

随机性测试的研究与实现

师国栋, 康 绯, 顾海文

(解放军信息工程大学信息工程学院, 郑州 450002)

摘 要: 介绍随机性测试方法的数理统计原理, 给出 16 种常见的随机性测试, 研究密码算法随机性测试的流程, 讨论测试 ID 的编排方法, 并用这些方法对欧洲加密标准——Camellia 算法进行随机性测试, 实验结果表明, 该算法 3 轮以上的缩减轮版本所产生的密文具有较高的随机性。

关键词: 随机性; 显著性水平; Camellia 算法

Research and Implementation of Randomness Tests

SHI Guo-dong, KANG Fei, GU Hai-wen

(School of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】 The principles of statistics about randomness tests are introduced. Sixteen common randomness tests are provided, as well as the processing flow of randomness tests towards cryptography. The layout methods of tests ID are discussed. These methods are used to test the standard of NESSIE——Camellia algorithm. Experimental results show high randomness on the cipher generated by reduced-Camellia at least three-round.

【Key words】 randomness; significance level; Camellia algorithm

随机数在密码学中的应用日益广泛。很多密码算法和协议中都要用到一些随机数, 如密钥、初始向量等。对一个密码算法来说, 其输出序列的随机性是其安全性的很重要的一个方面。因此, 随机性测试技术在密码学中占有很重要的作用。本文讨论一些常用的随机性测试方法, 并对欧洲加密标准——Camellia 算法进行随机性测试, 这些测试包括对 Camellia 算法完整轮、缩减轮的测试以及对几种特殊明文所产生的密文的测试。

1 随机性测试的数学基础

测试序列的随机性, 实质上是检验其是否是真随机的或与真随机的差距, 假设检验^[1]是随机性测试技术的基本理论。

在检验时, 提出一个待检验的假设, 称为原假设或零假设, 记作 H_0 ; 还有个与原假设相反的称备择假设, 记作 H_1 。如在本文中, 原假设 H_0 : 序列是随机的; 备择假设 H_1 : 序列是非随机的。对每个检验来说, 其检验结果是否接受原假设也就意味着待检数据是否是随机的。具体步骤如下:

Step1 确定原假设 H_0 和备择假设 H_1 ;

Step2 根据 H_0 的内容选取合适的统计量 X , 并确定其分布;

Step3 按给定的显著性水平 α , 查统计量所对应分布的分位数表, 找出临界值, 确定拒绝域;

Step4 将由样本计算出的统计量的值与查得的临界值进行比较, 做出判断, 即: 若统计量的值落入了拒绝域中, 则拒绝 H_0 , 否则接受 H_0 。

此外, 假设检验会产生 2 类检验错误类型, 如表 1 所示。

表 1 检验错误类型

H_0	判断结果	犯错误的概率
真	接受/正确	0
	拒绝/犯第 1 类错误	α
假	接受/犯第 2 类错误	β
	拒绝/正确	0

在实际应用中, 衡量随机性的方法通常有 2 种: 门限值法和 P -value 值法。这里以测试统计量 X 服从 χ^2 分布为例来说明。

(1) 门限值法, 先做出 χ^2 分布的概率密度曲线, 如图 1 所示, 根据给定的显著性水平 α (图 1 中的阴影部分面积) 查出它的 α 分位点 χ^2_α , 然后再拿统计量 X 与 χ^2_α 进行比较, 确定是拒绝还是接受原假设。

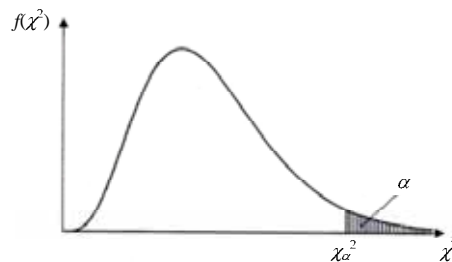


图 1 χ^2 分布的概率密度曲线及 α 分位点

(2) P -value 值法, 同样作出 χ^2 分布的概率密度曲线, 先求出统计量 X , 然后计算从 X 到无穷的积分, 将积分结果 (即 P -value 值, 图中的阴影部分面积) 与 α 进行比较, 进而确定拒绝还是接受。本文中讨论的随机性测试即是通过选取的测试统计量来计算 P -value 值, 将 P -value 值作为接受原假设的强度, 其含义是: 真随机数的随机性比待测序列差的概率。如果其值为 1 则是完全真随机的, 值为 0 则是完全非随机的。对于显著性水平 α , 如果 P -value $\geq \alpha$, 那么原假设被接受, 序列是随机的; 反之被拒绝, 序列是非随机的。参数 α 也即

基金项目: 国家 “863” 计划基金资助项目 (2007AA01Z471)

作者简介: 师国栋 (1984 -), 男, 硕士, 主研方向: 信息安全; 康 绯, 副教授; 顾海文, 硕士

收稿日期: 2009-05-30 **E-mail:** shiguodong_happy@163.com

是表 1 中错误类型 1 的概率,一般地, α 的取值范围是[0.001, 0.01], 如图 2 所示。

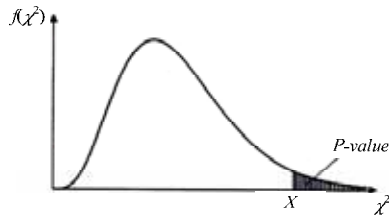


图 2 χ^2 分布的概率密度曲线及 P -value 值

2 随机性测试

目前,已公开的随机性测试方法不下百种^[2]。这些测试方法为测试序列的不同特性而设计,主要是基于序列特性的不同侧重点,它们中多数使用的原理没有显著差别,而且有的方法往往还蕴涵于另外的方法之中。因此,在选择随机性测试方法时既要考虑到能够测试序列随机特性的各个方面,又要兼顾测试的效率,根据这种思想,并参考相关文献^[3],最终确定 16 种测试方法。

2.1 测试内容

本文讨论的 16 种随机性测试分别为频数测试、块内频数测试、游程测试、块内最长连续“1”测试、矩阵秩的测试、离散傅里叶变换测试、非重叠模板匹配测试、重叠模板匹配测试、通用统计测试、压缩测试、线性复杂度测试、连续性测试、近似熵测试、部分和测试、随机游走测试和随机游走变量测试等^[3]。下面对各测试方法作简要介绍:

(1)频数测试:目的是确定二进制序列中,“0”和“1”数目是否如真随机序列那样近似相等。如果是,则序列是随机的。

(2)块内频数测试:目的是确定在待测序列中,所有非重叠的 M -bit 块内的“0”和“1”的数目是否表现为随机分布。如果是,则序列是随机的。

(3)游程测试:目的是确定待测序列中,各种特定长度的“0”和“1”的游程数目是否如真随机序列期望的那样。如果是,则序列是随机的。

(4)块内最长连续“1”测试:目的是确定待测序列中,最长连“1”串的长度是否与真随机序列中最长连“1”串的长度近似一致。如果是,则序列是随机的。

(5)矩阵秩的测试:目的是检测待测序列中,固定长度子序列的线性相关性。如果线性相关性较小,则序列是随机的。

(6)离散傅里叶变换测试:目的是通过检测待测序列的周期性质,并与真随机序列周期性质相比较,通过它们之间的偏离程度来确定待测序列随机性。如果偏离程度较小,序列是随机的。

(7)非重叠模板匹配测试:目的是检测待测序列中,子序列是否与太多的非周期模板相匹配。太多就意味着待测序列是非随机的。

(8)重叠模板匹配测试:目的是统计待测序列中,特定长度的连续“1”的数目,是否与真随机序列的情况偏离太大。太大是非随机的。

(9)通用统计测试:目的是检测待测序列是否能在信息不丢失的情况下被明显压缩。一个不可被明显压缩的序列是随机的。

(10)压缩测试:目的是确定待测序列能被压缩的程度,如果能被显著压缩,说明不是随机序列。

(11)线性复杂度测试:目的是确定待测序列是否足够复杂,如果是,则序列是随机的。

(12)连续性测试:目的是确定待测序列所有可能的 m -bit 组合子串出现的次数是否与真随机序列中的情况近似相同,如果是,则序列是随机的。

(13)近似熵测试:目的是通过比较 m -bit 串与 $(m-1)$ -bit 串在待测序列中出现的频度,再与正态分布的序列中的情况相对比,从而确定随机性。

(14)部分和测试:目的确定待测序列中的部分和是否太大或太小。太大或太小都是非随机的。

(15)随机游走测试:目的是确定在一个随机游程中,某个特定状态出现的次数是否远远超过真随机序列中的情况。如果是,则序列是非随机的。

(16)随机游走变量测试:目的是检测待测序列中,某一特定状态在一个游程中出现次数与真随机序列的偏离程度。如果偏离程度较大,则序列是非随机的。

2.2 密码算法的随机性测试

如果要测试某个密码算法的性能好坏,一般的方法是:选定由加密算法产生的若干个序列进行随机性测试,产生对应的 p 值集合,根据显著性水平 α 来判断 p 值是否通过测试,并通过考察 p 值通过率及其分布特性检验加密算法的好坏。

(1)序列通过测试的比例

如果选取显著性水平为 0.01,那么将允许在 100 个序列中有一个序列不通过,也就是说通过率不得低于 $1-0.01=0.99$ 。那么就可以这样处理数据:测试 1 000 个序列,有 996 个通过,显著性水平为 0.01,此时通过率 $=996/1\ 000=0.996$,显然 $0.996 > (1-0.01)=0.99$,就可以认为通过了随机性测试。

但事实上,对一个密码算法来说,要其产生的序列以这样的比例通过每一个检验还是比较困难,也是不全面的。因此,可以引用统计学中的置信区间理论,它定义为:

$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$, 其中, $\hat{p}=1-\alpha$; α 为显著性水平; m 为序列个数。如果 p 值通过率落到此区间之外,那么将拒绝原假设,序列是非随机的。以本文提到的 $\alpha=0.01$ 为例,它的置信区间就是

$$(1-0.01) \pm 3\sqrt{\frac{(1-0.01)0.01}{300}} = 0.99 \pm 0.017234$$

p 值通过率应高于 0.972 766。另外,选取的序列个数越多,判断越准确。

(2) P -value 值的分布

因为真随机序列的 P -value 值服从均匀分布,所以可以根据 P -value 值分布的均匀性来描述随机性。由于 P -value 是属于 0~1 之间的数,因此将 0~1 划分为 10 个区间: [0.0,0.1), [0.1,0.2), [0.2,0.3), [0.3,0.4), [0.4,0.5), [0.5,0.6), [0.6,0.7), [0.7,0.8), [0.8,0.9), [0.9,1.0], 当序列足够多时, P -value 应平均分布在这 10 个区间内,统计其在每个区间出现的频数,记为 v_i ,若序列个数为 m ,则有 $v_1+v_2+\dots+v_{10}=m$,计算统计量

$$\chi^2 = \sum_{i=1}^{10} \frac{(v_i - m/10)^2}{m/10}$$

再计算 $P\text{-value}_T = \text{igamc}(9/2, \chi^2/2)$ 。如果 $P\text{-value}_T > 0.001$,那么 P -value 值服从均匀分布,即序列是随机的,否则是非随机的。

2.3 测试 ID 号的选取

由于这些测试中某些测试会返回多个 P -value 值,而每

个 P -value 值都看作一个测试类别，因此本测试工具选取测试 ID 号的情况如表 2 所示。

表 2 测试 ID

测试方法	P 值	测试 ID
频数	1	1
块内频数	1	2
游程测试	1	3
块内最长连续“1”	1	4
矩阵秩	1	5
离散傅立叶变换	1	6
非重叠模板匹配	148	7~154
重叠模板匹配	1	155
通用统计	1	156
压缩	1	157
线性复杂度	1	158
连续性	2	159~160
近似熵	1	161
部分和	2	162~163
随机游走	8	164~171
随机游走变量	18	172~189

2.4 随机性测试工程的实现

在工程实现方面，建立一个测试类和一个返回结果的结构体。一个随机性测试类如下：

```
class RandTest
{private:
byte *m_pdata;           //待测数据
int m_dataLen;           //待测数据长度
tchar *m_errorMessage;   //错误消息
tchar *m_testName;       //测试名称
public:
//各种测试方法
RandTest();
RandTest(byte *a, int dataLen);
~RandTest();
bool FrequencyTest();
bool BlockFrequencyTest();
bool Runs();
bool LongestRunOfOnes();
bool Rank();
bool DiscreteFourierTransform();
bool NonOverlappingTemplateMatchings();
bool OverlappingTemplateMatchings();
bool Universal();
bool LempelZivCompression();
bool CommonLempelZivCompression();
bool LinearComplexity();
bool Serial();
bool ApproximateEntropy();
bool CumulativeSums();
bool RandomExcursions();
bool RandomExcursionsVariant();
tchar *GetMessageFromError();};
一个返回结果的结构体如下：
struct testRandRes
{dword NameID:8; //用 8 个 bit 位标识 NameID
dword TestID:24; //用 24 个 bit 位标识 TestID
bool Res; //返回测试结果“1”或“0”
double pValue; //返回 pValue 值
double confidence; //返回可信度};
```

3 Camellia 算法的随机性测试

3.1 选取的样本类型

本测试所选取的样本类型有以下 4 种：

(1)Camellia 的完整轮加密密文：给定一个随机 128 bit 的密钥、128 bit 的初始化向量和 8 192 个 128 bit 的全“0”明文块，在 CBC 模式下利用 Camellia 算法的完整轮版本进行加密，得到 1 048 576 bit 的序列。更换随机密钥，产生 300 组这样的序列。

(2)Camellia 的缩减轮加密密文：给定一个随机的 128 bit 的密钥、128 bit 的初始化向量和 8 192 个 128 bit 的全“0”明文块，在 CBC 模式下利用 Camellia 算法的缩减轮进行加密得到 1 048 576 bit 的二进制序列。更换随机密钥，产生 300 组这样的序列。这里特殊说明一下，对于 Camellia 算法的缩减轮版本，始终保留最后一轮，并把它看作 0.5 轮，本次测试测试了 0.5 轮到 6+0.5 轮的版本。

(3)低密度明文：测试数据由低密度明文产生，共 300 个序列，每个序列包含 8 257 个在 ECB 模式下加密得到的密文分组。在明文的选取上，第 1 个密文分组是由全“0”128 bit 的明文分组加密产生，第 2 个~第 129 个密文分组是由汉明重量为 1 的 128 bit 的明文分组加密产生，第 130 个~第 8 257 个密文分组是由汉明重量为 2 的 128 bit 的明文分组加密产生。更换随机密钥，产生 300 组这样的序列。

(4)高密度明文：测试数据由高密度明文产生，共 300 个序列，每个序列包含 8 257 个在 ECB 模式下加密得到的密文分组。在明文的选取上，第 1 个密文分组是由全“1”128 bit 的明文分组加密产生，第 2 个~第 129 个密文分组是由汉明重量为 127 的 128 bit 的明文分组加密产生，第 130 个~第 8 257 个密文分组是由汉明重量为 126 的 128 bit 的明文分组加密产生。更换随机密钥，产生 300 组这样的序列。

其中，低密度明文和高密度明文采用的加密算法是 Camellia 的完整轮版本。

3.2 测试结果

在 Visual Studio 2005 环境下，对每组样本序列进行测试，考察 p 值通过测试的比例及其均匀性，结果如表 3 所示。

表 3 测试结果

样本	P 值通过率不足的 测试 ID 数	P 值非均匀的 测试 ID 数	2 项均达标的 测试 ID 数
0.5 轮	189	189	0
1.5 轮	124	118	61
2.5 轮	68	72	124
3.5 轮	6	13	162
4.5 轮	1	3	189
5.5 轮	0	0	189
6.5 轮	0	0	189
完整轮	0	0	189
低密度	0	0	189
高密度	0	0	189

通过测试发现：随着加密轮数的增加，输出密文的随机性越来越好，从第 3 轮开始则表现出较高的随机性，Camellia 的完整轮版本则表现出完全的随机性，通过测试由高低密度明文产生的密文来看，它们的随机性并没有差异。

4 结束语

本文讨论 16 种常用的随机性测试方法，从不同角度考察二进制序列的随机特性，涵盖了序列随机性的各个方面，具有重要的理论和实际意义，另外，还实现一个随机性测试工具集，并对 Camellia 算法进行了随机性测试，得到具有一定应用价值的结论。

(下转第 150 页)