一种 Linux 用户空间下的快速伪随机数生成算法

高小鹏 张 哲 刘国良 龙 翔

(北京航空航天大学 计算机学院, 北京 100083)

摘 要: 随机数发生器是网络安全应用中的重要组成部分,对于构造加密算法的密钥具有重要作用. Linux 操作系统提供了内核级随机数发生器,但因其随机数产生效率较低而不宜将其应用于密钥变换频繁的网络安全应用. 给出了一个快速伪随机数生成算法. 算法以 64 bit 硬件高频计数器作为随机数源. 算法将计数器的低 32 bit 放入集合中,然后通过 SHA (Security Hash Algorithm)算法对集合进行处理,并采用集合的前 16 byte 作为随机数输出. 采用非参数检验方法检验算法产生的随机数质量,测试结果表明算法产生的随机数具有较高的安全性. 同时由于算法运行在用户空间,比 Linux 的内核级随机数发生器具有较高的随机数生成效率.

关 键 词: 随机数; 均匀性; 随机性; 统计独立; Linux

中图分类号: TP 393.08

文献标识码: A **文 章 编 号**: 1001-5965(2006)04-0489-05

Fast algorithm for pseudo random number generation in Linux user space

Gao Xiaopeng Zhang Zhe Liu Guoliang Long Xiang

(School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)

Abstract: RNGs(random number generators) are important building blocks for algorithms in security applications. They are paramount in construction of encryption keys. For security applications with key exchange in high frequency, the two RNGs provided by Linux kernel are not acceptable because of their low efficiency. An algorithms for fast pseudo random number generation as proposed is implemented in Linux user space. The source of random number is a high-frequency 64 bit counter. The lowest 4 bytes of the counter are added in a pool, then the pool is hashed with SHA(security hash algorithm). The first 16 bytes of the hash are output. This process is repeated until the requested number of random number is achieved. Several statistical tests are employed to investigate the randomness of RNGs. The results show that the quality of random number generated are guaranteed. Due to its running in Linux user space, this algorithm has much higher efficiency than Linux's two RNGs.

Key words: random number; uniformity; randomness; statistical independence; Linux

随机数发生器是网络安全应用中的重要组成部分,对于构造加密算法的密钥具有重要作用,其生成的随机数质量在相当程度上决定了加密算法(特别是对称加密算法)的安全性. 对于密钥变换频繁的网络安全应用(例如高性能 VPN 安全网关),不仅需要高质量的密钥,而且要求密钥更新必须保持较高频率. 对于此类应用,如果随机数生成效率较低,则会因应用程序被阻塞导致性能下降.

随着 Linux 的迅速普及,Linux 已大量应用于网络安全应用中. 但 Linux 内核提供的 2 个伪随机数生成算法在实际应用中存在着性能较低的缺陷. 本文提出了一种运行在 Linux 用户空间的快速伪随机数生成算法,并通过数理统计测试验证

收稿日期: 2005-03-07

作者简介: 高小鹏(1970-),男,山东济南人,讲师, gxp@buaa.edu.cn.

了伪随机数生成质量. 因算法运行在用户空间,故大幅度提高了伪随机数生成的性能. 同时该算法具有较好的可移植性,可以被广泛的应用于其他领域.

1 Linux 伪随机数发生器分析

1.1 内核级随机数发生器简介

Linux 建有内核级随机数发生器[1],并被抽象为2个字符设备/dev/random 和/dev/urandom. 前者为阻塞型随机数发生器,后者为非阻塞型随机数发生器.

1) 阻塞型随机数发生器

应用程序可通过 read( ) 系统调用读取/dev/random 内部熵池中的随机数数值. 当熵池中的随机数数目少于应用程序需读取的数目时,read( ) 系统调用被阻塞,应用程序随之被挂起.

2) 非阻塞型随机数发生器

/dev/urandom 是非阻塞伪随机数发生设备,可随时提供给应用程序任意数目的随机数. 当应用程序所需的随机数过多时,导致其内部的熵池无法得到及时更新,故其熵值将有所降低. 与/dev/random 相同,应用程序通过 read( ) 系统调用读取随机数数值.

1.2 内核级随机数发生器性能的定性分析

应用程序为了从/dev/random 或者/dev/urandom 获取随机数,必须调用 read( ) 系统调用,而 read( ) 系统调用将导致系统上下文发生切换. 可见,从应用程序角度出发,随机数生成的延迟不仅包括随机数算法自身的延迟,还包括上下文切换等系统开销. 对于密钥变换频繁的网络安全应用而言,如果随机数生成的时延较大,则会导致应用程序性能下降. 显然这对于此类应用是难以接受的.

2 伪随机数生成算法描述

本文提出的快速伪随机数生成算法(简称快速算法)采用高频硬件计数器的计数值作为随机数源,并通过 SHA(Security Hash Algorithm)[2] 对计数值集合进行处理.

2.1 随机数源

最常见的信息源包括用户的人为反应或某种经过排列变形后的高频时钟的序列[3]. 为了方便软件调试和性能测量等,Intel 公司在其 Pentium 处理器及后继处理器内建了时间戳计数器(TSC, Time Stamp Counter). TSC 为 64 bit 硬件计数器,每个处理器时钟周期其计数值加 1. 应用程序可以通过非特权级指令 RDTSC[4] 读取 TSC 计数值. 快速算法采用 TSC 作为随机数源. 由于应用程序可以直接在用户空间访问 TSC,这使得快速算法得以在用户空间实现.

2.2 SHA

SHA 被广泛应用于密码学算法、密钥散列验证和随机数的产生. SHA 因其单向散列函数的不可逆性使得不能根据已有随机数逆推算法的输入,从而不能推断 SHA 产生的后继随机数,故 SHA 生成的随机数具有不可预测性,保证了所生成的随机数是安全的.

2.3 算法流程

快速算法把从 TSC 中获取的计数值放入一个集合,并对该集合应用 SHA,算法流程见图1.

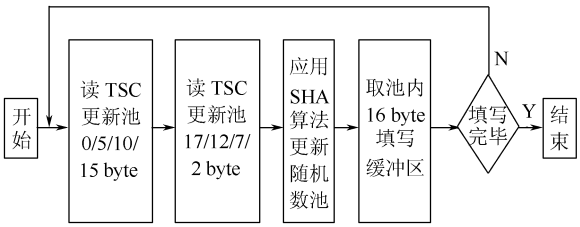

图1 伪随机数生成算法

由于 SHA 产生 160 bit(20 byte)的消息摘要,因此设置缓冲区(即前述集合)容量为 20 byte. 考虑到 TSC 的高 32 bit 计数值变化速率远低于低 32 bit 计数值,故每次更新集合前分 2 次读取 TSC 计数值,且只将低 32 bit 计数值写入缓冲区. 之后应用 SHA 更新缓冲区. 更新完成后,取散列结果的前 16 byte 作为可用的随机数返回给应用程序.

3 算法评价

本文提出快速算法的目的在于提高随机数产生效率. 但考虑到随机数安全性的重要性,故必须从安全性和效率两方面对算法进行评价. 本文采用非参数检验方法检验了随机数的安全性. 为评价快速算法的效率,不仅直接对比了 Linux 内核随机数生成器与快速算法的效率,而且以 IKE(Internet Key Exchange)[5] 为对象考察了快速算法对实际应用的影响. 测试用 PC 的处理器为 Pentium 4(主频 1.6 GHz),操作系统为 RedHat 7.2.

3.1 算法安全性定性分析

TSC 低 32 bit 数值变化剧烈(1GHz 处理器的 TSC 每 4.3 s 计数溢出并复位). 为预测如此剧变的随机数,攻击者必须首先解决网络环境下的时钟同步难题. 由于 SHA 优秀安全特性,攻击者对 1 bit计数值估计不准确将会使 SHA 输出的随机

数完全不同. 此外快速算法内部随机数池在每次随机数生成时完全更新,且仅部分作为随机数输出. 这些因素均使得对随机数的预测和攻击变得非常困难.

3.2　算法安全性定量分析

3.2.1　随机数性质及参数检验方法

一般而言,当密码学中的随机数满足如下要求则认为该随机数是安全的:

1) 均匀性. 将随机数样本分为 n 个等长区间,则每个区间内的随机数期望值为 N/n (N 为样本总数).

2) 独立性. 1 个随机数在某个区间出现的概率与其出现在其它区间内的概率无关.

利用非参数检验方法可检验所生成的随机数是否满足上述 2 个要求[6,7]. 非参数检验方法包括均匀性检验和独立性检验两大类,常用的有 χ^2 检验、柯尔莫戈洛夫-斯米尔诺夫检验、自相关检验和间隙检验等. 其中前两者检验随机数的均匀性,后两者检验随机数的独立性. 本文采用上述 4 种非参数检验方法对随机数的安全性进行定量分析.

3.2.2　检验结果

检验快速算法随机数安全性采用了大样本方法(样本总数 $N=2\ 000\ 000$). 限于篇幅本文仅对长度为 4 bit 的随机数作均匀性和独立性检验,表 1 为频数统计结果. 本节后续部分根据表 1 对长度为 4 bit 的随机数序列分别作 χ^2 检验、柯尔莫戈洛夫-斯米尔诺夫检验、自相关检验和间隙检验(均取检验水平 $\alpha=0.01$).

表 1　长度为 4 bit 随机数的频率数

数值	频数	百分比/%	累计频率
0	125 159	6.3	6.3
1	124 680	6.2	12.5
2	125 254	6.3	18.8
3	125 530	6.3	25.0
4	125 572	6.3	31.3
5	125 324	6.3	37.6
6	125 425	6.3	43.8
7	124 944	6.2	50.1
8	124 255	6.2	56.3
9	124 381	6.2	62.5
10	124 852	6.2	68.8
11	125 375	6.3	75.0
12	125 112	6.3	81.3
13	124 295	6.2	87.5
14	124 362	6.2	93.7
15	125 480	6.3	100.0
Σ	2 000 000	100.0	

1) χ^2 检验

根据表 1 构造 χ^2 检验,见表 2.

表 2　χ^2 检验表 (检验水平 $\alpha=0.01$)

序号	取值区间	实际频数 F_i	理论频数 E_i	F_i-E_i	$(F_i-E_i)^2$	$\frac{(F_i-E_i)^2}{E_i}$
1	[0,1]	125 159	125 000	159	25 281	0.202 248
2	[1,2]	124 680	125 000	−320	102 400	0.819 2
3	[2,3]	125 254	125 000	254	64 516	0.516 128
4	[3,4]	125 530	125 000	530	280 900	2.247 2
5	[4,5]	125 572	125 000	572	327 184	2.617 472
6	[5,6]	125 324	125 000	324	104 976	0.839 808
7	[6,7]	125 425	125 000	425	180 625	1.445
8	[7,8]	124 944	125 000	−56	3 136	0.025 088
9	[8,9]	124 255	125 000	−745	555 025	4.440 2
10	[9,10]	124 381	125 000	−619	383 161	3.065 288
11	[10,11]	124 852	125 000	−148	21 904	0.175 232
12	[11,12]	125 375	125 000	375	140 625	1.125
13	[12,13]	125 112	125 000	112	12 544	0.100 352
14	[13,14]	124 295	125 000	−705	497 025	3.976 2
15	[14,15]	124 362	125 000	−638	407 044	3.256 352
16	[15,16]	125 480	125 000	480	230 400	1.843 2

计算检验统计量:

$$\chi^2=\sum_{i=1}^{16}\frac{(F_i-E_i)^2}{E_i}=26.694$$

经查表[8]可知 χ^2 检验的临界值 $\chi^2_{0.99}(15)=30.578$. 显然 $\chi^2<\chi^2_{0.99}(15)$,故可认为所生成的随机数通过 χ^2 检验,即满足随机数的均匀性要求.

2) 柯尔莫戈洛夫-斯米尔诺夫检验

类似地构造表 3. 从表 3 可知最大偏差 $D_{\max}$ 出现在区间[6,7],数值为 0.000 972 00. 根据检验水平 $\alpha=0.01$ 可确定柯尔莫戈洛夫-斯米尔诺夫检验的临界值[8]:

$$D_{\lim}=\frac{1.63}{\sqrt{N}}=\frac{1.63}{\sqrt{2\ 000\ 000}}=0.001\ 152\ 6$$

$D_{\max}<D_{\lim}$,故可认为所产生的随机数通过柯尔莫戈洛夫-斯米尔诺夫检验,即满足随机数的均匀性要求.

3) 自相关检验

由于所选随机数样本巨大($N=2\ 000\ 000$),本文仅给出间隙 $m\in[1,16]$ 的计算结果(如表 4). 由表 4 知间隙长度 $m\in[1,16]$ 的随机数的检验统计量绝对值 ρ 均低于临界值 $Z\alpha/2$,即均满足随机数序列的独立性条件,故可认为所产生的随机数满足独立性要求.

4) 间隙检验

将快速算法所生成的随机数分成 12 个间隙区间,并给出各间隙区间的统计结果如表 5 所示.

表 3　柯尔莫戈洛夫-斯米尔诺夫检验表

(检验水平 $\alpha=0.01$)

序号	取值区间	F_i	实际频率	实际累积频率 $S_n(x)$	理论累积频率 $F(x)$	$\lvert F(x)-S_n(x)\rvert$
1	[0,1]	125 159	0.062 579 5	0.062 579 5	0.062 5	0.000 079 50
2	[1,2]	124 680	0.062 34	0.124 919 5	0.125	0.000 080 50
3	[2,3]	125 254	0.062 627	0.187 546 5	0.187 5	0.000 046 50
4	[3,4]	125 530	0.062 765	0.250 311 5	0.25	0.000 311 50
5	[4,5]	125 572	0.062 786	0.313 097 5	0.312 5	0.000 597 50
6	[5,6]	125 324	0.062 662	0.375 759 5	0.375	0.000 759 50
7	[6,7]	125 425	0.062 712 5	0.438 472	0.437 5	0.000 972 00
8	[7,8]	124 944	0.062 472	0.500 944	0.5	0.000 944 00
9	[8,9]	124 255	0.062 127 5	0.563 071 5	0.562 5	0.000 571 50
10	[9,10]	124 381	0.062 190 5	0.625 262	0.625	0.000 262 00
11	[10,11]	124 852	0.062 426	0.687 688	0.687 5	0.000 188 00
12	[11,12]	125 375	0.062 687 5	0.750 375 5	0.75	0.000 375 50
13	[12,13]	125 112	0.062 556	0.812 931 5	0.812 5	0.000 431 50
14	[13,14]	124 295	0.062 147 5	0.875 079	0.875	0.000 079 00
15	[14,15]	124 362	0.062 181	0.937 26	0.937 5	0.000 240 00
16	[15,16]	125 480	0.062 74	1	1	0.000 000 00

表 4 自相关检验表(检验水平 α=0.01)

随机数间隙	相关系数均值 r	检验统计量 ρ
1	−0.001 390	−1.966 431
2	0.000 567	0.801 560
3	−0.000 437	−0.617 611
4	0.000 514	0.726 514
5	−0.000 458	−0.647 177
6	−0.000 542	−0.767 044
7	0.000 567	0.801 157
8	0.000 199	0.282 079
9	0.000 864	1.222 205
10	0.000 173	0.243 996
11	−0.000 051	−0.071 700
12	0.000 896	1.266 605
13	−0.000 139	−0.197 151
14	−0.000 961	−1.359 745
15	−0.000 029	−0.041 601
16	0.000 566	0.800 478
临界值 $Z\alpha/2$	2.58	

从表 5 可知累积频率的最大偏差绝对值 D=0.000 089. 由柯尔莫戈洛夫-斯米尔诺夫检验确定 α=0.01 时的临界值:

$$D_{\lim}=\frac{1.63}{\sqrt{N}}=\frac{1.63}{\sqrt{2\ 000\ 000}}=0.001\ 152\ 6$$

由于 $D<D_{\lim}$,故不能拒绝随机数序列的独立性假设,即可认为所产生的随机数满足独立性要求.

3.2.3 小 节

以上对快速算法所产生长度为 4 bit 的随机数的 4 种统计检验表明,所产生的随机数符合均匀性和独立性要求,即快速算法生成的 4 bit 随机数具有较高的安全性. 采用同样方法可以验证快速算法生成的其他长度的随机数也满足均匀性和独立性的要求. 由此表明,快速算法生成的随机数具有较高的安全性.

3.3 算法效率分析

因为/dev/random 是阻塞型随机数发生器而/dev/urandom是非阻塞型随机数发生器,所以/dev/random 的随机数生成效率低于/dev/urandom 的随机数生成效率,故本文只对比/dev/urandom 与快速算法的随机数生成效率.

1) 随机数生成效率的改善

Linux 内核的/dev/urandom 和快速算法生成不同长度的随机数的时延如图 2 所示.

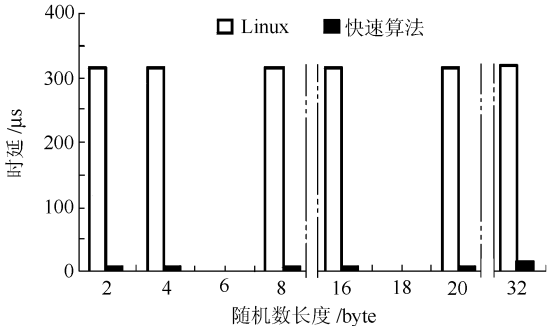

图 2 随机数生成时延对比

表 5 间隙检验表(检验水平 α=0.01)

序号	间隙长度	F_i	相对频率	$S_n(x)$	$F(x)$	$\lvert F(x)-S_n(x)\rvert$
1	[0,17]	1 374 239	0.687 125	0.687 125	0.687 044	0.000 081
2	[18,35]	429 836	0.214 920	0.902 045	0.902 059	0.000 014
3	[36,53]	134 784	0.067 393	0.969 438	0.969 349	0.000 089
4	[54,71]	42 018	0.021 009	0.990 447	0.990 407	0.000 040
5	[72,89]	13 086	0.006 543	0.996 990	0.996 998	0.000 008
6	[90,107]	4 186	0.002 093	0.999 083	0.999 060	0.000 023
7	[108,125]	1 237	0.000 619	0.999 702	0.999 706	0.000 004
8	[126,143]	420	0.000 210	0.999 912	0.999 908	0.000 004
9	[144,161]	132	0.000 066	0.999 978	0.999 971	0.000 007
10	[162,179]	33	0.000 017	0.999 995	0.999 991	0.000 004
11	[180,197]	9	0.000 003	0.999 998	0.999 997	0.000 001
12	[198,215]	4	0.000 002	1.000 000	0.999 999	0.000 001

从图 2 中可以直观看出快速算法的时延要远小于/dev/urandom 的时延,大约是后者是 2% 以下. 此外需要强调的是,如前文分析,当应用程序频繁调用/dev/urandom 产生随机数时,将导致随机数熵值的严重降低. 而快速算法由于每次被调用都完全更新内部缓冲区,因此所产生的随机数熵值不会降低.

2) IKE 整体交换效率的提高

IKE 协议的密钥交换分为 2 个阶段. 本文通过在 IKE 交换各条消息处理例程的入口和出口处测定所需时间,分别测试了应用 Linux 内核的/dev/urandom与快速算法后密钥交换各个阶段时延,测试数据见表 6.

表 6　算法改进前后 IKE 交换时间对比(PSK 验证)

IKE 交换	消息序号	Linux/μs	快速算法/μs	改进比例/%
阶段 1	1	1 666.82	418.29	75
	2	260.57	262.38	无差别
	3	26 411.48	18 115.48	31.40
	4	44 274.22	37 410.8	15.50
	5	18 273.44	18 290.43	无差别
	6	121.44	121.07	无差别
	累积	91 007.97	74 618.45	18
阶段 2	1	4 635.02	808.02	82.30
	2	3 567.95	454.48	87.30
	3	251.98	251.14	无差别
	累积	8 454.95	1 513.64	82.10
IKE 整体效率	累积	99 462.92	76 132.09	23.50

由表 6 中可见应用快速算法后,整个 IKE 交换阶段 1 的效率提高约 18%,而阶段 2 的效率提高约 82%,整体效率提高约 23%. 部分消息的处理在应用快速算法后时延无明显变化的原因有:

1) 未涉及随机数的使用. 包括阶段 1 的第 5 条和第 6 条消息,以及阶段 2 的第 3 条消息.

2) 未使用/dev/urandom 方法生成随机数. 仅限于阶段 1 的第 2 条消息.

4　结 束 语

本文提出了一种完全运行在 Linux 用户空间的快速伪随机数生成算法. 该算法不仅具有较高的安全特性,而且相对于 Linux 内核提供的伪随机数生成算法大幅度提高了随机数生成性能. 本文算法完全运行在用户空间,且仅需一个高频硬件计数器,故本文算法具有很好的可移植性,可被应用于其他系统.

参考文献 (**References**)

[1] Linux 2.4 内核随机数生成算法源代码[CP/OL]. /usr/src/linux/drivers/char/random.c
Source code of Linux 2.4 random number generator[CP/OL]. /usr/src/linux/drivers/char/random.c(in Chinese)

[2] FIPS-180-1. Secure hash standard[S]. National Institute of Standards and Technology, US Department of Commerce, 1995

[3] Mungkee D. Cryptographic random number generators[EB/OL]. http://www.phrack.org/phrack/59/p59-0x0f.txt

[4] Intel. Intel architecture software developer's manual volume 2: instruction set reference[EB/OL]. http://www.intel.com/design/pentiumii/manuals/243191.htm

[5] Kaufman C. Internet key exchange (IKE v2) protocol[EB/OL]. IETF draft-ietf-ipsec-ikev2-17.txt. Sep,2004

[6] Donald E K. The art of computer programming (2nd Edition), Volume 2/seminumerical algorithms[M]. Beijing: Tsinghua University Press,1980

[7] Juan S. Statistical testing of random number generators[C/OL]. Proceedings of the 22nd National Information Systems Security Conference. http://csrc.nist.gov/nissc/1999/proceeding/papers/p24.pdf

[8] 韩于羹. 应用数理统计[M]. 北京:北京航空航天大学出版社,1989
Han Yugeng. Mathematical statistics & application[M]. Beijing: Beijing University of Aeronautics & Astronautics Press,1989 (in Chinese)