关于随机数的研究

黄瑞轩　PB20111686

1 引入.

日常生活中经常要用到随机数。有时作为活动组织方，需要通过获取随机数的方式来产生中奖名单。对于一个五选三的随机抽奖项目，考虑如下模型。

$(*)$设待抽取名单为$\{a_0,a_1,a_2,a_3,a_4\}$，A和B是两个正常的人。B设置了5个不同的映射$f_i(i=1,2,...,5)$，f_i能将$\{l_1,l_2,l_3,l_4,l_5\}$这个数集映射到$\{a_0,a_1,a_2,a_3,a_4\}$，且随着i的不同，映射的方式也不相同。注意，A对B所设置的内容一无所知。A进行抽奖时首先要自行选定一个i，方法为在键盘上敲击1~5中的一个数字，并接着输入三个无顺序的数字$\{l_1',l_2',l_3'\}\subseteq\{l_1,l_2,l_3,l_4,l_5\}$，则屏幕将显示结果$\{f_i(l_1'),f_i(l_2'),f_i(l_3')\}\subseteq\{a_0,a_1,a_2,a_3,a_4\}$。

这一例子引出了计算机获得随机数的基本模型。用图可表示为：

图 1　计算机获得随机数的基本模型

在$(*)$中，B本人以及B设置的五个映射构成了这一随机数生成过程的随机样本，也即上图所示的熵池，这一概念会在后面的研究中予以解释。

2 定义随机数.

如何定义“随机数”？根据密码学原理，随机数的随机性检验可以分为如下三个标准：

Definition. 随机性

统计学伪随机性	统计学伪随机性指的是在给定的随机比特流样本中，0的数量大致等于1的数量，同理，“10”,“01”,“00”,“11”四者数量大致相等。类似的标准被称为统计学随机性。满足这类要求的数字在人类“一眼看上去”是随机的。
密码学安全伪随机性	给定随机样本的一部分和随机算法，不能有效的演算出随机样本的剩余部分。
真随机性	产生的随机样本不可重现。

相应地，随机数也分为三类：

Definition. 随机数

伪随机数	只满足统计学伪随机性的随机数。
密码学安全伪随机数	同时满足伪随机性和密码学安全伪随机性的随机数。可以通过密码学安全伪随机数生成器计算得出。
真随机数	产生的随机样本不可重现。

3 生成随机数的方法.

我们可以通过扩大随机样本来增加随机样本的不可预测性。这一扩大的随机样本被称作熵池。这熵池中可以是计算机从外界环境捕获的噪音，也可以是单位时间落在传感器上的颗粒数，甚至可以是衰变的信息（衰变是完全不可预测的过程）。计算机从熵池中捕获数据，通过特定算法处理数据，就可以生成随机数呈现给用户。 同任何优化问题一样，不可预测性越高，技术上实现的难度就越高。近年来已经有利用量子力学原理制造的随机数发生器。某些量子物理过程所产生的随机性是完全真随机的，如量子态的坍缩过程。

下面是一种计算机上随机数的生成方法及其优化方案。

给计算机一个初始的值seed，设产生的第n个随机数为a_n，随机函数$f(x)$。则有$a_{n+1} = f(a_n)$，$a_1 = f(\text{seed})$。比如使用如下代码：

```
1.  static unsigned seed = u;
2.  int rand(void) {
3.      return seed * 1234u + 5678u;
4.  }
```

如果需要获得0~_RAND_MAX 之间的随机数，将所得结果对_RAND_MAX取模即可。

这种随机数产生方式有两方面安全隐患：

1° 需要用户指定初值（否则将为default值），一次测试中不能保证用户输入初值时的公平性；

2° 程序极易被反编译，将直接暴露随机函数$f(x)$。

如果改进上述过程，从熵池中选择初值，例如利用time()函数获取当前时间，代码如下

```
1.  int seed(void) {
2.      return ((unsigned) time(&t));
3.  }
```

或者利用鼠标在下一时刻的平均移动方位。可以认为解决了 1° 。

为了解决上述 2°，需要使随机函数$f(x)$变得不可预测。什么是不可预测？

Definition. 可预测性 如果一个函数$f(x)$是可预测的，那么说明存在一个函数g，能在多项式时间内通过$f(x)$产生的部分信息$INFO_{1,\dots,i}$，获得剩余信息$INFO_{i+1}$。即：

$$\exists g, \forall i, g(INFO_{1,\dots,i}) = INFO_{i+1}$$

迭代下去，就可以获得全部的信息。一旦攻击者截获了部分明文与部分密文，将其进行异或(xor)运算就可以获得$f(x)$的信息，从而得到整段明文。

解决了 1° 和 2°，利用统计测试的方法可以检测一个二进制串是否随机。

设一个统计测试算法A，输入为一个n位二进制字符串，输出其是否随机（True or False）。可表示为：

$$x = \{0,1\}^n, A(x) = \begin{cases} 0, \text{if } x \text{ is not random} \\ 1, \text{if } x \text{ is random} \end{cases}$$

一种可能的算法如下：

$$A(x) = 1 \Leftrightarrow |\text{card}\{\text{bit}(x) = 0\} - \text{card}\{\text{bit}(x) = 1\}| \leq N(\text{for example}, \sqrt{n})$$

有意义的是“统计测试A把$f(x)$输出的伪随机数列认为是随机的”与“统计测试A把真随机数列Random_n认为是随机的”的概率之差。记为

$$\text{Advantage}\big(A, f(x)\big) = P(A(\text{Random}_n) = 1) - P(A(INFO) = 1)$$

显然，这个差值越小越好。如果$\text{Advantage}\big(A, f(x)\big)$小到可以忽略，则可以认为生成了密码学安全的伪随机数。实际上没有数学证明表示密码学安全的伪随机数生成器是确实存在的。其存在性证明涉及到 P 和 NP 的数学难题，已经超出了研究计算机与随机数的范围，这里不再赘述。

一个可以生成好的伪随机数列的方法是梅森旋转算法，代码如下：

```
1.  //创建一个长度为 624 的数组来存储发生器的状态
2.  int[0..623] MT
3.  int index = 0
4.  //初始化产生器，种子作为首项内容
5.  function initialize_generator(int seed) {
6.      i := 0
7.      MT[0] := seed
8.      for i from 1 to 623 { // 走访剩下的每个元素
9.          MT[i] := last 32 bits of(1812433253 * (MT[i-1] xor (right shift by 30 bits(MT[i-
    1]))) + i) // 1812433253 == 0x6c078965
10.     }
```

```
}
// Extract a tempered pseudorandom number based on the index-th value,
// calling generate_numbers() every 624 numbers
function extract_number() {
    if index == 0 {
        generate_numbers()
    }
    int y := MT[index]
    y := y xor (right shift by 11 bits(y))
    y := y xor (left shift by 7 bits(y) and (2636928640)) // 2636928640 == 0x9d2c5680
    y := y xor (left shift by 15 bits(y) and (4022730752)) // 4022730752 == 0xefc60000
    y := y xor (right shift by 18 bits(y))

    index := (index + 1) mod 624
    return y
}
// Generate an array of 624 untempered numbers
function generate_numbers() {
    for i from 0 to 623 {
        int y := (MT[i] & 0x80000000)                       // bit 31 (32nd bit) of MT[i]
                        + (MT[(i+1) mod 624] & 0x7fffffff)   // bits 0-
30 (first 31 bits) of MT[...]
        MT[i] := MT[(i + 397) mod 624] xor (right shift by 1 bit(y))
        if (y mod 2) != 0 { // y is odd
            MT[i] := MT[i] xor (2567483615) // 2567483615 == 0x9908b0df
        }
    }
}
```

4 比较生成随机数方法的优劣.

生成随机数的方法众多，如何比较它们之间的优劣？

直观上看，可以用随机数来填充一个位图。例子：

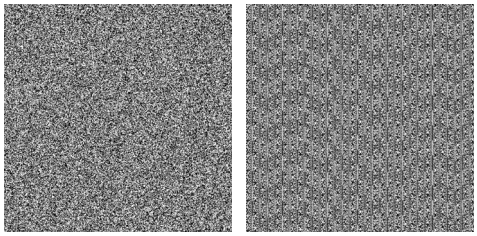

图 2　用两个随机数生成方法填充的位图

图 2 左侧是用 C#的System. Random类生成的随机数填充的位图，图 2 右侧则是用 php 的 rand 函数生成的随机数填充的位图。显然，后者在直观上更具有可重复性，随机性差。

避开直观，德国联邦信息安全办公室给出了评判的四标准：

K1 — A sequence of random numbers with a low probability of containing identical consecutive elements.

K2 — A sequence of numbers which is indistinguishable from 'true random' numbers according to specified statistical tests.

K3 — It should be impossible for any attacker (for all practical purposes) to calculate, or otherwise guess, from any given sub-sequence, any previous or future values in the sequence, nor any inner state of the generator.

K4 — It should be impossible, for all practical purposes, for an attacker to calculate, or guess from an inner state of the generator, any previous numbers in the sequence or any previous inner generator states.

与此规则相符，存在一系列检验方法——经验的或理论的。线性复杂度测试、连续性测试等算法目前都被用于开发在线的检测工具，用户可以利用检测工具来检测计算机生成的随机数是否容易被测试算法攻破。

参考文献.

1. https://zh.wikipedia.org/wiki/%E9%9A%8F%E6%9C%BA%E6%95%B0
2. Jonathan Katz; Yehuda Lindell. *Introduction to Modern Cryptography*: Principles and Protocols
3. https://iqe.pku.edu.cn/dht/lzgxylzxx/yjfx/lzsjsfsq/index.htm
4. https://www.bsi.bund.de/cae/servlet/contentblob/478152/publicationFile/30552/ais20e_pdf.pdf