第 57 卷 第 9 期 2008 年 9 月
1000-3290/2008/57(09)/5413-06

物 理 学 报
ACTA PHYSICA SINICA

Vol.57,No.9,September,2008

基于鼠标轨迹和混沌系统的真随机数产生器研究*

周 庆† 胡 月 廖晓峰

(重庆大学计算机学院,重庆 400030)

(2007 年 12 月 2 日收到;2008 年 1 月 27 日收到修改稿)

提出了一种基于鼠标轨迹的真随机数产生器,并对该类产生器的优缺点、总体设计和基本技术进行了研究.为了消除相同用户鼠标轨迹中存在的相似性,利用混沌系统的敏感性,分别采用图像加密算法和 Hash 函数两种方法对鼠标轨迹进行后处理.大量严格的测试和实验表明,改进的基于混沌 Hash 函数的真随机数产生器具有安全、快速、方便和廉价的优点,可以在个人电脑上实际使用.

关键词:混沌,真随机数产生器,鼠标轨迹

PACC:0545,0540

1. 引 言

随机数在密码学领域中具有非常重要的作用.对称加密算法,无论是块加密方式还是序列密码方式,其密钥必须随机产生,公开加密算法在产生密钥对时也需要较大的随机数.此外许多数字签名算法和密码协议也要求使用随机数[1].随着电子商务安全在日常生活中的普及,PC 机(Personal Computer,个人电脑)用户对安全随机数的需求也日益增加.

随机数由两种类型的产生器生成,即真随机数产生器(true random number generators, TRNG)和伪随机数产生器(pseudo random number generators, PRNG).TRNG 建立在一种称为熵源(entropy source)的不确定性源的基础上,如热力学噪声、空气噪声、核衰变等物理现象,对这些现象采集的数据再进行后处理(post-process),即可得到随机数.而 PRNG 则根据输入的随机数种子,采用确定性的算法生成多个随机数.TRNG 通常比 PRNG 更安全,因此在安全级别较高的应用中,都会采用 TRNG.但是在 PC 机上使用 TRNG 一般需要采用额外的电路或设备,因此从成本和便利性两方面阻碍了其在 PC 机上的应用.

鼠标是 PC 机上使用最广泛的输入设备之一,几乎所有的 PC 机都配备了鼠标,并且无论中央处理器、主板或操作系统采用哪个厂商的产品,PC 机都能提供对鼠标信号的处理.假设用户只需要简单地移动鼠标就可产生安全的随机数,那么这种 TRNG 与其他 TRNG 相比具备很多优点.首先用户毋须购买额外的芯片,从而降低了成本;其次这种 TRNG 可在任意一台配置了鼠标的 PC 机上运行,使用非常方便;此外这种 TRNG 独立于处理器、主板和操作系统生产厂商,具有很强的通用性.最后用户使用这种 TRNG 也会更放心,因为系统产生随机数必须经过用户的参与和控制.相反的,对于一个伪随机数产生器,如果它的种子被黑客窃取,则其产生的随机数就很容易被预测.

然而鼠标作为熵源有一个重要的缺陷,即同一用户的鼠标轨迹存在许多相似性或模式.如果在后处理时不能有效消除这种相似性,就不能产生具有良好统计性能的随机数.但是从另一个角度考虑,即使对于同一用户,其产生的鼠标轨迹也不可能完全相同.利用这一事实,如果后处理的输出对原始轨迹的变化足够敏感,我们或许能消除鼠标轨迹中存在的模式.

众所周知,混沌系统对初始状态和系统参数具有极高的敏感性.不仅如此,混沌系统还具有伪随机性、遍历性等特点,因此基于混沌系统的各种加密技术被广泛研究,包括 Hash 函数[2—6]、图像加密算法[7—9]、分组加密算法[10,11]、流密码算法[12]和公开加

* 国家自然科学基金(批准号:60573047,60703035)和新世纪优秀人才支持计划资助的课题.

† E-mail:tzhou@cqu.edu.cn

密算法[13]等技术.

本文提出了一种基于鼠标移动轨迹的 TRNG 的设计方案.该方案具有通用性强、使用方便和成本低等优点,适合在任意一台配备了鼠标的 PC 机上使用.在对鼠标轨迹进行后处理时,分别尝试了基于混沌的图像加密算法和 Hash 函数,有效地去除了鼠标轨迹中存在的模式,产生的随机数序列均通过了美国 NIST 建议的 16 种随机数测试[1].

2. 基于鼠标轨迹的 TRNG 设计

基于鼠标轨迹的 TRNG 的总体设计包括三个过程:

1)模/数转换:将鼠标移动的模拟信号转换为 PC 机可处理的数字信号;

2)后处理:采用适当的处理方法去除信号中存在的模式;

3)2D/1D 转换:将二维信号转换成一维信号.

其中第个 3 过程,即 2D/1D 转换可能在后处理之前或之后进行.经过以上三个过程,就实现从鼠标移动轨迹到一个 256 比特随机数的转换.下面我们就这三个过程进行讨论.

2.1. 模/数转换

鼠标移动实质上是一种物理运动,其轨迹是连续的模拟信号,而 PC 机只能处理离散的数字信号,因此在进一步处理前,首先要实现鼠标移动轨迹的模/数转换.由于所有配置了鼠标的 PC 机都能向应用程序提供鼠标信息,这一转换过程很容易编程实现.程序可以直接使用 PC 机底层提供的接口,也可以间接地调用各种软件开发包提供的应用程序接口.通过转换,鼠标轨迹被采样成平面上的一系列点,其坐标从 1 到几千不等.如果用图像来表示这些坐标,连续的鼠标轨迹就被转换成了离散的二维数字信号.图 1 显示了三个用户的鼠标轨迹经过模/数转换后的结果.

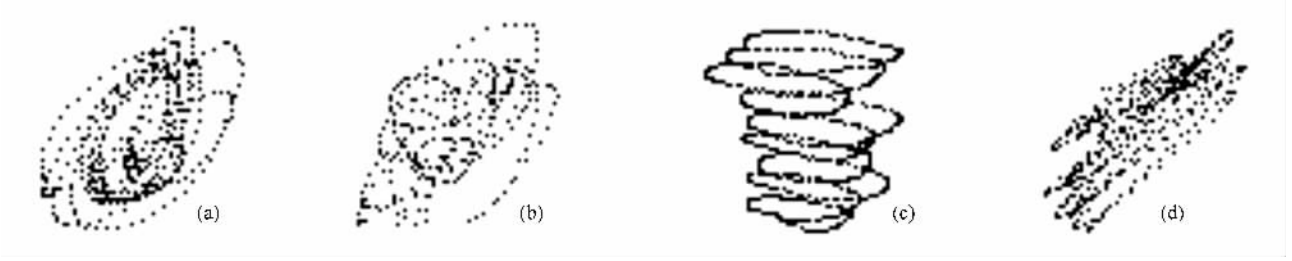

图 1 各用户鼠标运动轨迹抽样图 (a)用户 A 第 1 个轨迹;(b)用户 A 第 2 个轨迹;(c)用户 B 第 1 个轨迹;(d)用户 C 第 1 个轨迹

2.2. 后处理

如前所述,同一用户产生的鼠标轨迹中可能存在许多相似之处.从图 1 中三个用户的鼠标轨迹中可以看出这一现象.为了消除鼠标轨迹中存在的模式,必须使处理后的输出对鼠标轨迹非常敏感.因此可借助混沌系统对初始状态和系统参数极高的敏感性对鼠标轨迹进行后处理.我们发现在众多的混沌加密技术中[2—14],有两种技术可满足这一要求,即图像加密算法和 Hash 函数.

采用图像加密算法是最直观的方法,因为鼠标轨迹采样后的结果可直接转换成图像.另一方面,目前已有许多研究者提出了各种图像加密算法[7—9],实验结果表明这些算法都具有非常好的随机性和对明文图像极高的敏感性.因此使用图像加密算法对鼠标轨迹进行处理很可能产生统计性能好的随机数.

Hash 函数是另一种可用于后处理的备选技术.由于 Hash 函数必须具有很强的抗碰撞能力,因此好的 Hash 函数输出的 Hash 值应服从均匀分布且对明文具有很高的敏感性,因此基于 Hash 函数的 TRNG 可能具有很好的统计特性.Hash 函数的另一个优点是运算速度比加密算法快,这也是设计 TRNG 时应考虑的重要指标之一.

本文采用两种已知的图像加密算法和 Hash 函数分别对鼠标轨迹进行后处理,具体过程在第 3 节描述.

2.3. 2D/1D 转换

由于鼠标的轨迹是二维信号,而最终的输出信号是一维的随机数,因此需要将二维信号转换成一维信号.对于采用图像加密算法的后处理方法,由于图像加密算法的输入是二维信号,因此 2D/1D 转换应在加密后进行.加密后密文仍为二值图像,其转换

方法如下：

1）将图像分成 256 个相等的块；

2）按从上到下，从左到右的顺序扫描每一块，对第 i 块，计算

$$s_i = \mathrm{mod}\left(\sum_{j,k} S_{i,j,k}, 2\right), \tag{1}$$

其中 $S_{i,j,k}$ 表示第 i 块中第 j 行第 k 列的象素，$i = 1, \cdots, 256$，mod 为模运算；

3）输出 256 位的随机向量 $r = (s_1, \cdots, s_{256})$.

图 2 显示了针对图像加密算法进行的 2D/1D 转换方式.

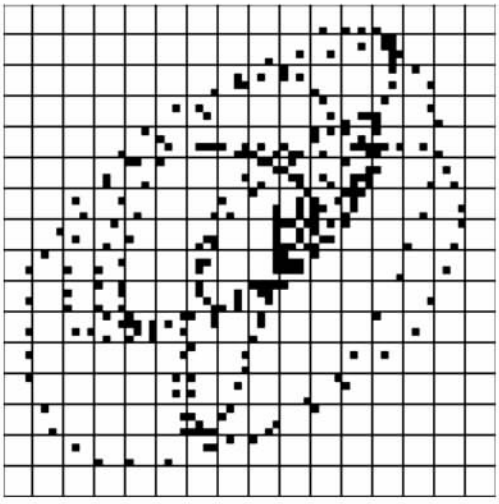

图 2　将鼠标轨迹划分成 256 块

对于采用 Hash 函数的后处理方法，由于 Hash 函数的输入是一维信号，因此 2D/1D 转换应在 Hash 函数之前进行，其转换方法如下：

1）得到相邻两个点的坐标 $A(x_i, y_i)$ 和 $B(x_{i+1}, y_{i+1})$；

2）计算两个点在水平和竖直方向的距离 $Y_i = |y_{i+1} - y_i|$，$X_i = |x_{i+1} - x_i|$；

3）计算两个点的夹角 θ_i 和距离 r_i：

$$\theta_i = \begin{cases} \pi/2 & x_{i+1} = x, \\ \arctan\left(\dfrac{Y_i}{X_i}\right), & \text{其他}, \end{cases} \tag{2}$$

$$r_i = \sqrt{X_i^2 + Y_i^2}; \tag{3}$$

4）重复以上步骤，对鼠标轨迹中的前 129 个相邻的点进行处理，得到 256 个数，即 $\theta_1, \cdots, \theta_{128}, r_1, \cdots, r_{128}$；

5）将 256 个数规范化到区间[0，1]上，

$$\theta_i' = \frac{\theta_i}{\pi/2}, r_i' = \frac{r_i}{r_{\max}}, \tag{4}$$

其中 $r_{\max}$ 是 r_i 中的最大值.

通过以上 5 个步骤，我们可由鼠标轨迹得到 256 个介于 0 和 1 之间的小数. 图 3 显示了针对 Hash 函数进行的 2D/1D 转换方式.

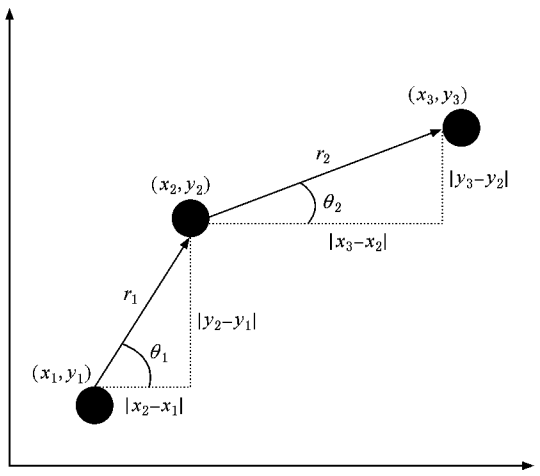

图 3　将相邻鼠标轨迹点映射为距离和夹角

3. 两种基于混沌加密技术的 TRNG

在这一节中我们采用两种混沌加密技术，即基于混沌系统的图像加密算法和 Hash 函数来实现 TRNG. 对每种加密技术，我们分别选用一个已知的算法来实现.

3.1. 基于图像加密技术的 TRNG

采用图像加密技术生成随机数的步骤如下：

1）将连续的鼠标运动轨迹进行模/数转换，得到一系列的坐标点；

2）将坐标线性映射成区间[1，64]上的整数，并由这些坐标生成 64×64 的二值图像；

3）对二值图像进行加密，得到 64×64 的密文图像；

4）用 2.3 节介绍的方法将密文图像转换成 256 位的随机数.

在以上过程中最关键的是步骤 3. 本文采用文献[9]中提出的 MASK 图像加密算法. MASK 图像加密算法包括 9 轮加密，每轮由 M，A，S 和 K 四个变换组成，描述如下：

1）M 变换

M 变换定义为

$$c_{ij} = a_{ij} + \sum_i \sum_j a_{ij}, \tag{5}$$

其中 a_{ij} 为块内第 i 行第 j 列的像素，c_{ij} 为对应的密文像素. 注意等式(5) 中的加法均为整数的按位异或运算.

2) A 变换

A 变换定义为

$$c_{ij} = a_{ij} + k_{ij}, \tag{6}$$

即将明文像素与对应的密钥像素相加，注意这里的加法与(5)式的含义相同.

3) S 变换

即 S 盒替换运算，这里采用文献[14]定义的 S 盒. 文献[14]表明该 S 盒满足安全 S 盒所要求的 6 个特性. 注意的是这里的图像是二值图像，因此在进行 S 盒替换时应将相邻的 8 位连起来作为 S 盒的输入.

4) K 变换

即 Kolmogorov 流，又称为广义 Baker 映射，文献[8]定义该映射的离散形式如下：

$$T_{\mathrm{no}}(x,y) = (p_s(x - F_s) + (y \bmod p_s), F_s + (y \operatorname{div} p_s)), \tag{7}$$

$$F_s = \begin{cases} 0, & s = 1 \\ n_1 + n_2 + \cdots + n_{s-1}, & s = 2,3,\cdots,t-1, \end{cases} \tag{8}$$

其中 n_s 表示第 s 个图像块的行数，$p_s = N/n_s$，N 为图像的总行数. K 变换实现了对图像像素的置换操作，当 n_s 的各个值固定时，K 变换可通过查表实现. 文献[9] 已证明，只需 3 轮变换，原图像中的任一像素的改变将扩散到整个图像.

在以上四个变换中 M，A 和 S 变换在图像的一行上进行操作，而 K 变换则对整个图像进行操作. 关于 MASK 算法更详细的内容可参见文献[9]. 由于整个加密过程只使用了异或和查表操作，图像加密的速度非常快. 我们将 MASK 算法产生的随机序列输入到 NIST 测试软件进行测试，其结果在第 4 节给出.

3.2. 基于 Hash 函数的 TRNG

采用 Hash 函数生成随机数的步骤如下：

1) 将连续的鼠标运动进行模/数转换，得到一系列的坐标点；

2) 使用 2.3 节介绍的方法将坐标点转换成 128 个 θ'_i 和 128 个 r'；

3) 使用 Hash 函数对 128 个 θ'_i 进行处理，得到 128 位 Hash 值；

4) 使用 Hash 函数对 128 个 r'_i 进行处理，得到 128 位 Hash 值；

5) 将步骤 3 和 4 的两个 Hash 值组合成一个 256 位的随机值.

在以上过程中最关键的是步骤 3 和 4 中使用的 Hash 函数. 本文采用文献[2]中的 Hash 函数对鼠标轨迹进行处理. 该 Hash 函数基于 CML(耦合映像格子)，采用双向耦合方式：

$$x_n(j) = (1-\varepsilon) f_\mu(x_{n-1}(j)) + \frac{\varepsilon}{2}(f_\mu(x_{n-1}(j-1)) + f_\mu(x_{n-1}(j+1))), \tag{9}$$

其中 $x_n(j)$ 表示第 j 个格子迭代 n 次后的状态，f_μ 为 logistic 映射，μ 为映射参数，$\varepsilon = 0.99$ 为耦合因子，根据文献[2] 的建议在实验中格子数设为 37. 该 Hash 函数的步骤如下：

1) 将输入的 128 个小数排列成 4 行 32 列的矩阵 M.

2) 对 CML 进行 4 次迭代，第 i ($i \leqslant 4$) 次迭代时第 j 个格子的 logistic 参数 μ_j 下式确定：

$$\mu_j = \begin{cases} 3.75 + (M_{i,j} + x_{n-1}(j))/8, & j \leqslant 32, \\ 4, & j > 32. \end{cases} \tag{10}$$

3) 保持各个格子 logistic 参数不变，再进行 80 次 CML 迭代.

4) 分别从 $x_{84}(11)$，$x_{84}(31)$ 和 $x_{84}(37)$ 取出 40，40 和 48 比特，组成 128 位的 Hash 值.

注意在步骤 2 和 3 中，为了加快扰动的扩散速度，每次迭代后还要对第 33 个格子进行一次 S 盒替换. 关于该算法更详细的内容请参见文献[2]. 同样地，我们将该方法产生的随机序列输入到 NIST 测试软件进行测试，第四节给出了测试结果.

由于文献[2]提出的 Hash 函数每次只能产生 128 位随机数，为了生成 256 位随机数，必须两次调用 Hash 函数，这大大增加了该 TRNG 的运算时间. 为了提高运算速度，我们提出了一个改进的方案. 该 TRNG 的输入只包含 128 个 θ'_i，但在 84 次迭代后，我们从 $x_{84}(7)$，$x_{84}(13)$，$x_{84}(19)$，$x_{84}(25)$，$x_{84}(31)$和 $x_{84}(37)$中分别抽取 40，40，48，40，40，48 比特，组成 256 比特的随机数. 改进的 TRNG 的运算时间可降为原来的一半，但其随机性能是否下降还需经过检验. 第 4 节将对上面提到的 3 个 TRNG 进行检测和对比.

4. 实验结果

在这一节中，我们对上面提到的 3 个 TRNG 进行检测和对比. 为了获得足够的原始数据，我们请 3 位用户独立地用鼠标进行 4096 次移动，每次移动的取样点不少于 150 个，然后分别使用 3 个 TRNG 对这些数据进行处理，共产生 9 个 1Mbit 的随机序列.

为了对序列的随机性进行严格的测试，我们采用了美国 NIST 机构建议的 16 种随机数测试（详见文献[2]）. 根据文献[2]，每种测试的检测结果用 $p\_value(0 \leqslant p\_value \leqslant 1)$ 表示，若 $p\_value > 0.01$，则说明该序列通过了随机性检测. 同时 p_value 的值距离 1 越近，则该序列越接近理想的随机序列. 表 1 显示了用户 A 使用三个 TRNG 产生的随机数的检测结果，其中加 * 号的测试含有多个测试值，在表格中以这些数值的平均值代替. 表 2 显示了三个 TRNG 的处理速度（所有程序均采用未优化的 Matlab 代码编写，CPU 为 1.5G 的 Intel Celeron）.

表 1　三个 TRNG 生成数据的随机性检测结果（用户 A）

测试名称	MASK 算法	Hash 函数	改进方案
FT	0.7710	0.9517	0.3607
FBT	0.7233	0.3840	0.3843
RT	0.5451	0.3607	0.6122
LROBT	0.9877	0.4061	0.0421
RBMRT	0.7988	0.9184	0.5601
DFTT	0.6238	0.8671	0.9879
ATMT*	0.5968	0.5672	0.4890
PTMT	0.6367	0.9855	0.0695
MUST	0.6845	0.0853	0.6321
LZCT	1.0000	1.0000	1.0000
LCT	0.5853	0.1326	0.7487
ST*	0.9678	0.6637	0.2062
AET	0.6342	0.7364	0.2024
CST*	0.9947	0.7837	0.5052
RET*	0.7059	0.6260	0.5027
REVT*	0.5523	0.2697	0.2502

从实验结果来看，三个 TRNG 均通过了 NIST 建议的全部测试，16 种测试结果的平均值分别为 0.7380，0.6086 和 0.4721. 单从随机性测试结果来看，基于 MASK 算法的 TRNG 要优于基于 Hash 函数的两个 TRNG. 然而，从表 2 列出的运算时间来看，基于 MASK 算法的 TRNG 在速度上要比另两种方案慢很多，而改进方案的运算速度最快，分别是前两个方案的 3.74 倍和 2.02 倍. 此外改进方案对于不同用户产生的随机数的检测结果表明（参见表 3），对于不同用户生成的鼠标轨迹，改进的方案均能通过所有的随机数测试. 综合考虑各方面的实验结果，我们认为改进方案已能满足 TRNG 的各项要求，可以应用于实际的 PC 平台上.

表 2　三个 TRNG 生成 256bit 随机数的平均时间

方案名称	MASK	Hash 函数	改进的方案
时间/ms	535	289	143

表 3　改进方案对不同用户的检测结果

测试名称	用户 A	用户 B	用户 C
FT	0.3607	0.5384	0.3043
FBT	0.3843	0.1929	0.2481
RT	0.6122	0.5687	0.5547
LROBT	0.0421	0.5758	0.1052
RBMRT	0.5601	0.9941	0.2797
DFTT	0.9879	0.2530	0.5797
ATMT*	0.4890	0.4785	0.5012
PTMT	0.0695	0.0831	0.0813
MUST	0.6321	0.2095	0.6627
LZCT	1.0000	1.0000	1.0000
LCT	0.7487	0.2403	0.9395
ST*	0.2062	0.9410	0.1591
AET	0.2024	0.7727	0.5567
CST*	0.5052	0.4127	0.2737
RET*	0.5027	0.7396	0.3692
REVT*	0.2502	0.6426	0.2861

5. 结　论

本文提出了一种基于鼠标轨迹的 TRNG 设计方法，使用该方法设计的 TRNG 具有通用性强、成本低和方便的优点. 由于同一用户产生的鼠标轨迹通常含有相似的模式，我们借助混沌系统对初始状态和系统参数极高的敏感性来去除这些模式. 我们分别使用了两种混沌密码技术，即图像加密算法和 Hash 函数对鼠标轨迹进行后处理，并采用 NIST 机构建议的 16 种随机数测试方法对处理后产生的大量数据进行了严格的测试. 测试的结果表明，本文提出的三

个 TRNG 均通过了所有随机数测试.综合检测结果和运算速度两个指标,我们认为采用改进方案的 TRNG 可应用于实际的 PC 平台,为 PC 用户提供安全、快速、方便和廉价的随机数.

[1] NIST Special Publication 800 – 22, http://csrc.nist.gov/rng/rng2.html, 2001

[2] Liu J D, Yu Y M 2007 *Acta Phys. Sin.* **56** 1297 (in Chinese) [刘建东、余有明 2007 物理学报 **56** 1297]

[3] Yi X 2005 *IEEE Trans. Circuits Syst.* Ⅱ **52** 354

[4] Zhang J S, Wang X, Zhang W F 2007 *Phys. Lett.* A **362** 439

[5] Wang X M, Zhang J S, Zhang W F 2003 *Acta Phys. Sin.* **52** 2737 (in Chinese) [王小敏、张家树、张文芳 2003 物理学报 **52** 2737]

[6] Peng F, Qiu S S, Long M 2005 *Acta Phys. Sin.* **54** 4562 (in Chinese) [彭 飞、丘水生、龙 敏 2005 物理学报 **54** 4562]

[7] Zhang Y W, Wang Y M, Shen X B 2007 *Science in China* (Series E) **37** 183 (in Chinese) [张翌维、王育民、沈绪榜 2007 中国科学(E 辑) **37** 183]

[8] Fridrich J 1998 *Int. J. Bifurc. Chaos* **8** 1259

[9] Zhou Q, Wong K W, Liao X F, Xiang T, Hu Y 2007 *Chaos, Solitons & Fractals* **38** 1081

[10] Jakimoski G, Kocarev L 2002 *IEEE Trans. Circuits Syst.* Ⅰ **48** 163

[11] Wang Y, Liao X F, Xiang T, Wong K W, Yang D G 2007 *Phys. Lett.* A **363** 277

[12] Li P, Li Z, Wolfgang A H, Chen G 2006 *Phys. Lett.* A **349** 467

[13] Tenny R, Tsimring L S 2005 *IEEE Trans. Circuits Syst.* Ⅰ **52** 672

[14] Chen G, Chen Y, Liao X F 2007 *Chaos, Solitons & Fractals* **31** 571

Ture random number generators based on mouse movement and chaos systems*

Zhou Qing Hu Yue Liao Xiao-Feng

(*College of Computer, Chongqing University, Chongging* 400030, *China*)

(Received 2 December 2007; revised manuscript received 27 January 2008)

Abstract

True random number generator (TRNG) is usually more secure than pseudo-random number generator. In this paper, a novel TRNG which produces random bits by noving the mouse casually is proposed which is cheap, convenient and universal for personal computer (PC) platforms. To eliminate the patterns among mouse movements of the same user, two approaches based on image encryption algorithm and Hash function were poposed, both making use of the sensitivity of chaos system. Abundant random bits generated by three users were tested strictly using U.S. NIST's 16 statistical tests. All TRNGs passed the test successfully, and the revised TRNG is believed to be applicable practically on PC platforms.

Keywords: chaos, TRNG, mouse movement

PACC: 0545, 0540

* Project supported by the National Natural Science Foundation of China (Grant Nos. 60573047, 60703035) and the Program for New Century Excellent Talents in University.