Pseudo-random number generators

A **(pseudo-) random number generator (pRNG)** is an algorithm which produces a long stream of seemingly random numbers in a prescribed range, from a specified initial state, the **seed**.

But what does **(pseudo-) random** mean?

Note that a deterministic process cannot produce truly random outputs, no matter how haphazard or confusing the outputs may appear.

There are two types of conditions we might demand of a stream of numbers, to suit various purposes: *statistical* and *cryptographic*.

Statistical conditions on a stream of numbers are requirements that the stream resemble samples from a sequence of idential random variables.

Such pRNGs are useful in simulations and modeling.

But periodicity or other accidental internal patterns in pRNGs can produce dangerously erroneous results in simulations if those patterns happen to resonate with the model being studies.

Cryptographic conditions are *strictly stronger* than statistical conditions, and demand **unpredictability**.

A stream of numbers can meet a great variety of statistical tests and still be predictable.

One cryptographic application is to *asynchronous stream ciphers* where a small key produces a *long random-looking* **keystream**

$$S = (s_0, s_1, s_2, s_3, \ldots)$$

of integers modulo m by a chosen pRNG to use as key for a *fake* one-time pad.

This is a *fake* one-time pad because the keystream is not *truly random*.

If the keystream is *periodic*, then the cipher degenerates into a *Vigenere*, which is vulnerable. Therefore, the very first imperative for pRNGs is to assure that the **period** (number of steps before it repeats) is as long as possible.

Periodicity is an extreme example of *predictability*, a bad cryptographic feature for a pRNG to have.

More precisely, a sequence

$$s_0, s_1, s_2, \ldots$$

is said to be **periodic** with **period** p if

$$s_{i+p} = s_i \qquad \text{(for all indices } i\text{)}$$

Certainly not every sequence is periodic, but sequences produced by simple mechanisms by 'finite-state' machines tend to be. The first issue is to make the period as large as possible for a given size of mechanism.

Internal versus external states

In the most naive form of pRNGs the next output is computed from the previous output (or from a finite set of previous outputs) using a public algorithm but secret constants and secret seed value.

A subtler version uses a little more memory to keep track of a secret **internal state**, which is computed from the previous internal state using a public algorithm and secret constants and secret seed. But the actual *outputs* are computed from the internal state in a supposedly irreversible manner.

That is, in the first, more naive version, the **state** is **external**.

In the second, more sophisticated version, the **state** is **internal**.

Linear congruential generators (LCGs)

Linear congruential generators (LCGs) are *not* good by themselves for use in secret ciphers but *are* useful for other things, such as simulation.

Fix a modulus m, an a invertible mod m, and integer b. Take a **seed** s_0 in $\mathbf{Z}/m$. The stream of pseudorandom numbers produced from this data is

$$\begin{aligned} s_1 &= (a \cdot s_0 + b) \,\%\, m \\ s_2 &= (a \cdot s_1 + b) \,\%\, m \\ s_3 &= (a \cdot s_2 + b) \,\%\, m \\ &\dots \\ s_{n+1} &= (a \cdot s_n + b) \,\%\, m \end{aligned}$$

Thus, the state is external. The secrets are the constants a, b, m and the seed s_o.

For example, $m = 23$, $a = 2$, $b = 5$, so the next element of a stream is obtained from the previous by

$$s_{n+1} = (2 \cdot s_n + 5) \,\%\, 23$$

With seed $s_0 = 1$, the stream produced (including the seed value 1) is

$$1, 7, 19, 20, 22, 3, 11, 4, 13, 8, 21, 1, 7, 19, 20, \ldots$$

If an earlier value recurs, then the sequence starts repeating, since each value is completely determined by the previous. The **period** is 11 and (necessarily) *not* every element of $\mathbf{Z}/23$ occurs, but only 11 of them.

If we choose a seed *not* among the states appearing starting with seed 1, for example $s_o = 2$, then we get another 11 values mod 23:

$$2, 9, 0, 5, 15, 12, 6, 17, 16, 14, 10, 2, 9, 0 \ldots$$

Thus, with $m = 23$, $a = 2$, $b = 5$, the only missing value is 18 (mod 23). If we use seed $s_o = 18$ we get a very unrandom

$$18, 18, 18, 18, 18, 18, 18, 18, \ldots$$

That is, 18 is a **fixed point** of this pRNG, meaning that (with these constants m, a, b) if the (external) state is ever 18 then it does not change.

For *any* m, b if the constant a is just 1, then the pRNG is badly degenerate, especially from the viewpoint of predictability:

$$s_{n+1} = (s_n + b) \,\%\, m$$

For example, with $m = 23$, $a = 1$, $b = 5$, and seed $s_o = 1$

$$1, 6, 11, 16, 21, 3, 8, 13, 18, 0, 5, 10, 15, \ldots$$

The wrap-around effect is not sufficient to fool anyone.

Another example: $m = 23$, $a = 3$, $b = 5$, so

$$s_{n+1} = (3 \cdot s_n + 5) \,\%\, 23$$

With seed $s_0 = 1$, the stream is

$$1, 8, 6, 0, 5, 20, 19, 16, 7, 3, 14, 1, 8, \ldots$$

The **period** is 11 and *not* every element of $\mathbf{Z}/23$ occurs. With the same constants m, a, b, try seeds $2, 3, 4, 5, 6, 7, 8, 9, 10$:

$2, 11, 15, 4, 17, 10, 12, 18, 13, 21, 22, 2, 11, \ldots$
$3, 14, 1, 8, 6, 0, 5, 20, 19, 16, 7, 3, 14, \ldots$
$4, 17, 10, 12, 18, 13, 21, 22, 2, 11, 15, 4, 17, \ldots$
$5, 20, 19, 16, 7, 3, 14, 1, 8, 6, 0, 5, 20, \ldots$
$6, 0, 5, 20, 19, 16, 7, 3, 14, 1, 8, 6, 0, \ldots$
$7, 3, 14, 1, 8, 6, 0, 5, 20, 19, 16, 7, 3, \ldots$
$8, 6, 0, 5, 20, 19, 16, 7, 3, 14, 1, 8, 6, \ldots$
$9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, \ldots$
$10, 12, 18, 13, 21, 22, 2, 11, 15, 4, 17, 10, 12, \ldots$

We see that 9 is the **fixed point** for the LCG.

Every LCG

$$s_{n+1} = (a \cdot s_n + b) \,\%\, p$$

with *prime* modulus p and $2 \leq a < p$ has **exactly one fixed point**, found by solving

$$\begin{gathered} a \cdot x + b = x \bmod p \\ (a-1) \cdot x = -b \bmod p \\ x = -b \cdot (a-1)^{-1} \bmod p \end{gathered}$$

where the inverse is modulo p.

In the example above,

$$s_{n+1} = (3 \cdot s_n + 5) \,\%\, 23$$

has fixed point x where

$$\begin{gathered} 3 \cdot x + 5 = x \bmod 23 \\ (3-1) \cdot x = -5 \bmod 23 \\ x = -5 \cdot (3-1)^{-1} \bmod 23 \\ x = -5 \cdot 12 = -60 = \boxed{9} \bmod 23 \end{gathered}$$

As an example mod 23 with longer period than 11, take

$$s_{n+1} = (5 \cdot s_n + 5) \,\%\, 23$$

With seed $s_0 = 1$, the stream is
1, 10, 9, 4, 2, 15, 11, 14, 6, 12, 19, 8, 22, 0, 5, 7, 17, 21, 18, 3, 20, 13, 1, 10, ... The period is 22. Every value mod 23 except the fixed point 16 appears before it repeats.

Theorem: For prime modulus p, if a is a *primitive root* modulo p, then the period of the LCG

$$s_{n+1} = (a \cdot s_n + b) \,\%\, p$$

is the maximum possible, namely $p - 1$, except when the seed is the single fixed point. Otherwise the period is a proper divisor of $p - 1$. Specifically, except for the fixed point, the period is the **order** of a modulo p, the smallest positive integer ℓ such that $a^\ell = 1 \bmod p$.

Remarks:

Whichever of the a, b, m, s_0 are 'secret', it turns out that it is easy to recover all of them from a small number of values $s_1, s_2, \ldots$ by linear algebra.

Having a large keyspace is *necessary*, but definitely not *sufficient*, for security. An example is Vigenere.

Especially in the context of trying to use a pRNG to make a key for a OTP, we have an obvious important question: how long before the sequence *repeats*? *How long is the period?* If the period is shorter than the message, the cipher is a Vigenere, which is vulnerable to Friedman's index-of-coincidence attack.

... no matter how apparently random the actual values are before they repeat!

Proof: (of theorem on LCGs) Consider the LCG

$$s_{i+1} = (a \cdot s_i + b) \,\%\, p$$

with $2 \leq a < p$. In terms of two-by-two matrices

$$\begin{pmatrix} s_{n+1} \\ 1 \end{pmatrix} = \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} s_n \\ 1 \end{pmatrix}$$

Iterating

$$\begin{pmatrix} s_{n+k} \\ 1 \end{pmatrix} = \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}^k \begin{pmatrix} s_n \\ 1 \end{pmatrix}$$

To understand the matrix

$$L = \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}$$

we use its **eigenvalues** and **eigenvectors**, numbers λ (eigenvalues) and 2-by-1 matrices (eigenvectors) v such that

$$Lv = \lambda \cdot v$$

That is, we look for vectors on which the action is as simple as possible, as though it were scalar multiplication.

For example, the effect of *iteration* of L is clear:

$$L^n v = \underbrace{L \circ \ldots \circ L}_{n} v = \lambda^n \cdot v$$

A corresponding benefit of expressing a *general* vector as a sum of *eigenvectors* is seeing the effect of *iteration*

$$u = v + w$$

where

$$Lv = \lambda \cdot v \quad Lw = \mu \cdot w$$

then

$$L^n u = \lambda^n \cdot v + \mu^n \cdot w$$

The **Cayley–Hamilton theorem** tells how to find eigenvalues of an n-by-n matrix M. Let I be the n-by-n identity matrix (that is, with 1's down the upper-left to lower-right diagonal, 0's off this diagonal). Let x be an indeterminate, and compute the determinant

$$P_M(x) = \det(xI - M)$$

This polynomial is the **characteristic polynomial** of M. The Cayley-Hamilton theorem says that the eigenvalues are the roots of the **characteristic equation**

$$P_M(x) = 0$$

Recall the easy formula for the determinant of a two-by-two matrix:

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = AD - BC$$

Here let I be the 2-by-2 identity matrix and compute

$$\begin{aligned} P_L(x) &= \det(\begin{pmatrix} x & 0 \\ 0 & x \end{pmatrix} - \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}) \\ &= \det \begin{pmatrix} x-a & -b \\ 0 & x-1 \end{pmatrix} \\ &= (x-a)(x-1) \end{aligned}$$

This does not depend on b. The characteristic equation is

$$(x-a)(x-1) = 0$$

which has roots $a, 1$. Some fooling around gives eigenvectors

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \text{eigenvector for eigenvalue } a$$

$$\begin{pmatrix} -(a-1)^{-1}b \\ 1 \end{pmatrix} = \text{e.vector for e.value } 1$$

That is, abbreviating $v = -(a-1)^{-1}b$,

$$\begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = a \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v \\ 1 \end{pmatrix} = 1 \cdot \begin{pmatrix} v \\ 1 \end{pmatrix}$$

We want to express $\begin{pmatrix} s_o \\ 1 \end{pmatrix}$ as a sum of eigenvectors. Since

$$\begin{pmatrix} s_o \\ 1 \end{pmatrix} - \begin{pmatrix} v \\ 1 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

whatever the value $*$ may be, the left-hand side is an a-eigenvector for $\begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}$. Thus,

$$\begin{pmatrix} s_o \\ 1 \end{pmatrix} = \left(\begin{pmatrix} s_o \\ 1 \end{pmatrix} - \begin{pmatrix} v \\ 1 \end{pmatrix} \right) + \begin{pmatrix} v \\ 1 \end{pmatrix}$$

expresses $\begin{pmatrix} s_o \\ 1 \end{pmatrix}$ as a sum of a-eigenvector and an 1-eigenvector.

That is, for any n

$$\begin{pmatrix} s_n \\ 1 \end{pmatrix} = \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} s_o \\ 1 \end{pmatrix}$$

$$= a^n \cdot \left(\begin{pmatrix} s_o \\ 1 \end{pmatrix} - \begin{pmatrix} v \\ 1 \end{pmatrix} \right) + 1^n \begin{pmatrix} v \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} a^n(s_o - v) + v \\ 1 \end{pmatrix}$$

That is we have a **formula** for the n^{th} output, given s_o and a, b, p with $v = -(a-1)^{-1}b \mod p$

$$\boxed{s_n = a^n(s_o - v) + v \mod p} \text{ or}$$

$$\boxed{s_n = a^n \cdot s_o + (a^n - 1)(a-1)^{-1}b \mod p}$$

Thus, the periodicity of a LCG is the same as the periodicity of powers of a, except for fixed point $s_o = v$. ///

Remark: Fermat's Little Theorem and/or fast modular exponentiation further simplify the computation.

For example, for the LCG

$$s_{n+1} = (3 \cdot s_n + 1) \,\%\, 23$$

suppose the seed is $s_o = 1$ and we want s_{1000}. The formula says that for p prime and $2 \leq a < p$ the LCG

$$s_{n+1} = (a \cdot s_n + b) \,\%\, p$$

with seed s_o has n^{th} state

$$s_n = a^n \cdot s_o + (a^n - 1)(a-1)^{-1} b \mod p$$

By brute force or Euclid, the inverse of $a - 1 = 3 - 1 = 2 \mod 23$ is 12. Thus

$$s_n = 3^n \cdot s_o + (3^n - 1) \cdot 12 \cdot 1 \mod 23$$

By Fermat, $3^{23-1} = 1 \mod 23$, and then $1000 = 45 \cdot 22 + 10$ gives

$$3^{1000} = 3^{45 \cdot 22 + 10} = (3^{22})^{45} \cdot 3^{10}$$

$$= 1^{45} \cdot 3^{10} = 3^{10} = 8 \mod 23$$

using fast modular exponentiation.

In summary, for the LCG

$$s_{n+1} = (3 \cdot s_n + 1) \,\%\, 23$$

with seed $s_o = 1$ we have

$$s_{1000} = 3^{1000} \cdot 1 + (3^{1000} - 1) \cdot 12 \cdot 1 \mod 23$$

$$= 8 \cdot 1 + (8 - 1) \cdot 12 \cdot 1 = 92 = 0 \mod 23$$

Remark: The linear algebra idea of using eigenvectors to break down the effect of a matrix into simpler pieces is very important throughout mathematics.

Example:

A linear congruential generator which meets many statistical, but not cryptographic criteria, from *Comm. of ACM*, vol. 31, 1988, pp. 1192-1201. Take

$$s_{n+1} = (16807 \cdot s_n) \,\%\, 2147483647$$

(Here 'b' is 0.) The modulus p is prime, and $16807 = 7^5$ is a primitive root. In fact, the modulus is the Mersenne prime

$$2147483647 = 2^{31} - 1$$

The smallest primitive root is 7, and the exponent 5 does not divide $p - 1$, so 7^5 is still a primitive root modulo p. The period is 2147483646.

And 7^5 is close to $\sqrt{p}$ which might suggest that multiplication by it will mix well.

Linear feedback shift registers

Linear feedback shift registers (LFSRs) are *not* adequate for secret ciphers but *are* useful for other things.

For size N, modulus m (often $m = 2$), **coefficients** $c = (c_0, \ldots, c_{N-1})$, and **seed** or **initial state** $s = (s_0, s_1, s_2, s_3, \ldots, s_{N-1})$ define for $n + 1 \geq N$

$$s_{n+1} = (c_0\, s_n + c_1\, s_{n-1} + \cdots + c_{N-1}\, s_{n-N+1}) \,\%\, m$$

For example, with lengths $N = 2$ and $N = 3$

$$s_{n+1} = (c_0\, s_n + c_1\, s_{n-1}) \,\%\, m$$

$$s_{n+1} = (c_0\, s_n + c_1\, s_{n-1} + c_2\, s_{n-2}) \,\%\, m$$

As with any pRNG, a basic question is to find the **period**, that is, *how soon does it repeat?*

The state $s_{n-N+1}, \ldots s_{n-2}, s_{n-1}, s_n$ is external. There is nothing hidden except the constants.

As with LCGs, these recursive definitions can be written in terms of matrices. For simplicity take $N = 4$. With coefficients $c = (c_0, c_1, c_2, c_3)$ the recursion relation is (mod m)

$$\begin{pmatrix} s_{n+1} \\ s_n \\ s_{n-1} \\ s_{n-2} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} s_n \\ s_{n-1} \\ s_{n-2} \\ s_{n-3} \end{pmatrix}$$

Letting C be that matrix, the effect of moving forward ℓ times in the pRNG's stream is

$$\begin{pmatrix} s_{n+\ell} \\ s_{n+\ell-1} \\ s_{n+\ell-2} \\ s_{n+\ell-3} \end{pmatrix} = C^{\ell} \cdot \begin{pmatrix} s_n \\ s_{n-1} \\ s_{n-2} \\ s_{n-3} \end{pmatrix}$$

and starting from the initial state

$$\begin{pmatrix} s_{n+3} \\ s_{n+2} \\ s_{n+1} \\ s_n \end{pmatrix} = C^{n} \cdot \begin{pmatrix} s_3 \\ s_2 \\ s_1 \\ s_o \end{pmatrix}$$

For example, suppose that the modulus is $m = 2$ and the coefficients are $(c_0, c_1, c_2, c_3) = (1, 0, 0, 1)$. Then the output stream is produced by

$$\begin{aligned} s_{i+1} &= c_0 \cdot s_i + c_1 \cdot s_{i-1} + c_2 \cdot s_{i-2} + c_3 \cdot s_{i-3} \\ &= 1 \cdot s_i + 0 \cdot s_{i-1} + 0 \cdot s_{i-2} + 1 \cdot s_{i-3} \\ &= s_i + s_{i-3} \end{aligned}$$

With seed $(s_0, s_1, s_2, s_3) = (1, 1, 0, 0)$, the whole stream produced (including the initial $(1, 1, 0, 0)$) is

$$1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, \ldots$$

With length 4, if an earlier-occuring pattern of 4 consecutive bits recurs, then the stream will repeat itself, since four consecutive values completely determine the next one. (The state is *external*.) Here the initial $(1, 1, 0, 0)$ recurred after 15 steps.

That is, the successive *states* of the LFSR $s_{i+1} = s_i + s_{i-3}$ are

1100
1001
0010
0100
1000
0001
0011
0111
1111
1110
1101
1010
0101
1011
0110
1100

This vertical display shows how the bits move from right to left in the state, with a new right-most bit being computed according to the coefficients.

With initial state 0000001 some *states* of the length-7 LFSR $s_{i+1} = s_i + s_{i-6}$, where the new *bit* is the sum of the first and last bits of the previous state, are

0000001
0000011
0000111
0001111
0011111
0111111
1111111
1111110
1111101
1111010
1110101
1101010
1010101
0101010
1010100
0101001
1010011
... (period is $127 = 2^7 - 1$)

Remark: *Period length* can depend both on the coefficients and the seed. For example, for the length-8 LFSR with coefficients

$$(c_0, c_1, \ldots, c_7) = 00011111$$

there are different periods depending upon input:

input	period
11111111	1
00001111	$3 \cdot 7$
00000011	$2 \cdot 3 \cdot 7$
00000001	$2 \cdot 2 \cdot 3 \cdot 7$
00000101	$3 \cdot 7$
00001001	$2 \cdot 7$
10000000	$2 \cdot 2 \cdot 3 \cdot 7$
10001000	$3 \cdot 7$
10010010	3
10010001	$2 \cdot 3 \cdot 7$
10011001	$2 \cdot 2$

An explicit computation for one of the inputs to that LFSR with coefficients 00011111, which computes the *new bit* by adding the *oldest* 5 bits of the current state (mod 2), and shifts the state to the left (forgetting the *oldest* bit). After 14 steps the initial state re-appears.

0^{th}	00001001
1^{st}	00010011
2^{nd}	00100111
3^{rd}	01001111
4^{th}	10011110
5^{th}	00111101
6^{th}	01111011
7^{th}	11110110
8^{th}	11101100
9^{th}	11011000
10^{th}	10110000
11^{th}	01100001
12^{th}	11000010
13^{th}	10000100
14^{th}	00001001

Theorem: For a length N binary (modulus 2) LFSR, the best possible (longest) period is $2^N - 1$. This best possible is obtained exactly when the **characteristic polynomial**

$$P(x) = c_{N-1} + c_{N-2}x + \ldots + c_0 x^{N-1} + x^N$$

is **primitive**, and then the initial state does not matter (as long as it's not the zero vector).

Definition: A polynomial P of degree n with coefficients in $\mathbf{Z}/p$ is **primitive** if $P(x)$ divides $x^n - 1$ but does not divide $x^t - 1$ for any $0 \leq t < n$. *(More discussion later.)*

The possibility of expressing the computation of the output stream in terms of linear algebra (matrices) is convenient both for computations and as an exploitable vulnerability for cryptanalytic attacks.

Blum-Blum-Shub

The BBS pRNG has a secret *internal state* from which a pseudo-random bit is computed.

In sharp contrast to LCGs and LFSRs, BBS is *provably secure* (assuming it's hard to factor): if an adversary can succeed in guessing the next bit more than half the time, they can factor a large $n = p \cdot q$.

Choose two large primes p, q equal to 3 mod 4, and compute $n = p \cdot q$. The modulus n is public and can be reused. For given **seed** s_0 compute the sequence of *internal states* $s_1, s_2, s_3, \ldots$ by

$$s_{i+1} = s_i^2 \,\%\, n$$

Then compute the i^{th} pseudo-random bit by

$$b_i = s_i \,\%\, 2$$

With luck, period $\sim \text{lcm}(p-1, q-1)/2$.

Tiniest example of BBS: Take $p = 3$, $q = 7$, so $n = 21$. Take $s_o = 2$. Then

$$s_1 = s_o^2 \,\%\, 21 = 4$$
$$s_2 = s_1^2 \,\%\, 21 = 16$$
$$s_3 = s_2^2 \,\%\, 21 = 4$$
$$s_4 = s_3^2 \,\%\, 21 = 16$$
$$s_5 = s_4^2 \,\%\, 21 = 4$$

Once the internal state returns to any previous value, it is in a loop. The sequence of bits produced is $b_i = s_i \,\%\, 2$, but in this example they're all just 0.

Failure due to too-small a situation. Try again...

Second-to tiniest example of BBS: Take $p = 7$, $q = 11$, so $n = 77$. Take $s_o = 2$. Then

$$\begin{aligned} s_1 &= s_o^2 \,\%\, 77 = 4 \\ s_2 &= s_1^2 \,\%\, 77 = 16 \\ s_3 &= s_2^2 \,\%\, 77 = 25 \\ s_4 &= s_3^2 \,\%\, 77 = 9 \\ s_5 &= s_4^2 \,\%\, 77 = 4 \\ s_5 &= s_4^2 \,\%\, 77 = 16 \end{aligned}$$

so the internal state repeats with a loop of length $4 \ll \mathrm{lcm}(7-1, 11-1)$. The sequence of bits produced is $b_i = s_i \,\%\, 2$, in this case

$$\begin{aligned} b_o &= s_o \,\%\, 2 = 0 \\ b_1 &= s_1 \,\%\, 2 = 4 \,\%\, 2 = 0 \\ b_2 &= s_2 \,\%\, 2 = 16 \,\%\, 2 = 0 \\ b_3 &= s_3 \,\%\, 2 = 25 \,\%\, 2 = 1 \\ b_4 &= s_4 \,\%\, 2 = 9 \,\%\, 2 = 1 \\ b_5 &= s_5 \,\%\, 2 = 4 \,\%\, 2 = 0 \end{aligned}$$

The loop of bits $0, 0, 1, 1, 0, 0, 1, 1, 0, 0, \ldots$ is *not* impressively random.

Third-to tiniest example of BBS: Take $p = 11$, $q = 19$, so $n = 209$. Take $s_o = 2$. The sequence of internal states (starting with s_1) generated by $s_{i+1} = s_i^2 \,\%\, n$ is

$$4, 16, 47, 119, 158, 93, 80, 130, 180, 5, 25, 207, 4$$

so the internal state repeats with a loop of length 12. (Don't count the 4 twice.) The loop of pseudo-random bits produced by $b_i = s_i \,\%\, 2$ (starting with b_1) is

$$0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1$$

Somewhat more random-looking.

Fourth-to tiniest example of BBS: Take $p = 19$, $q = 23$, so $n = 437$. Take $s_o = 2$. The sequence of internal states (starting with s_1) generated by $s_{i+1} = s_i^2 \,\%\, n$ is

$$4, 16, 256, 423, 196, 397, 289, 54, 294, 347, 234,$$
$$131, 118, 377, 104, 328, 82, 169, 156, 301, 142,$$
$$62, 348, 55, 403, 282, 427, 100, 386, 416$$

so the internal state repeats with a loop of length 30. The loop of pseudo-random bits produced by $b_i = s_i \,\%\, 2$ (starting with b_1) is

$$0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1,$$
$$0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0$$

Somewhat more random.

Remark: Security of BBS:

Since $p = 3 \bmod 4$, for z a square mod p

$$y = z^{(p+1)/4} \bmod p$$

computes the *principal square root* of z. Thus, the squaring map is a **permutation** of the set of squares mod p. The same is true for q, so (by Sun Ze's theorem) the squaring map is a permutation of the set of squares mod n.

A careful analysis (see Stinson's book, for example) can show that (in rough terms) if a sequence of bits produced by a BBS generator modulo n can be distinguished from a sequence of *random* bits, then there is a fast probabilistic algorithm to find square roots modulo n,

... and a square root oracle gives a fast probabilistic algorithm to factor n into its factors $n = p \cdot q$ as we saw earlier.

Critical obstacle:

Ironically, we need a very good random number generator to set up BBS in the first place!

The primes p and q and the seed s_o must themselves already be random of high quality, or an adversary will simply search the space of 'easy' choices for such primes and seed.

That is, without high quality random materials, the outputs are not high-quality random.

For high quality random materials, hardware devices are necessary.

For example, */dev/random* and */dev/urandom* on Linux.

Reminder about RSA key sizes, etc.:

Not only have CPUs generally gotten faster on a regular basis, roughly doubling every year plus-or-minus, but also:

In mid-1999 Adi Shamir, the 'S' in RSA, designed a specialized part-analogue part-digital computer 'Twinkle' that can run factorization attacks from 100 to 1000 times faster than previously, making 512-bit RSA moduli much less secure than before, since an adversary with sufficient resources can now factor your 512-bit RSA modulus in a few hours, rather than 10 years.)

Naor-Reingold pRNG

The Naor–Reingold pRNG is relatively new. It is provably secure *assuming* the infeasibility of factoring.

Fix a size n, and choose two random n-bit primes p and q, and let $N = pq$. Choose a random integer g which is a square mod N. Let

$$a = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$$

be random $2n$ values in $\{1, 2, \ldots, N\}$. Let $r = (r_1, \ldots, r_n)$ be random 0's and 1's. For an integer t of at most $2n$ bits, let $b_{2n}(t)$ be the binary-expansion 'digits' of t, padding on the left to have length $2n$.

For two binary vectors

$$v = (v_1, \ldots, v_n), \qquad w = (w_1, \ldots, w_n)$$

with v_i's and w_i's all 0's or 1's, define a dot product mod 2

$$v \cdot w = (v_1, \ldots, v_n) \cdot (w_1, \ldots, w_n)$$
$$= (\sum_{1 \leq i \leq n} v_i w_i) \ \% \, 2$$

For an n-tuple $x = (x_1, \ldots, x_n)$ of 0's and 1's define a $\{0, 1\}$-valued function

$$f(x) = f_{N,g,a,r}(x)$$
$$= r \cdot b(g^{a_{1,x_1} + a_{2,x_2} + a_{3,x_3} + \cdots + a_{n,x_n}} \ \% \, N)$$

Theorem: Assuming that it is infeasible to factor N, the output of the function $f = f_{N,g,a,r}$ is indistinguishable from random bits, meaning that the sequence

$$f(1), f(2), f(3), \ldots, f(t)$$

with t much smaller than N, is indistinguishable from a sequence of random bits.

Remark: The proof is easier than for BBS but still non-trivial.