Blog Entry © Saturday July 11, 2026, by James Pate Williams, Jr. Testing a LCG PRNG and a CSPRNG

Reference: ***Handbook of Applied Cryptography*** © 1996-1997 by Alfred J. Menezes, P. van Oorschot and S. Vanstone Chapter 5 Section 5.4.4 Pages 181-184

Reproducing **5.31 Example** in the Reference on pages 182-183:

```
== Menu ==
1 HoAC 5.31 Example
2 Conversion Tests
3 Create a Provable Prime
4 Generate Pseudorandom Bits C PRNG
5 Generate BBS Parameters
6 Generate and Test CSPRNG Bits
7 Exit
1
monobit test
n0 = 84
n1 = 76
serial bit test
n00 = 44
n01 = 40
n10 = 40
n11 = 35
poker test
111     ni[7] = 1
000     ni[0] = 1
110     ni[6] = 1
001     ni[1] = 1
000     ni[0] = 2
101     ni[5] = 1
001     ni[1] = 2
110     ni[6] = 2
111     ni[7] = 2
100     ni[4] = 1
100     ni[4] = 2
100     ni[4] = 3
100     ni[4] = 4
111     ni[7] = 3
100     ni[4] = 5
011     ni[3] = 1
000     ni[0] = 3
100     ni[4] = 6
010     ni[2] = 1
100     ni[4] = 7
111     ni[7] = 4
```

```
011     ni[3] = 2
110     ni[6] = 3
010     ni[2] = 2
010     ni[2] = 3
010     ni[2] = 4
011     ni[3] = 3
110     ni[6] = 4
001     ni[1] = 3
100     ni[4] = 8
010     ni[2] = 5
001     ni[1] = 4
010     ni[2] = 6
011     ni[3] = 4
101     ni[5] = 2
111     ni[7] = 5
001     ni[1] = 5
001     ni[1] = 6
001     ni[1] = 7
001     ni[1] = 8
111     ni[7] = 6
000     ni[0] = 4
110     ni[6] = 5
001     ni[1] = 9
000     ni[0] = 5
101     ni[5] = 3
001     ni[1] = 10
110     ni[6] = 6
111     ni[7] = 7
100     ni[4] = 9
100     ni[4] = 10
100     ni[4] = 11
100     ni[4] = 12
s = 53  k = 53
ni[0] = 5
ni[1] = 10
ni[2] = 6
ni[3] = 4
ni[4] = 12
ni[5] = 3
ni[6] = 6
ni[7] = 7
runs test
e[0] = 5.000000
e[1] = 10.062500
e[2] = 20.250000
B[0] = 25
B[1] = 4
```

```
B[2] = 5
G[0] = 8
G[1] = 20
G[2] = 12
autocorrelation test
A(8) = 100
X1 = 0.400000
X2 = 0.625157
X3 = 9.641509
X4 = 31.791306
X5 = 3.987563
```

Next, we test the standard pseudorandom number generator in the Microsoft C computer language partial output:

```
runs test
e[0] = 9.762695
e[1] = 19.526367
e[2] = 39.054688
e[3] = 78.113281
e[4] = 156.234375
e[5] = 312.484375
e[6] = 625.000000
e[7] = 1250.062500
e[8] = 2500.250000
B[0] = 2505
B[1] = 1273
B[2] = 581
B[3] = 323
B[4] = 145
B[5] = 93
B[6] = 37
B[7] = 22
B[8] = 7
G[0] = 2440
G[1] = 1291
G[2] = 642
G[3] = 299
G[4] = 169
G[5] = 83
G[6] = 39
G[7] = 19
G[8] = 5
autocorrelation test
A(8) = 10037
X1 Monobit Test = 0.369800
X2 Serial Bit Test = 0.379187
X3 Poker Test = 14.195200
```

```
X4 Runs Test = 16.252851
X5 Autocorrelation Test = 0.587031
C LCPRNG Runtime = 1.106000
== Menu ==
1 HoAC 5.31 Example
2 Conversion Tests
3 Create a Provable Prime
4 Generate Pseudorandom Bits C PRNG
5 Generate BBS Parameters
6 Generate and Test CSPRNG Bits
7 Exit
```

As a last statistical test, we test the Blum-Blum-Shub CSPRNG:

```
s = 5000        k = 5000
ni[0] = 328
ni[1] = 303
ni[2] = 315
ni[3] = 351
ni[4] = 299
ni[5] = 333
ni[6] = 280
ni[7] = 306
ni[8] = 316
ni[9] = 319
ni[10] = 293
ni[11] = 315
ni[12] = 305
ni[13] = 317
ni[14] = 311
ni[15] = 309
runs test
e[0] = 9.762695
e[1] = 19.526367
e[2] = 39.054688
e[3] = 78.113281
e[4] = 156.234375
e[5] = 312.484375
e[6] = 625.000000
e[7] = 1250.062500
e[8] = 2500.250000
B[0] = 2539
B[1] = 1212
B[2] = 632
B[3] = 334
B[4] = 141
B[5] = 68
B[6] = 49
```

```
B[7] = 21
B[8] = 11
G[0] = 2530
G[1] = 1236
G[2] = 604
G[3] = 314
G[4] = 168
G[5] = 79
G[6] = 42
G[7] = 22
G[8] = 11
autocorrelation test
A(8) = 9982
X1 = 0.096800
X2 = 0.225766
X3 = 12.966400
X4 = 11.728346
X5 = -0.190962
```

We can also create provable prime numbers using Maurer's Algorithm:

```
== Menu ==
1 HoAC 5.31 Example
2 Conversion Tests
3 Create a Provable Prime
4 Generate Pseudorandom Bits C PRNG
5 Generate BBS Parameters
6 Generate and Test CSPRNG Bits
7 Exit
3
number of bits = 2048
provable prime:
21210876565365622638368995661873242700210436860632119259155385800
04403\
24592983397501006904794296075219784919603492030790016894466305980
37693387\
52472527173156790546170845423599278138950368075898052538144253250
56074129\
23502565318671350751532591822431905338851810805870829033128500520
28449096\
07815013637388481736261837479852724524602568168189216267656843830
07300058\
86270908535618366063361384841516407281200163106005742439002109290
69689267\
29663881241352168689286834964214563796141178621643184673697220520
07533508\
26384575312748794927813446351027943339339026831558560743470423980
74496103\
```

```
5687323342011339811567482901622551999061412 7
Miller-Rabin (20) = 1
(long)(floor(log2(N) + 1)) = 2048
seed1 = 1783814649
seed2 = 1783814649
runtime in seconds = 6.731000
== Menu ==
1 HoAC 5.31 Example
2 Conversion Tests
3 Create a Provable Prime
4 Generate Pseudorandom Bits C PRNG
5 Generate BBS Parameters
6 Generate and Test CSPRNG Bits
7 Exit
```

We use two different PRNGs: the regular C pseudorandom number and the large integer PRNG in the large integer package.

: