The Mersenne Twister and Cryptographically-Secure PRNGs

Henry Stoen

August 2024

1 Introduction

Obtaining mathematical randomness has proven to be a hallmark of modern computer simulation. Computers are inherently deterministic and predictable, two conventions which run contrary to the idea of random number generation. Current applications are manyfold: computer modeling and simulating, statistics, and namely, cryptography. Padding of ciphertexts and key generation are two examples. However, there exist two methods that have been uncovered to produce consistent streams of random numbers, Pseudo-random Number Generators (PRNGs), and True Random Number Generators (TRNGs). TRNGs make use of atmospheric noise or fluctuations, which tend to behave with true unpredictability. These work for smaller scale examples, but lack the speed of calculation necessary for many computer programming purposes. Thus PRNGs, based on mathematical algorithms, are more feasible for widespread computer usage. However due to the required processes, PRNGs can never be truly random as an initial seed of some kind is necessary to generate sequences of numbers. The Mersenne Twister (MT) was released in 1997 by Makoto Matsumoto and Takuji Nishimura. MT satisfies many requirement of the ideal PRNG, namely independence, uniformity, and efficiency. Its period is based on the Mersenne Prime $2^{19937} - 1$, a uniquely long number which bodes well for its security. It also has proved successful against statistical measures such as K-Distribution and Diehard tests. That being said, it is not cryptographically secure as it's based on Linear Recurrence, a method known for its predictability like in the case of the Linear Congruential Generator, a widely used PRNG in the past that is now revered as vastly unsecure.

2 Linear Congruential Generator

The linear congruential generator (LCG) is a rudimentary and simply PRNG that uses a piecewise linear function. This expedites the algorithm, resulting in quickly generated numbers, though these are generally easily predictable. This predictability can be observed through a Spectral Test, where hyperplanes will clearly reveal themselves.

The generator operates based upon recurrence:

$X_{n+1} = (aX_n + c) \bmod m$

where:

1. X is the sequence of pseudo-random values
2. m, $0<m$ is the modulus
3. a, $0<a<m$ is the multiplier
4. c, $0<c<m$ is the increment
5. X_0, $0<X_0<m$ is the start value

2.1 Limitations

The period length can prove valuable to increase the successfulness of the algorithm, but can also greatly compromise it. Appropriate parameters are the difference maker that can result in a long period length. If a and c are both set to one, for example, the string of numbers will be a modulo m counter, obviously non random. Logically, there exist known parameters that yield better results. However, even then, it fails against a well established algorithm like the Mersenne Twister, which will be discussed next.

3 Mersenne Twister

The generator algorithm is a modification of a Twisted Generalized Feedback Shift Register (TGFSR). The standard implementation uses a 32 bit word length. Simply put, the Mersenne Twister relies on matrix linear recurrence over a binary field f_2.

3.1 Initialization

The MT algorithm begins by initializing an internal state array with a length N, (usually of value 624 for MT19937). The array is primed using a seed value. By definition, the Mersenne Twister is a PRNG, which is therefore determinstic. This in an inherent, unavoidable weakness, so therefore we must generate unique seeds for which the algorithm operates upon. The process for initialization is as follows:

seed= $x_1 = (f \cdot (x_{i-1} \bigoplus (x_{i-1} \gg (w-2))) + i) \bmod 2^w$

where:

1. x is the state array
2. i is the index of said array
3. f is the constant 0x6c078965
4. w is the word size, (32 bits for version MT19937)
5. $\gg$ is the right bitwise shift operator
6. $\bigoplus$ is the bitwise XOR operator

3.2 MT Theories

The Mersenne Twister generates a sequence of word vectors which act as pseudo-random inters between 0 and $2^w - 1$, where w is the dimension of row vectors over Finite field f_2. The MT algorithm is based upon the a recurring equation. A recurrence relation is an equation which recursively defines a sequence given prior terms. The equation is as follows:

$$x_{k+n} := x_{k+m} \bigoplus (x_k^u | x_{k+1}^l) A$$

Where:
Integer n is the degree of recurrence
Integer m, 1≤m≤n
A is a constant w × w matrix chosen to ease the following multiplication

$$\begin{bmatrix} 0 & 1 & & & & \\ 0 & 0 & 1 & & & \\ 0 & \cdots & \cdots & \ddots & & \\ & & & & \ddots & \\ & & & & & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & \cdots & a_0 \end{bmatrix}$$

k is 0,1,2...
x_n is a row vector of a word size w generated by k=0
$x_0, x_1 ... x_{n-1}$ are seeds
x_{k+1}^l are lower or rightmost bits of x_{k+1}
x_k^u are upper or leftmost bits of x_k
$\bigoplus$ is a bitwise XOR
| denotes a concatenating operation
$x_k^u | x_{k+1}^l$ is the vector obtained by concatenating the upper w - r bits of x_k and the lower r bits of x_{k+1} in order. The constant matrix A us then multiplied by the right by this new vector. The bitwise addition operation is then performed to add x_{k+m}, and then generates x_{k+n}. The multiplication can be performed with the pair of bitwise operations: $xA = \begin{cases} \{x \gg 1 & \text{if } x_0 = 0 \\ \{(x \gg 1) \bigoplus a & \text{if } x_0 = 1 \end{cases}$
Where: x is $(x_{w-1}, x_{w-2}, ..., x_0)$
a is the bottom row vector of A
$\gg$ denotes right bitwise shift

The k-distribution is a statistical measure that can be used to assess the success in equidistribution. However, it is improved through a method known as tempering. Each generated word is multiplied by a w × w invertible matrix T from right yielding a result of tempering matrix x into z:=xT. T is chosen such that binary operations can be performed as:

1. $y := x \bigoplus (x \gg u)$
2. $y := y \bigoplus = ((y \ll s) \& b)$

3. $y := y \bigoplus = ((y \ll t) \& c)$

4. $z := y \bigoplus = (y \gg l)$

where:
u,s,t,l are shift constants (11,7,15,18 for MT19937)
b and c are tempering bitmasks
$\ll$ denotes a bitwise left shift
& denotes an and operation

These are transformations which temper each word via bitshifting.
The shift register has 624 elements and a 19937 cell output. The following steps show the workings of the Mersenne Twister algorithm.

```
Step 0. Create a bitmask for upper and lower bits
$u \leftarrow \underbrace{1...1}_{w-r}\ \underbrace{0...0}_{r}$ ;(bitmask for upper w-r bits)
$ll \leftarrow \underbrace{0...0}_{w-r}\ \underbrace{1...1}_{r}$ ;(bitmask for lower r bits)
$a \leftarrow a_{w-1}a_{w-2}...a_1a_0$

Step 1.
$i \leftarrow 0$
$x[0], x[1], ..., x[n-1]$

Step 2.
$y \leftarrow (x[i]\&u)\mathrm{OR}x[(i+1) \bmod n]\&\mathrm{ll}$
;computing$(x_k^u|x_{k+1}^l)$

Step 3.
$x[i] \leftarrow x[(i+m) \bmod n]\mathrm{XOR}(y \ll 1)$

$\mathrm{XOR} \begin{cases} \{0 & \text{if the least significant bit } y = 0 \\ \{a & \text{if the least significant bit } y = 1 \end{cases}$
;(multiplying A)

Step 4.
;(Calculate x[i]T)

$y \leftarrow x[i]$

$y \leftarrow y$ XOR $(y \gg u)$ ;(shiftright y by u bits and add to y)

$y \leftarrow y$ XOR $(y \ll s)$

$y \leftarrow y$ XOR $(y \ll t)$

$y \leftarrow y$ XOR $(y \gg l)$

output y
```

Step 5.
$i \leftarrow (i+1) \bmod n$

Step 6.
Return to Step 2.

It is worth noting the parameters importance on the algorithms effectiveness. There are many areas where difference in value can greatly effect the outcome, such as: period parameters, intege]r parameters (w size), the degree of recursion n, the middle term m, the separation point r, vector parameter a (Matrix A), integers parameters u,s,t,l, and vector parameters b and c. The original publishing on the Mersenne Twister goes into great depth on how to accurately choose these values.

4 Limitations

The initialization earlier described is one of the largest issues associated with the initial version of the Mersenne Twister. Should the initial state possess disproportionately many zeros, then the generated sequence of numbers will similarly contain an excess of zeroes for as many as 1000 iterated generations. If seeds are systematically chosen, such as being multiples, this will have a negative bearing on the generated values. These issues were realized by Matsumoto and Nishimura, and fixed the implementation code.

Another relevant issue is pertinent to its use cryptographically. The Mersenne Twister's original variant was not preferred for secure usage due to its relative predictability. Due to its large output of 19937 bits and linearity, the next state can be obtained. To originally overcome this discrepancy, the outputs were run through a hash function known as SHA-1. Simply put, a hash function is an algorithm which takes an input and returns a fixed-size string of bytes, typically mapping it in a unique way. These functions have widespread influence within the field of cryptography. The CryptMT variant was based upon this principle.

5 CryptMT

CryptMT is a stream cipher, an encryption technique that parses byte by byte, transforming plain text into code via a certain key. The algorithm variant makes significant changes to MT to enhance cryptographic usage. Additional bitwise operations are performed along with non-linear transformations, which proves largely successful to the pseudo-random nature of the output string. It retains the uniquely large period held by MT. These modifications are not bulletproof to attacks, but they do obfuscate the process, resulting in an initial state that's much more difficult to reverse-engineer. It is largely viewed as safe for data-encryption and other similarly important use cases.

6 Bibliography

1. M. Matsumoto and T. Nishimura, ”Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator” ACM Trans. on Modeling and Computer Simulation, vol. 8, no. 1, pp. 3-30, Jan. 1998.
2. A. Jagganata, ”Mersenne Twister – A Pseudo Random Number Generator and its Variants” CiteSeerx
3. M. Matsumoto, M. Saito, T. Nishimura and M. Hatagi, “Cryptographic Mersenne Twister and Fubuki stream/block cipher” Cryptographic ePrint Archive, June 2005.
4. P. L’Ecuyer, ”Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure” Mathematics of Computation, Jan. 1999