变,因此该算法可以有效地抵抗LEX算法所不能抵抗的滑动攻击。

关键词 流密码,Camellia,随机性,安全性,快速

中图法分类号 TP309.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.08.026

Design and Characteristic Study on Fast Stream Cipher Algorithm Based on Camellia

DING Jie SHI Hui GONG Jing DENG Yuan-qing
(College of Communications Engineering, PLA University of Science & Technology, Nanjing 210007, China)

Abstract As the encryption standard of the block cipher of NESSIE, Camellia algorithm has the same security and applicability as AES algorithm. In this paper, a novel fast stream cipher algorithm was proposed based on Camellia algorithm. The idea is to extract parts of the internal state at certain round function F and give them as the output keystream. We analyzed the relative characteristics of the new algorithm. The result shows that the new algorithm achieves almost the same performance as the optimal performance obtained in LEX, in terms of keystream generation speed and randomness. Besides, it can resist slide attack, with both input and key changing in each Camellia module.

Keywords Stream cipher, Camellia, Randomness, Security, Fast

1 引言

对称密码体制作为当今世界两大密码体系之一,其因计算开销小、算法简单、加密速度快等优点而普遍应用于信息加密系统中。根据加密形式的不同,对称密码体制可进一步划分为分组密码体制和流密码体制两类。其中,以香农一次一密的密码体制为理论基础的流密码因实现简单、加解密速度快、错误传播率低等特点,在无线通信与外交通信中得到了广泛应用,但其本身也存在低扩散性与低混淆性等缺陷。与流密码不同,分组密码具有扩散性好、混淆度高、插入敏感、保密度强等优点,因此,当前流密码的发展趋势之一便是将高强度分组密码引入流密码设计中,以提高流密码的安全性。虽然分组密码的计数器模式(CTR)、密码反馈模式(CFB)、输出反馈模式(OFB)可以直接运用于流密码,但因密钥流产生速度不高而影响了流密码的加解密速度。如何提高基于分组密码的流密码加解密速度,是流密码设计中需要解决的一个重要问题。

本文设计了一种基于Camellia算法[1]的快速流密码算法,并分析了该算法的密钥流随机性、算法安全性及运行速度。采用Camellia算法设计流密码算法的主要原因是Camellia算法具有极高的抗破译性能,通过它可以大大提升流密码算法的安全性能。

2 Camellia算法简介

Camellia算法由日本电报电话公司(NTT)与三菱公司(MEC)联合设计,并于2003年2月由欧洲信息协会技术委员会(Information Society Technologies, IST)发布,与美国高级数据加密标准算法AES[1]一同成为21世纪欧洲NES-SIE(New European Schemes for Signatures, Integrity, and Encryption)[2]密码工程确定的分组密码算法标准。Camellia算法采用经典的Feistel结构,其外部接口与AES算法相同,数据分组长度为128bit,主密钥(种子密钥)长度可以根据性能要求的不同分别选择128,192或256bit。

到稿日期:2016-07-17 返修日期:2017-01-05 本文受国家自然科学基金项目(61501512)资助。

丁 杰(1989—),女,硕士生,主要研究方向为保密技术与管理,E-mail:731661839@qq.com;石 会(1982—),女,硕士,副教授,主要研究方向为保密技术与管理,E-mail:hpkry@126.com(通信作者);龚 晶(1968—),女,硕士,副教授,主要研究方向为信息安全;邓元庆(1958—),男,硕士,教授,主要研究方向为保密技术与管理。

2.1 Camellia 算法所采用的符号及其含义

$X_{(n)}$：表示向量 x 的比特长度为 n；

X_L：表示向量 x 的左半部分；

X_R：表示向量 x 的右半部分；

$\oplus$：逐比特异或(模 2 加)；

$\|$：将两个操作数串接起来；

$\lll n$：操作数循环左移 n 个比特；

$\cap$：逐比特与运算；

$\cup$：逐比特或运算；

$kw_{t(64)}, kl_{v(64)}$：子密钥，其中，$t=1,2,3,4, v=1,2,3,4$。

2.2 Camellia 的算法结构

128 比特密钥的 Camellia 算法的加密结构如图 1 所示。与 AES 采用 Squarer 的加密结构有所不同，Camellia 算法采用了经典的 Feistel 密码结构，总体包括密钥扩展、明文 $M_{(128)}$ 预白化、18 轮 F 函数的 Feistel 迭代、2 轮 FL/FL^{-1} 变换以及后白化处理 5 个部分，最后输出 128bit 密文 $C_{(128)}$，第 6 轮和第 12 轮 Feistel 迭代后的 2 轮 FL/FL^{-1} 变换的作用是打乱算法结构中 Feistel 迭代的规律性。

Camellia 的加密算法伪代码可描述如下：

$M_{(128)} \oplus (kw1_{(64)} \| kw2_{(64)}) = L_{0(64)} \| R_{0(64)}$

for r=1 to r=18(除了 r =6 和 r=12)

$L_r = R_{r-1} \oplus F(L_{r-1}, k_r)$

$R_r = L_{r-1}$

for r=6 and r=12

$L_r' = R_{r-1} \oplus F(L_{r-1}, k_r)$

$R_r' = L_{r-1}$

$L_r = FL(L_r', kl_{2r/6-1})$

$R_r = FL^{-1}(R_r', kl_{2r/6})$

$C_{(128)} = (R_{18(64)} \| L_{18(64)}) \oplus (kw3_{(64)} \| kw4_{(64)})$

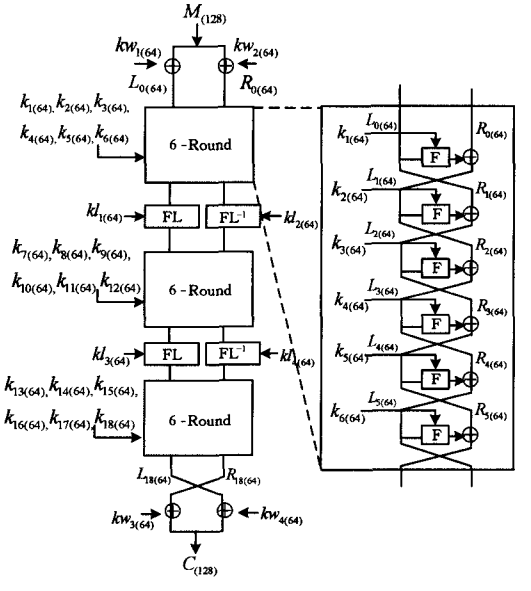


图 1　Camellia 的加密算法结构

Camellia 的解密算法结构与加密算法结构完全相同，只是在进行迭代运算时子密钥使用顺序相反，且 $L_{i(64)}$ 和 $R_{i(64)}$ 的位置结构与加密时的定义正好相反。当密钥为 192 或 256bit 时，只需在图 1 中第 18 轮迭代之后再增加一个 FL/FL^{-1} 变换层和一个 6 轮的 F 函数 Feistel 迭代模块，共计需要进行 24 轮 F 函数 Feistel 迭代和 3 轮 FL/FL^{-1} 变换。

2.3 Camellia 加密算法中的变换函数

2.3.1 F 变换

F 变换是 Camellia 算法的核心函数，在密钥扩展、加密和解密过程中均有使用。F 变换内部由非线性变换 S 盒运算和线性变换 P 盒组成。其中，S 变换由 4 个不同的 8 比特 S 盒 S_1, S_2, S_3, S_4 组成，主要起到混淆作用；P 变换是一个线性映射，帮助算法抵抗差分和线性密码分析。

其具体变换过程为：64bit 明文 $X_{(64)}$ 被划分为 8 个字节，在 8 个字节的子密钥 $k_{i(64)}$ 的作用下，进行模 2 加法运算 $y_{j(8)} = x_{j(8)} \oplus k_{ij(8)}$，然后再进行复杂的 S 变换(见式(1))和 P 变换(见式(2))，最后输出 64bit 密文 $Z'_{(64)}$，该过程的数学表达式可描述为式(3)，结构框图如图 2 所示。

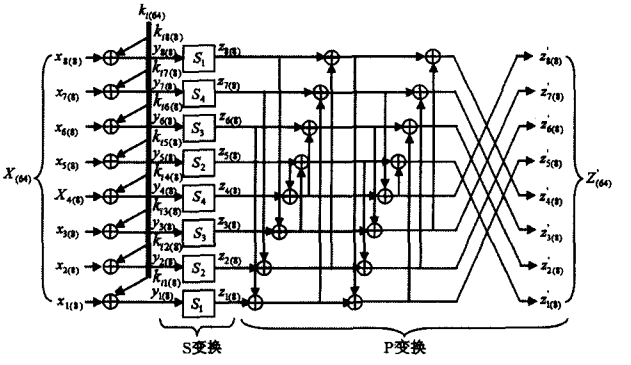


图 2　F 变换的结构框图

其中，f 变换、h 变换和 g 变换均为有限域 $GF(2^8)$ 上的变换，非常复杂，计算时可根据设计者给出的 S 盒变换表格直接进行查表。

$$\begin{cases} S_1 : z_{j(8)} = h(g(f(c5 \oplus y_{j(8)}))) \oplus 6e \\ S_2 : z_{j(8)} = S_1(y_{j(8)}) \lll 1 \\ S_3 : z_{j(8)} = S_1(y_{j(8)}) \ggg 1 \\ S_4 : z_{j(8)} = S_1(y_{j(8)} \lll 1) \end{cases} \tag{1}$$

$$P: GF(2^8) \rightarrow GF(2^8) \tag{2}$$

P 变换的详细过程如下：

$$\begin{cases} z'_{1(8)} = z_{8(8)} \oplus z_{7(8)} \oplus z_{6(8)} \oplus z_{4(8)} \oplus z_{3(8)} \oplus z_{1(8)} \\ z'_{2(8)} = z_{8(8)} \oplus z_{7(8)} \oplus z_{5(8)} \oplus z_{4(8)} \oplus z_{2(8)} \oplus z_{1(8)} \\ z'_{3(8)} = z_{8(8)} \oplus z_{6(8)} \oplus z_{5(8)} \oplus z_{3(8)} \oplus z_{2(8)} \oplus z_{1(8)} \\ z'_{4(8)} = z_{7(8)} \oplus z_{6(8)} \oplus z_{5(8)} \oplus z_{4(8)} \oplus z_{3(8)} \oplus z_{2(8)} \\ z'_{5(8)} = z_{8(8)} \oplus z_{7(8)} \oplus z_{6(8)} \oplus z_{2(8)} \oplus z_{1(8)} \\ z'_{6(8)} = z_{8(8)} \oplus z_{7(8)} \oplus z_{5(8)} \oplus z_{3(8)} \oplus z_{2(8)} \\ z'_{7(8)} = z_{8(8)} \oplus z_{6(8)} \oplus z_{5(8)} \oplus z_{4(8)} \oplus z_{3(8)} \\ z'_{8(8)} = z_{7(8)} \oplus z_{6(8)} \oplus z_{5(8)} \oplus z_{4(8)} \oplus z_{1(8)} \end{cases}$$

$$Z'_{(64)} = F(X_{(64)}, k_{i(64)}) = P(S(X_{(64)} \oplus k_{i(64)})) \tag{3}$$

2.3.2 FL/FL^{-1} 变换

为了打乱整个算法的规律性，确保算法能够有效地抵抗攻击，Camellia 算法设计者在算法每进行 6 轮 F 函数后嵌入一轮 FL/FL^{-1} 变换。由于 FL 与 FL^{-1} 互为逆变换，因此嵌入

后并不影响加解密结构的一致性。

FL 变换的算法描述如下：

$Y_{R(32)}=((X_{L(32)}\cap kl_{L(32)})\lll 1)\oplus X_{R(32)}$

$Y_{L(32)}=(Y_{R(32)}\cup kl_{R(32)})\oplus X_{L(32)}$

FL^{-1} 变换的算法描述如下：

$X_{L(32)}=(Y_{R(32)}\cup kl_{R(32)})\oplus Y_{L(32)}$

$X_{R(32)}=((X_{L(32)}\cap kl_{L(32)})\lll 1)\oplus Y_{R(32)}$

3 基于 Camellia 算法的快速流密码算法设计

基于 Camellia 算法的快速流密码算法结构如图 3 所示。所设计的算法采用 128bit 密钥的 Camellia 算法作为密钥流产生部件，从每一个 Camellia 算法模块中输出 320 比特中间变量作为密钥流输出。

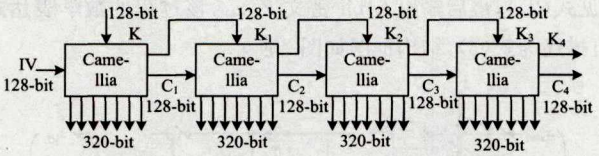


图 3 基于 Camellia 的流密码算法结构框图

算法的运行过程如下：首先，给定一个 128bit 的初始向量 IV 作为输入，在 128bit 种子密钥 K 的作用下，用标准 Camellia 算法对 IV 进行加密，得到第一次加密结果 $C_1=Camellia_k(IV)$，种子密钥 K 在执行完第一个 Camellia 加密模块后经过密钥扩展发生变化。第二个 Camellia 加密模块的种子密钥 K_1 为上一个 Camellia 加密模块的后白化密钥 $kw_{3(64)}\parallel kw_{4(64)}$。$C_1$ 和 K_1 作为第二个 Camellia 模块的输入与加密密钥，进行第二次 Camellia 加密运算，得到 C_2 和 K_2，并作为下一轮的输入与加密密钥。以后各模块类似。

算法每执行一次，可提取 320bit 密钥流，具体过程如下：执行 Camellia 算法模块，进行子密钥生成、初始向量预白化后，进入 18 轮 F 函数变换，挑选其中 10 轮迭代并从中提取密钥流，即从每次迭代后所得的 128bit 中间状态中挑选 4 字节(共 32bit)作为输出密钥流的一部分。经过分析，本算法建议每个 Camellia 加密模块的第 1－7 轮迭代作为算法的初始化部分，使各模块能够充分扩散与混淆，以增强提取密钥流的随机特性，降低与后续加密模块输入向量间的相关性；第 8－17 轮迭代为密钥流提取轮，其中第 12 轮和第 13 轮之间的 FL/FL^{-1} 变换层不进行密钥流提取，起置乱作用；最后一轮不进行密钥流提取。

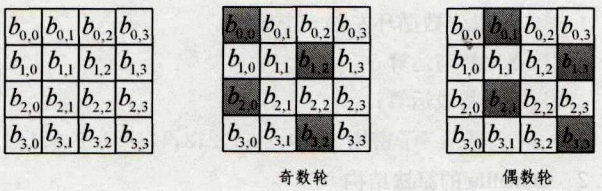
算法设计中最关键的部分是输出密钥流的提取位置和提取频率。经过反复测试、验证，本文建议采取图 4 所示的(方案 A 和方案 B)两个提取方案之一。

以方案 A 为例说明密钥提取方式。方框内一个 $b_{i,j}$ 代表一个字节的中间变量，奇数轮提取 $b_{0,0}$，$b_{2,0}$，$b_{1,2}$，$b_{3,2}$ 4 个字节，偶数轮提取 $b_{0,1}$，$b_{2,1}$，$b_{1,3}$，$b_{3,3}$ 4 个字节。对初次提取密钥的第 8 轮不作轮数上的更改，即为第 8 轮。其过程可用公式描述为：

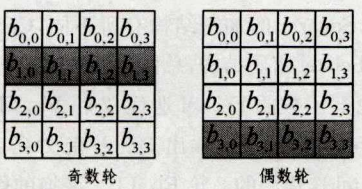
$oddout32=(t_0\&0xFF000000)\oplus((t_2\&0xFF000000)\gg 8)\oplus(t_1\&0x0000FF00)\oplus((t_3\&0x0000FF00)\gg 8)$

$evenout32=((t_0\&0x00FF0000)\ll 8)\oplus(t_2\&0x00FF0000)\oplus((t_1\&0x000000FF)\ll 8)\oplus(t_3\&0x000000FF)$

其中，$t_i=(b_{i,0},b_{i,1},b_{i,2},b_{i,3})$，$i=0,1,2,3$，根据 i 的不同取值，其可代表每轮迭代后中间状态的第 0,1,2,3 个行向量，各由 4 个字节组成。



(a)方案 A



(b)方案 B

图 4 奇数轮和偶数轮的具体提取位置

通过选择奇偶两种不同的密钥提取方式，可以确保攻击者在实施攻击时不能仅通过一轮函数迭代分析输入与输出比特的关系，攻击者必须分析两轮连续的迭代函数，但通过 Camellia 的两轮 F 函数之后，中间状态已经得到了充分的混淆，这就限制了攻击者的攻击能力。

第 8－12 轮密钥提取的伪代码如下(其他轮类似)：

```
void six_F2(unsigned C[2][8], unsigned ki[2][8], unsigned m[11][4])
{  unsigned t[8]; //用作运算的中间变量
   static p=0;
   for(int i=0;i<6;i++) //进行中间 6 轮 F 函数运算
   {
      copy(C[0],t);//将本轮输入赋给中间变量 t
      F(t,ki[i]);//F 函数迭代
      for(int j=0;j<8;j++)
        t[j]=t[j]^C[1][j];
      copy(C[0],C[1]);
      copy(t,C[0]);//将迭代后的中间状态重新赋给 C
      if (i%2==0&&i>0) //奇数轮提取密钥位置
      {
         m[p][0]=C[0][0];
         m[p][1]=C[1][0];
         m[p][2]=C[0][6];
         m[p][3]=C[1][6];
      }
      if (i%2!=0&&i<6) //偶数轮提取密钥位置
      {
         m[p][0]=C[0][1];
         m[p][1]=C[1][1];
         m[p][2]=C[0][7];
         m[p][3]=C[1][7];
```

```
    }
    P++;
  }
  P=0;
}
```

4　基于 Camellia 的流密码算法的随机性检验

评判一个流密码算法是否符合要求的重要标准是密钥流的随机性检测。我国随机性检测标准《随机性检测规范》[3]规定，序列须通过单比特频数检测、重叠子序列检测、块内频数检测、扑克检测、游程总数检测、块内最大“1”游程检测、游程分布检测、二元推导检测、近似熵检测、自相关检测、累加和检测、线性复杂度检测、矩阵秩检测、Maurer 通用统计检测、离散傅里叶检测共 15 种随机性检测，才能说它具有好的随机特性。

序列随机性的检测方法如下：

(1)用随机数发生器产生 1000 个长度为 100 万比特的 0-1序列作为待检样本。

(2)对待检验的 1000 个样本依次进行上述 15 项随机性检测。

(3)对于每一项随机性检测项目，若统计 P-value 值不小于显著性水平 α(建议取 $\alpha=0.01$)，则表示该样本通过该项检测。

(4)记样本数量为 s，如果通过检测的样本个数不小于 $s(1-\alpha-3\sqrt{\frac{\alpha(1-\alpha)}{s}})$，则序列产生器通过随机性检测。

下面以游程总数检测为例进行简要说明。

游程，是指一段二元序列中连续 0 或者连续 1 所组成的子序列，并且该子序列的前导和后继元素都与游程本身元素不同，检测的方法[3]如下。

(1)设待检测序列 $\varepsilon_1\varepsilon_2\cdots\varepsilon_n$ 的长度为 n，游程总数用 $V_n(obs)$表示，则 $V_n(obs)=\sum_{i=1}^{n-1}r(i)+1$。其中，当 $\varepsilon_i=\varepsilon_{i+1}$ 时，$r(i)=0$；否则 $r(i)=1$；

(2)计算序列中 1 的比例，即 $\pi=\frac{\sum_{i=1}^{n}\varepsilon_i}{n}$；

(3)计算统计量：$V=\frac{|V_n(obs)-2n\pi(1-\pi)|}{\sqrt{n}\pi(1-\pi)}$；

(4)计算 $P\text{-}value=erfc(\frac{V}{2\sqrt{2}})$；

(5)如果 $P\text{-}value\geqslant\alpha$，则认为该序列通过游程总数检测。

本文以 C++编程软件和《随机性检测规范》为依托，编写了基于 Camellia 算法的流密码算法程序，并搭建了针对该算法而设计的随机性检测平台，分别对算法设计的多个方案所产生的 0-1 序列进行测试。为确保测试结果具有统一的效应，每个方案均采用相同的十六进制初始向量 IV 与种子密钥 K：

IV:01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10

K:01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10

每个方案分别产生 1000 个 100 万比特 0-1 序列，如果每项检测通过数不小于 981(未通过数不大于 19)，则通过该项检测；如果本规范所规定的 15 项检测全部通过，则该流密码算法通过本规范检测；否则，未通过本规范检测。

方案 A 的样本随机性检测结果局部图如图 5 所示。

0614Camellia 方案A1000组样本测试结果 - Microsoft Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
962	0.876033	0.5848952	0.1087139	0.46847	0.741418	0.5163494	0.535953	0.105878	0.249319	0.315846	0.288556	0.3737646	0.0680309	0.7288403	0.559222	
963	0.678874	0.3538776	0.0680687	0.92501	0.718975	0.979017	0.153665	0.236017	0.75656	0.550484	0.728962	0.6347745	0.0996114	0.2474295	0.475367	
964	0.736871	0.9836788	0.1036944	0.737086	0.570112	0.8294813	0.674244	0.598193	0.309628	0.02946	0.307699	0.216216	0.0525607	0.6758158	0.948263	
965	0.825871	0.2925941	0.3437577	0.571569	0.318256	0.4934625	0.353268	0.83601	0.119232	0.315846	0.286912	0.8951151	0.9317049	0.7136819	0.745603	
966	0.470295	0.3440761	0.0916667	0.065634	0.29256	0.0429741	0.000895	0.629095	0.091795	0.259966	0.31468	0.2654336	0.192498	0.2958446	0.436178	00
967	0.332047	0.5289105	0.8387055	0.08292	0.381606	0.6467872	0.259636	0.477084	0.488936	0.043843	0.443674	0.8594046	0.5489998	0.8917854	0.948263	
968	0.179596	0.4630943	0.5925798	0.071676	0.108317	0.4903329	0.79752	0.522822	0.744423	0.557117	0.919132	0.9271357	0.3536988	0.0755259	0.153422	
969	0.687684	0.8180551	0.4310114	0.454962	0.06516	0.2827681	0.403139	0.487055	0.308676	0.057883	0.702655	0.4617963	0.1370711	0.1128453	0.104758	
970	0.309629	0.4805282	0.4763078	0.737041	0.106751	0.4627566	0.532007	0.115256	0.675946	0.679142	0.668023	0.1615553	0.7577249	0.3103444	0.697031	
971	0.470295	0.1062463	0.6679679	0.333441	0.659559	0.5223404	0.095225	0.192574	0.695057	0.574792	0.365592	0.8480844	0.1248167	0.9273907	0.475367	
972	0.234833	0.5418465	0.5269409	0.393635	0.060188	0.5937099	0.106255	0.162413	0.879187	0.76722	0.231896	0.9641502	0.6718471	0.5317431	0.948263	
973	1	0.0352152	0.0822878	0.532109	0.598888	0.965638	0.68813	0.012245	0.947378	0.552748	0.538182	0.874484	0.8558589	0.4728193	0.948263	
974	0.977662	0.9105175	0.1563147	0.084118	0.777943	0.7115095	0.949146	0.050819	0.629805	0.759865	0.126622	0.1167116	0.4209266	0.477839	0.21762	
975	0.166359	0.1052754	0.2583925	0.897462	0.583749	0.923184	0.326379	0.278808	0.128509	0.259966	0.85463	0.9201989	0.9607306	0.3177879	0.697031	
976	0.89024	0.7433809	0.6618582	0.417974	0.607265	0.7686547	0.739785	0.241598	0.839916	0.391613	0.981775	0.478949	0.9496564	0.6310934	1	
977	0.993617	0.7392378	0.765553	0.815543	0.171939	0.8454054	0.805844	0.689893	0.145391	0.699055	0.865378	0.7632248	0.3872276	0.9191076	0.436178	
978	0.838354	0.9477423	0.2234019	0.988637	0.968126	0.9880626	0.651996	0.743668	0.5144	0.302218	0.79217	0.9989961	0.6593632	0.8618997	0.242809	
979	0.794864	0.6306361	0.7075865	0.61985	0.403193	0.6187411	0.879806	0.221329	0.011669	0.547764	0.810283	0.7652448	0.1307055	0.7268226	0.845655	
980	0.010407	0.8461054	0.3833483	0.362329	0.92625	0.5688035	0.779118	0.400347	0.33907	0.409558	0.832407	0.9084094	0.9913626	0.3990345	0.39892	
981	0.603064	0.0325794	0.7939409	0.282569	0.405562	0.2327173	0.998888	0.334544	0.730845	0.285583	0.933776	0.7109076	0.7549738	0.812589	0.475367	
982	0.177016	0.1449934	0.4119096	0.223994	0.301988	0.3125345	0.128228	0.718099	0.634073	0.032082	0.600222	0.9533089	0.1582448	0.8525962	0.104758	
983	0.38649	0.2686194	0.4351851	0.772273	0.908247	0.6853891	0.948631	0.911616	0.774877	0.043843	0.57008	0.3785948	0.8468346	0.3198268	0.436178	
984	0.723339	0.3042101	0.6743082	0.61674	0.907553	0.2017416	0.628438	0.861079	0.596111	0.285583	0.893606	0.654783	0.6541715	0.2270031	0.795208	
985	0.665741	0.8876203	0.7541216	0.948577	0.141612	0.2727927	0.039366	0.925904	0.378857	0.020562	0.452752	0.1454689	0.2801645	0.2572861	0.33039	
986	0.250144	0.7553362	0.538335	0.841399	0.340426	0.2612699	0.073859	0.983246	0.73687	0.032094	0.499167	0.6921442	0.8287699	0.112223	0.603685	
987	0.25932	0.8983764	0.9129077	0.145429	0.431122	0.2089508	0.169084	0.259742	0.555189	0.302218	0.05421	0.7147125	0.328332	0.155909	0.948263	
988	0.400908	0.2113944	0.6799374	0.397974	0.74995	0.0648515	0.747631	0.445463	0.574115	0.679142	0.004501	0.8259243	0.3919004	0.1684157	0.948263	00
989	0.860294	0.9756984	0.3542694	0.730209	0.409922	0.4666261	0.885414	0.984841	0.153867	0.574792	0.509782	0.8868272	0.8083251	0.9744016	0.649671	
990	0.193601	0.1257133	0.8351449	0.539092	0.335893	0.9172042	0.995332	0.285069	0.433038	0.557117	0.36954	0.4642846	0.6652172	0.7553349	0.39892	
991	0.15854	0.9459701	0.5887874	0.561043	0.295566	0.3840774	0.597557	0.426613	0.265277	0.426499	0.312923	0.674809	0.5008882	0.0738202	0.896743	
992	0.904483	0.1698494	0.4265661	0.461529	0.703956	0.3875495	0.881388	0.513756	0.58369	0.699055	0.995968	0.3625075	0.6765245	0.1759297	0.242809	
993	0.513112	0.2981319	0.3619052	0.492444	0.889902	0.1971399	0.404295	0.278808	0.850876	0.06617	0.315267	0.4997784	0.5505442	0.717512	0.603685	
994	0.684743	0.0418786	0.5862796	0.823511	0.218759	0.9736671	0.843911	0.574797	0.571392	0.259966	0.036946	0.8126284	0.8895391	0.8796007	0.269981	
995	0.610052	0.5300366	0.0534495	0.137475	0.654341	0.2390332	0.21765	0.242403	0.095712	0.426499	0.895844	0.6588102	0.4045056	0.2970608	0.299169	
996	0.821201	0.0126852	0.0089389	0.815159	0.284642	0.6355448	0.998494	0.834448	0.180244	0.715227	0.402949	0.5444267	0.1905375	0.9799328	0.436178	00
997	0.564615	0.8629933	0.634212	0.104478	0.984308	0.3135675	0.99511	0.038546	0.980853	0.866289	0.743033	0.6960037	0.0049134	0.3834197	0.649671	00
998	0.015016	0.8135749	0.0786111	0.037577	0.289103	0.3979856	0.49803	0.973675	0.004994	0.107263	0.789427	0.4621223	0.6318877	0.5141499	0.896743	00
999	0.615668	0.2539924	0.3386824	0.640763	0.168281	0.4528045	0.156027	0.206589	0.741399	0.082211	0.677353	0.3648713	0.6947419	0.7188621	0.269981	
1000	0.453255	0.0613208	0.2205583	0.287391	0.109474	0.5319562	0.95824	0.494606	0.041151	0.397583	0.477563	0.6003517	0.349237	0.0919784	0.948263	

图 5　方案 A 随机性检测结果局部图

截取序号为第 962－1000 的样本检测情况，列 A 至列 O 依次代表规范中的 15 项不同的随机性检测，列 P 为设定的函数，若任一样本 A1－O1 的检测项中存在小于 0.01 的P-value 值，则该行 P 值为 00，表格背景加深项表示对应样本的对应检测项的检测值不合格。

方案 A 和方案 B 的样本随机性检测总体情况见表 1。为便于对比分析，表 1 同时列出了基于 AES 算法的同类高速流密码算法 LEX 算法[4]的 1000 个长度为 100 万比特的样本数据随机性检测结果[5]。从检测结果可知，两种方案的 15 项检测未通过值均小于 19，与 LEX 算法的检测结果相近，通过了随机性检测，符合序列的随机性要求。

表 1 各项检测未通过样本数

检测方法	各项检测未通过的样本数		
	LEX	方案 A	方案 B
单比特频数检验	6	14	7
块内频数检测	10	13	10
扑克检测	2	17	8
重叠子序列检测	9	14	6
游程总数检测	13	12	10
游程分布检测	13	13	18
块内最大 1 游程检测	10	11	9
二元推导检测	7	16	11
自相关检测	9	6	12
矩阵秩检测	18	16	17
累加和检测	7	8	16
近似熵检测	8	14	12
线性复杂度检测	14	2	14
Maurer 通用统计检测	14	12	9
离散傅里叶检测	0	0	1
检测结果	通过	通过	通过

表 2 列出了关于方案 A 和方案 B 的 15 项随机性检测值的特征分析结果。表中也同时列出了 LEX 算法的对应检测项的检测值[5]的特征分析结果。

表 2 P-value 值特征分析

检测方法	P-value 值的期望			P-value 值的方差		
	LEX	方案 A	方案 B	LEX	方案 A	方案 B
单比特频数检验	0.502	0.514	0.497	0.082	0.084	0.083
块内频数检测	0.500	0.501	0.487	0.087	0.082	0.083
扑克检测	0.512	0.503	0.503	0.082	0.082	0.086
重叠子序列检测	0.492	0.509	0.505	0.082	0.085	0.084
游程总数检测	0.509	0.495	0.491	0.081	0.080	0.084
游程分布检测	0.478	0.488	0.482	0.083	0.086	0.080
块内最大 1 游程检测	0.493	0.489	0.499	0.088	0.082	0.085
二元推导检测	0.494	0.498	0.492	0.081	0.087	0.080
自相关检测	0.520	0.494	0.491	0.084	0.081	0.077
矩阵秩检测	0.507	0.391	0.405	0.082	0.063	0.065
累加和检测	0.502	0.516	0.493	0.081	0.085	0.087
近似熵检测	0.496	0.508	0.515	0.084	0.085	0.078
线性复杂度检测	0.493	0.498	0.489	0.082	0.080	0.086
Maurer 通用统计检测	0.482	0.500	0.485	0.083	0.086	0.085
离散傅里叶检测	0.502	0.597	0.618	0.083	0.067	0.065

分析检测结果可知，方案 A、方案 B 与 LEX 算法所生成的密钥流的随机特性基本保持一致，除矩阵秩和离散傅里叶两项检测的 P-value 值期望分别在 0.4 和 0.6 左右，方差约为 0.06 外，其他各项 P-value 值的期望约为 0.5，方差在 0.08 上下浮动。考虑到字节提取位置复杂度与算法设计安全性之间的关系，本文认为选择方案 A 中的密钥流提取方法更好。

结束语 本算法以 Camellia 算法为核心，通过提取 F 函数的部分中间状态作为输出密钥流，每个 Camellia 加密模块可以得到 320bit 的密钥流输出，密钥流产生速度为 Camellia 算法 CTR，CFB，OFB 模式的 2.5 倍，与 LEX 算法相当[5]。编程测试表明，该算法产生的密钥流可以通过国家密码局的《随机性检测规范》检验，具有与 LEX 算法同样好的随机性。此外，由于每个 Camellia 模块的输入与加密密钥均不同，因此本算法能够有效抵抗差分故障攻击[6]和滑动攻击[7]，故其抗差分故障攻击和滑动攻击的性能优于 LEX 算法。与一般的流密码算法相比，本算法由于依托高强度的 Camellia 算法，因此具有更高的安全性。

参 考 文 献

[1] 邓元庆，龚晶，石会. 密码学简明教程[M]. 北京：清华大学出版社，2011：71-105.

[2] European Union. European Project IST-1999-12324：New European Schemes for Signatures，Integrity，and Encryption[EB/OL]. [2002-03-16]. http://www.cosic.esat.kuleuven.bc/nessie.

[3] Randomness Test Specification：GM/T0005-2012[S]. Beijing：Standards Press of China，2012. (in Chinese)
随机性检测检测规范：GM/T0005-2012[S]. 北京：中国标准出版社，2012

[4] BIRYUKOV A. A new 128 bit Key Stream Cipher LEX [EB/OL]. [2005-06-13]. http://www.ecrypt.eu.org/stre-am/chiphers/lex/lex.pdf.

[5] LI J Y，SHI H，DENG Y Q，et al. Improvement and Analysis on Slide Attack-Resistant Stream Cipher LEX[J]. Communications Technology，2015，48(2)：203-207. (in Chinese)
李佳雨，石会，邓元庆，等. 抗滑动攻击的 LEX 算法改进及分析[J]. 通信技术，2015，48(2)：203-207.

[6] LI J Y，SHI H，DENG Y Q，et al. Differential Fault Attack and Analysis of Improvement on LEX[J]. Computer Science，2015，42(11A)：352-356. (in Chinese)
李佳雨，石会，邓元庆，等. 针对流密码 LEX 的差分故障攻击及算法改进分析[J]. 计算机科学，2015，42(11A)：352-356.

[7] WU H J，PRENEEL B. Attacking the IV Setup of Stream Cipher LEX[EB/OL]. [2006-03-15]. http://www.ecrypt.eu.org/stream/papersdir/059.pdf.