DOI:10.11992/tis.201606002
网络出版地址:http://www.cnki.net/kcms/detail/23.1538.TP.20160808.0830.012.html

随机序列的扑克检测优化研究

杨先伟[1],康红娟[2],廖祖华[3,4]
(1. 无锡职业技术学院 基础部,江苏 无锡 214121; 2. 四川长虹电器股份有限公司,四川 成都 610041; 3. 江南大学,江苏 无锡 214122; 4.江南大学 智能系统与网络计算研究所,江苏 无锡 214122)

摘　要:现代计算机系统的安全性依赖于二元随机序列,随机性检测利用概率统计方法对二元序列的随机性进行分析测试。我国国家密码管理局发布了随机性检测规范,扑克检测为其中一个检测项。本文通过充分分析扑克检测效率不高的原因有针对性地提出一种新的快速实现算法,优化算法充分利用 CPU 字长一次处理多个比特,将 m 为 4 和 8 的情况整合在一起,减少不必要的处理流程。同时精简并优化统计量的计算和判断过程,避免余不完全伽马函数的计算。分析和实验的结果表明该优化算法可以使得扑克检测的速度提升 9.5 倍左右。

关键词:二元序列;随机序列;随机数发生器;随机性检测;扑克检测;密码算法;效率分析;余不完全伽玛函数
中图分类号: TP18　　**文献标志码**:A　　**文章编号**:1673-4785(2016)04-0513-06

中文引用格式:杨先伟,康红娟,廖祖华. 随机序列的扑克检测优化研究[J]. 智能系统学报, 2016, 11(4): 513-518.
英文引用格式:YANG Xianwei, KANG Hongjuan, LIAO Zuhua. Study on optimization of poker test random sequences[J]. CAAI Transactions on Intelligent Systems, 2016, 11(4): 513-518.

Study on optimization of poker test random sequences

YANG Xianwei[1], KANG Hongjuan[2], LIAO Zuhua[3,4]
(1. Department of Fundamental Courses, Wuxi Institute of Technology, Wuxi 214121, China; 2. Sichuan Changhong Electric Co., Ltd., Chengdu 610041, China; 3.School of Science, Jangnan University, Wuxi 214122, China; 4.Institute of Intelligence System &Network Computing, Jiangnan University, Wuxi 214122, China)

Abstract:The security of modern computer systems depends on binary random sequences, such as cipher algorithms keys, RSA algorithm prime numbers, the digital signature system, the identity authentication system, etc. Randomness tests analyze and test the randomness of sequences, using probability and statistics. The Chinese National Cryptography Administration has released national randomness test specifications and the Poker test is one of these. This paper analyzed the reasons for the low efficiency of the Poker test, then proposes a fast implementation algorithm. This new algorithm deals with bytes by making full use of CPU word length, integrates the detection process, and reduces some unnecessary operations under the conditions when m equals 4 and 8. At the same time, the method reduces and optimizes the computation and assessment of statistical quantity, avoiding computation of incomplete gamma functions. The results show that the efficiency of the new algorithm increases 9.5 fold.

Keywords: binary sequence; random sequence; pseudorandom bit generator; randomness test; poker test; encryption algorithms; efficiency analysis; incomplete gamma functions.

二元随机序列在密码应用中占有举足轻重的地位。现在大量的计算机系统的安全性需要依赖于二元随机序列,比如各种密码算法中使用的密钥、非对称密码算法 RSA 加密、数字签名方案中大素数的生

收稿日期:2016-06-01.　**网络出版日期**:2016-08-08.
基金项目:国家自然科学基金项目(61170121,11401259);江苏省自然科学基金项目(BK20151117).
通信作者:廖祖华. E-mail:liaozuhua57@163.com.

成以及挑战应答身份识别系统中的挑战数等,这些都充分体现了二元随机序列的实际使用价值。在应用密码学中,随机性检测采用概率统计的方法对随机数发生器等生成的二元序列的随机性进行分析和检测,判断待检序列在统计上是否难以和真随机数区分开来。不同的随机性检测算法从不同的侧面分析刻画待检二元序列与真随机序列之间的差距。经过多年的发展,随机性检测算法已取得丰硕成果,出现并颁布了大量的随机性检测算法和相关标准,与此同时,大量新的随机性检测算法还在源源不断地涌现。美国国家标准与技术研究院(National Institute of Standards and Technology, NIST)发布了 SP 800-22 标准[1],其中建议了 16 种用于随机性测试的统计检测方法。德国以此规范为基础发布了 BSI AIS 30 规范[2]。我国国家密码管理局于 2009 年颁布了适用于我国的随机性检测规范[3]。随机性检测在实际应用中有重要的实用价值,不仅可以用于测评按照密码算法或特定标准生成的伪随机数据的性质,如分组算法与某些工作模式相结合生成的数据流[4],还可以分析测试以杂凑算法为核心生成的数据流,如我国 SM3 杂凑算法[5]生成的数据流。上述工作不仅可以帮助算法分析,减少分析的难度和复杂度,而且可以检测出其他检测方法难以检测出的某些安全性隐患。因此,国际上很多著名的密码算法竞赛均进行了大量的随机性检测评估,比如 AES 密码算法竞赛、欧洲的 NISSIE 竞赛、estream 算法竞赛等,我国的祖冲之序列密码算法[6-7]也进行了大量相关的随机性检测。此外,还有大量文章对随机性检测算法和标准进行进一步的分析讨论,比如讨论随机性检测规范的各项检测项的快速实现,例如研究单比特检测以及块内频数的快速实现[8],尝试新的统计测试方法以便作为现有测试规范的有益补充[9],考虑统计测试算法的 GPU 并行化,搭建可并行计算的统计测试实现框架[10]。本文的研究重点是扑克检测的快速实现,因为扑克检测不仅是我国随机性检测规范的检测项,而且也是很多基本的随机性检测的必检项目之一,所以此项检测的快速实现研究具有非常重要的现实意义。

1 扑克检测

我国的随机性检测规范有 15 项检测,包括:单比特频数检测、块内频数检测、扑克检测、重叠子序列检测、游程总数检测、块内最大 1 游程检测、矩阵秩检测、累积和检测、近似熵检测、线性复杂度检测、Maurer 通用统计检测、离散傅里叶变换检测、游程分布检测、二元推导检测、自相关检测。扑克检测不仅出现在我国的随机性检测规范中,也出现在很多别的基本随机性检测中,比如作为 5 项基本检测项中的一项出现,5 项基本检测包括单比特频数检测、序偶检测、扑克检测、游程分布检测、自相关检测。

我国随机性检测规范对扑克检测的执行流程分为 4 个步骤,描述如下[3]:

1) 将长度为 n 的二元待检序列 $\varepsilon_1\varepsilon_2\cdots\varepsilon_n$ 划分为 $N = n/m$ 个长度为 m 的非重叠子序列,将多余的比特舍弃。统计第 i 种子序列模式出现的频数,用 n_i ($1 \leqslant i \leqslant 2^m$)表示。我国随机性检测规范规定 m 取 4 和 8。

2) 计算统计值:

$$V = \frac{2^m}{N}\left(\sum_{i=1}^{2^m} n_i^2\right) - N \tag{1}$$

3) 计算 P 值:

$$P_{\text{value}} = \text{igamc}\left(\frac{2^m - 1}{2}, \frac{V}{2}\right) \tag{2}$$

4) 如果 $P_{\text{value}} \geqslant \alpha$,则认为待检序列通过检测。

式(2)中使用的 igamc 是余不完全伽马函数,该函数的定义为

$$\text{igamc}(\alpha, x) = 1 - \frac{\int_x^{\infty} \mathrm{e}^{-t} t^{\alpha-1} \mathrm{d}t}{\Gamma(\alpha)}, \alpha > 0, x > 0 \tag{3}$$

式中:$\Gamma(x)$ 为伽马函数,该函数的定义为

$$\Gamma(x) = \int_0^{\infty} \mathrm{e}^{-t} t^{x-1} \mathrm{d}t, x > 0 \tag{4}$$

扑克检测在很多基本的随机性检测中都会出现,因此这是一种基础检测项,在随机性检测中具有非常重要的作用。在实际应用中,该检测项需要具有较高的检测速度,以便快速剔除那些明显不满足随机性特征的样本。

2 原算法的检测效率分析

在实际应用中,送入的待检序列数据多为字节序列,但传统的实现方式会根据算法描述先将待检数据转化为单比特表示,然后对 $m = 4$ 和 $m = 8$ 各执行一遍算法中的 1) ~ 4) 的操作。扑克检测算法中 3) 和 4) 在一次样本检测中执行两次,1) 的执行次数则为 $O(n)$;2) 的执行次数则为 $O(1)$,所以其中最关键也最耗时操作为准备工作的字节转比特操作以及 1) 的统计各种频数。

在分析算法的计算量时,本文采用如下缩略符号:XOR 表示异或运算;SHIFT 表示左/右移位运算;ADD 表示加法运算;AND 表示与运算。

传统的扑克检测中最关键也最耗时的步骤是

1),因此以下分析执行一组样本检测时 1)所需的运算量。在 $m = 4$ 和 $m = 8$ 时 1)需分别执行 n 次 SHIFT、n 次 LOAD 和 $2n$ 次 ADD,即合计总共需执行 $2n$ 次 SHIFT、$2n$ 次 LOAD 和 $4n$ 次 ADD。1)进行一组样本检测所需执行的操作数详情如表 1。

表 1　扑克检测算法的操作数量

Table 1　The calculation of poker test

步骤 1	运算量
$m = 4$	nSHIFT + nLOAD +2nADD
$m = 8$	nSHIFT +nLOAD +2nADD
合计	2nSHIFT +2nLOAD +4nADD

按我国随机性检测规范规定一组样本的大小 n 为 10^6 bit,因此由表 1 的统计结果可知,算法的运算量不小,执行效率不会太快。在实际检测中,需要加快扑克检测的执行速度,以增强它的基础筛选作用。

3　优化思想和优化算法

根据上一节的分析可知,扑克检测的效率不高的主要原因是计算统计量时出现了以下问题:

1)采用了单比特统计方式,每次仅仅处理一个比特,CPU 的字长没有得到充分利用;如果一次处理多个比特则处理速度将有明显的提升;

2)对参数 $m = 4$ 和 $m = 8$,传统实现方式会各执行一遍算法中的 1)~4)的操作,存在相同数据反复加载的情况;

3)算法的统计量计算和判断过程没有精简和优化,存在不必要的余不完全伽马函数的计算。

针对扑克检测原算法出现的效率不高问题,下面有针对性地提出几点优化想法。具体的优化想法如下:

1)一次处理多个比特,比如一个字节或半个字节,加快频数统计过程;

2)对 $m = 4$ 和 $m = 8$ 整合在一起实现,减少不必要的数据加载;

3)精简并优化统计量的计算和判断过程,事先计算 $P_{\text{value}} \geqslant \alpha$ 时统计量 V 的阈值,让统计值直接和此阈值比较,避免每个样本都计算两次余不完全伽马函数。

记待检序列为 $n/8$ 字节的字节数据 E_i , $E_i = \varepsilon_{8i+1} \| \varepsilon_{8i+2} \| \cdots \| \varepsilon_{8i+8}$, $0 \leqslant i \leqslant n/8 - 1$。为区分两种不同参数取值时各种序列模式出现的频数,记 $C_4[i], 0 \leqslant i \leqslant 15$ 为 $m = 4$ 时各种序列模式出现的频数,记 $C_8[i], 0 \leqslant i \leqslant 255$ 为 $m = 8$ 时各种序列模式出现的频数。

参数 $m = 8$ 时的频数统计方式为直接加载字节数据并更新频数,即

$$C_8[E_i] = C_8[E_i] + 1, 0 \leqslant i \leqslant \frac{n}{8} - 1 \quad (5)$$

参数 $m = 4$ 时的频数统计方式为先加载字节数据,接着获取高半字节和低半字节,

$$H = E_i \gg 4, L = E_i \wedge 0xF, 0 \leqslant i \leqslant \frac{n}{8} - 1 \quad (6)$$

最后利用获取的高半字节和低半字节更新对应的频数,即

$$C_4[H] = C_4[H] + 1, C_4[L] = C_4[L] + 1 \quad (7)$$

参数 $m = 4$ 时和 $m = 8$ 时的统计过程分开实现会使得待检数据序列重复加载,这也是扑克检测效率不高的主要原因之一。如果能更进一步将参数在两种不同取值时的统计过程合并在一起,则可以减少大量的数据重复加载。参数 m 取 4 和 8 合并实现时的频数统计方式为先加载字节数据,然后获取高半字节和低半字节,最后更新 $m = 4$ 的频数和 $m = 8$ 的频数,合并式(5)~(7),可得

$$\begin{aligned} &H = E_i \gg 4, L = E_i \wedge 0xF, 0 \leqslant i \leqslant n/8 - 1 \\ &C_4[H] = C_4[H] + 1, C_4[L] = C_4[L] + 1 \\ &C_8[E_i] = C_8[E_i] + 1 \end{aligned} \quad (8)$$

统计量的计算和判断过程还可以进行精简和优化:可根据余不完全伽马函数的性质预先求出 $P_{\text{value}} \geqslant \alpha$ 时统计量 V 的阈值,让统计值 V 直接和此阈值比较,如此可以减少余不完全伽马函数的计算次数。

记 $m = 4$ 时的统计量为 V_4,即

$$V_4 = \frac{64}{n}\left(\sum_{i=0}^{15} C_4[i]^2\right) - \frac{n}{4} \quad (9)$$

记 $m = 8$ 时的统计量为 V_8,即

$$V_8 = \frac{2\,048}{n}\left(\sum_{i=0}^{255} C_8[i]^2\right) - \frac{n}{8} \quad (10)$$

计算统计值所用的余不完全伽马函数满足性质 $\text{igamc}(\alpha, 0) = 1$, $\text{igamc}(\alpha, \infty) = 0$。经简单计算可知,当显著水平 $\alpha = 0.01$, $m = 4$ 时,统计量 V_4 的阈值为 $\lambda_4 = 30.577\,914$;当显著水平 $\alpha = 0.01$, $m = 8$ 时,统计量 V_8 的阈值为 $\lambda_8 = 310.457\,388$。即如果 $V_4 < \lambda_4$ 且 $V_8 < \lambda_8$ 则认为待检序列通过检测。根据以上描述,优化实现的扑克检测算法如下。

算法 1　优化实现的扑克检测算法

输入　$n/8$ 字节的数据 $E_i, 0 \leqslant i \leqslant n/8 - 1$;

输出　检测结果。

1)初始化数据:$i = 0$。

$C_4[j] = 0, 0 \leqslant j \leqslant 15$,

$C_8[j] = 0, 0 \leqslant j \leqslant 255$。

2) 当 $i < n/8$ 时执行如下步骤;否则转 3)。

① $X = E_i$, $H = X \gg 4$, $L = E_i \wedge 0xF$

② $C_4[H] = C_4[H] + 1$,

$C_4[L] = C_4[L] + 1$,

$C_8[X] = C_8[X] + 1$,

③ $i = i + 1$。

3) 计算两个统计值 V_4 和 V_8。

$$V_4 = \frac{64}{n}\left(\sum_{i=0}^{15} C_4[i]^2\right) - \frac{n}{4}$$

$$V_8 = \frac{2\,048}{n}\left(\sum_{i=0}^{255} C_8[i]^2\right) - \frac{n}{8}$$

4) 如果 $V_4 < \lambda_4$ 且 $V_8 < \lambda_8$,则认为待检序列通过检测。否则未通过。

算法 1 的主要优化方式是直接对输入的待检序列按字节而不是比特进行处理,减少了大量不必要的数据拆分为单比特等操作;并且将两种参数下的频数统计合并在一起,避免了大量的数据加载等操作。

4 优化前后计算量分析与对比

本节对优化前的算法和优化后的算法的计算量进行定量评估与对比。

原算法的 2) ~4) 的计算量都很小,因此本节在进行计算量评估对比时,只比较最关键最耗时的步骤统计频数所需要的运算量。为简化表示,将 n 比特二元序列的字节长度记为 M, $M = n/8$。而且通常情况下输入的二元序列都是以字节表示,因此这里默认待检二元序列的比特长度 n 能被 8 整除。

根据第 2 节的分析结果知,对一个 $n = 10^6$ bit ($M = 125 \times 10^3$ 字节)的样本而言,原算法 1) 的计算量为 16M 次 SHIFT、16M 次 LOAD 和 32M 次 ADD。扑克检测优化算法 1 的 1) 进行简单分析可知,对一个 $n = 10^6$ bit 的样本而言,算法 1 的 2) 的计算量为 M 次 SHIFT、M 次 LOAD、M 次 AND 和 3M 次 ADD。原算法和优化算法(算法 1)的运算量详情以及对比情况见表 2。

表 2 两个算法的运算量对比

Table 2 The performance comparison of two algorithms

运算	原算法	算法 1
LOAD	16M	M
SHIFT	16M	M
AND	0	M
ADD	32M	3M

由表 2 可知,优化后的扑克检测的计算量显著降低。

在现在的 CPU 中,常见的整数运算都比较快,如整数的加、减、比较、比特运算及移位等仅需一个时钟周期(cycle)。但数据加载的执行时间则有很多因素,无法以准确的值计量。这里仅做粗略估计,因此加速数据加载也仅需要一个时钟周期。这样一来原算法对一个样本进行检测的粗略估计时间为 64M 个时钟周期,算法 1 对一个样本进行检测的粗略估计时间为 6M 个时钟周期,以此法粗略估计,算法 1 的检测速度为原算法的 10.7 倍左右。当然,此处为不精确的粗略估计而已,具体的速度提升情况以实验为准。

5 模拟实验测试

为更准确地说明本文提出的算法的效率,本节测试优化前后算法的执行效率。

测试数据是利用我国的分组密码算法 SM4 算法生成的 10^9 bit 的伪随机数据,按样本大小 10^6 比特划分为 1 000 个样本。

测试平台为 Intel Core i3 @ 3400MHz 处理器、4 GB DDR3 1600MHz 内存、Windows XP SP3 操作系统、Visual Studio 2008 编译器。处理器的缓存情况为:一级缓存为每个核心 32 KB,二级缓存为每个核心 64 KB,三级缓存为多核共享 3 MB。

模拟实验使用的代码情况如下。优化前的测试代码来源是先从 NIST 的官方网站取得检测代码,然后按原算法以及 NIST 代码思想对以比特表示的二元序列,按比特操作实现扑克检测,NIST 代码完成字节序列转比特表示的二元序列的相关功能。优化后的代码(参见附录)是对以字节表示的序列按算法 1 的步骤以字节处理为主实现扑克检测。所有的算法都采用标准 C 实现。

实验采用欧洲 estream 算法竞赛的速度测试模型的简化版本,该测试模型不仅在 estream 算法竞赛中采用,后续许多算法的性能评估也常采用该测试模型。具体来讲速度测试流程如下。1) 在被测试代码段的前后各设置一个时间计数器 T_S 和 T_F ;2) 将两个计时器之差 $T = T_F - T_s$ 作为这段代码的耗时;3) 重复 1) 和 2) 多次,为统计方便设定重复次数为奇数,记重复次数为 C,得到一系列的耗时值 $T[i]$, $1 \leqslant i \leqslant C$;4) 将统计得到的耗时值序列按从大到小的顺序排列得到 $T'[1] \geqslant T'[2] \geqslant \cdots \geqslant T'[C]$,当然也可按从小到大的顺序排列;5) 取新序列的中值 $T'[(C+1)/2]$ 作为本段代码的统计耗时值。w 为了保证测试结果的准确性,本测试模型中 1) 的时间计数器使用 CPU 频率计时器,直接调用汇编指令 RDTSC,在 Windows 环境下也可调用__

rdtsc( ) 函数,该指令或函数返回 CPU 时钟周期值,按现代 CPU 的时钟频率计算,此计数器可精确到纳秒级。两次 RDTSC 指令返回的时钟周期之差再除以 CPU 频率,即可得到以 s 为单位的耗时值。

原算法和优化算法(算法 1)对 1 000 个样本进行检测的性能统计结果见表 3。

表 3　算法性能对比

Table 3　The performance comparison of two algorithms

性能对比	原算法	算法 1
耗时/s	2.244	0.236
速度比	1:1	9.508:1

理论评估时只估算了最重要也最耗时的步骤 1),略去了后面的步骤的耗时。虽然步骤 2)的计算过程和原算法一样,但优化算法对步骤 3)也做了相应的优化。实验的结果表明,优化算法通过充分利用 CPU 字长一次处理多个比特,优化整合算法流程减少不必要的计算,精简统计量的计算和判断过程的技术方式和方法,的确可以显著地提升扑克检测的检测性能,且检测速度可提升 9.5 倍左右。

此外,算法性能提升显著的另外一个重要原因是扑克检测中的参数 m 取值较为特殊,$m=8$ 恰好是一个字节,$m=4$ 恰好是将一个字节拆分为两个“半字节”。这使得优化算法基于字节的处理方式得到了淋漓尽致的发挥。

6　结论

本文对我国随机性检测规范采用的扑克检测算法进行优化实现。通过充分利用 CPU 字长一次处理多个比特,优化整合算法流程减少不必要的计算,精简统计量的计算和判断过程的技术方式和方法,可以显著地提升扑克检测的检测性能,实验结果表明检测速度可提升 9.5 倍左右。因此,建议在实际检测中采用软件实现时使用本文提出的扑克检测快速实现方式以提高扑克检测的检测效率。NIST 和我国的随机性检测规范还有很多别的检测项,其中还有很多检测项可以做必要的性能优化,这将是今后工作的一个研究方向。另外,怎样利用并行化技术(如多线程技术、SIMD 指令、GPU 运算)快速实现这些检测项,也是今后研究的另一个方向。

附录　优化算法的代码

本节列出优化算法(算法 1)的标准 C 代码。其中 poker_test 函数为扑克测试优化后的功能实现函数,poker_get_statistics 函数为扑克检测中计算统计量的函数。

```
//扑克测试优化后代码
int poker_test(BYTE *p_u8, int n )
{
double v;
BYTE *p_bound = p_u8 + n / 8, d;
u32 c4[16] = {0}, c8[256] = {0};
while( p_u8 < p_bound )
{
d = *p_u8++;
c8[ d ]++;
c4[ d >> 4 ]++;
c4[ d & 0xf ]++;
}
//m = 8 时计算统计量
v = poker_get_statistics(8, n, c8);
if(v >= POKER_BOUND_M_8)
{
return -8;
}
//m = 4 时计算统计量
v = poker_get_statistics(4, n, c4);
if(v >= POKER_BOUND_M_4)
{
return -4;
}
return 1;
}
//扑克检测计算统计量,n 为序列比特长度
// m 为参数,p_ctr 为子序列频数
double poker_get_statistics(int m, int n, u32 *
p_ctr)
{
int i, blk_sz, pow_m;
double blk_sz_inv, sum;
sum = 0.0;
pow_m = 1 << m;
blk_sz = n / m;
blk_sz_inv = 1.0 / blk_sz;
for( i = 0; i < pow_m; i++ )
{
sum += p_ctr[i] * p_ctr[i] * blk_sz_inv;
}
return ( sum * pow_m ) - blk_sz;
}
```

参考文献：

[1] National Institute of Standards and Technology. NIST SP 800-22, A statistical test suite for random and pseudorandom number generators for cryptographic applications[S]. Revision 1a. Washington DC, USA: Information Technology Laboratory of National Institute of Standards and Technology, 2010.

[2] BSI AIS-20, AIS-30,. Application notes and interpretation of the scheme functionality classes and evaluation methodology for deterministic and physical random number generators [S]. Berlin, Germany: German Federal Office for Information Security, 2008.

[3] 随机性检测规范[S]. 中国北京：国家密码管理局, 2009.
Randomness test specification[S]. Beijing: National Cryptography Administration, 2009.

[4] 罗影, 刘冬梅, 康红娟. NIST 新分组密码工作模式及快速实现研究[J]. 通信技术, 2014, 47(9): 1066-1070.
LUO Ying, LIU Dongmei, KANG Hongjuan., NIST new block cipher modes of operation and their fast implementationoperation modes and their fast implementations of nist new block cipher[J]. Communications technology, 2014, 47(9): 1066-1070.

[5] 杨先伟, 康红娟. SM3 杂凑算法的软件快速实现研究[J]. 智能系统学报, 2015, 10(6): 9541-9597.
YANG Xianwei, KANG Hongjuan. Fast software implementation of SM3 hash algorithm[J]. CAAI transactions on intelligent systems, 2015, 10(6): 9541-9597.

[6] CCSA. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC specification[S]. Cedex, France: CCSA, 2011.

[7] 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.
FENG Xiutao. ZUC algorithm: 3GPP LTE international encryption standard[J]. Information security and communications privacy, 20112, 9(12): 45-46.

[8] 罗影, 张文科, 尹一桦, 等. 单比特频数检测和块内频数检测的快速实现研究[J]. 通信技术, 2015, 48(9): 1073-1077.
LUO Ying, ZHANG Wenke, YIN Yihua, et al. Fast Implementation of monobit frequency test and frequency test within a block[J]. Communications technology,. 2015, 48(9): 1073-1077.

[9] Edro M AALCOVER P M, GUILLAMóN A, RUIZ M D CAntonio G, et al. A new randomness test for bit sequences [J]. Informatica, 2013, 24(3): 339-356.

[10] KAMINSKY A. GPU parallel statistical and cube test analysis of the SHA-3 finalist candidate hash functions[EB/OL]. (2012-02-13)[2016-03-31]. http://www.cs.rit.edu/~ark/parallelcrypto/sha3test01/.

作者简介：

杨先伟，男，1980 年生，讲师，主要研究方向为密码学及通信与系统工程。

康红娟，女，1983 年生，硕士，工程师，主要研究方向为保密通信。

廖祖华，男，957 年生，教授，主要研究方向为人工智能、模糊与粗糙代数、广义逆理论及应用。主持省自然科学基金项目 1 项。发表学术论文 130 余篇，其中被 SCI 和 EI 检索 30 余篇。