Mersenne Twister – A Pseudo Random Number Generator and its Variants

Archana Jagannatam

Abstract:

Random number generators(RNG) are widely being used in number of applications, particularly simulation and cryptography. They are a critical part of many cryptographic systems such as key generation, initialization vectors, message padding, nonces and many more. This paper discusses about the Mersenne Twister(MT), a pseudo random number generator(PRNG) and its variants. It mainly emphasizes on two of its variants. SIMD-Oriented Fast Mersenne Twister(SFMT) which is a 128-bit PRNG analogous to MT making full use of its features. And the cryptographically secure CryptMT, considered to be one of the fastest stream ciphers on a CPU with SIMD operations. It also briefly discusses the theories and the choice of parameters used in the algorithms. The requirements for a PRNG to be certified as a good and cryptographically secure PRNG will be presented.

1. Introduction:

Random number generators are devices that generate a series of numbers or some kind of symbols that appear random. RNG'S are used for a variety of purposes such as simulating, modeling complex phenomena, cryptography and of course ever popular for games and gambling. There are two main approaches to generating random numbers, Pseudo Random Number Generators(PRNG) and True Random Number Generators(TRNG). TRNG extracts randomness from physical phenomena like atmospheric noise, little variations in mouse movements or the time between mouse strokes and feed then in to a computer. They are mainly used in games and gambling where genuine randomness is required, as they are to slow for use in statistical and cryptographic applications. In comparison to TRNG, PRNG's are algorithms using some kind of mathematical formula or pre-calculated tables to generate a sequence of numbers that appear random. The sequence of numbers produced is not truly random. It is completely determined by an arbitrary initial state called seed state. If a PRNG's internal state contains *n* bits, its period can be no longer than 2^n results. PRNG's are efficient, deterministic and periodic which makes them suitable for applications where many numbers are required and where it is useful that the same sequence can be replayed easily such as simulation and modeling. A PRNG should posses some requirements to be considered as a good PRNG, such as uniformity, independence, long period, proper initialization, unpredictability, efficiency and portability. The quality of randomness of a PRNG is measured by subjecting it to a set of statistical testes called diehard tests. Every cryptographically secure PRNG should satisfy two basic requirements, next-bit test and state compromise extensions. This paper discusses Mersenne Twister(MT) which is a PRNG and which satisfies all the requirements to be certified as a good PRNG. MT is proposed in 1997 by **Makoto Matsumoto** and **Takuji Nishimura**. It provides for fast generation of very high-quality pseudorandom numbers with a long period length which is chosen to be a Mersenne prime, high order of dimensional equidistribution, speed and reliability. Later many

different variants of MT has been introduced for better speed and security for cryptographic uses. Two of the variants this paper talks about are SIMD-Oriented Fast Mersenne Twister(SFMT) and cryptographically secure CryptMT. SFMT is introduced by Mutsuo Saito and **Makoto Matsumoto** in 2006. It is typically a Linear Feedback Shift Register(LFSR) which generates a 128-bit pseudorandom integer at one step. SFMT is introduced to use the new features of CPU's such as Single Instruction Multiple Data (SIMD) operations (i.e., 128bit operations) and multi-stage pipelines. CryptMT is a stream cipher which uses MT and is considered to be cryptographically secure. It was developed by **Makoto Matsumoto**, Hagita Mariko, **Takuji Nishimura** and Mutsuo Saito in 2005. CryptMT ver. 3 is the latest version. In section 2.1 of this paper we discuss the theories behind MT and its algorithm. Section 2.2 discusses the choice of parameters for MT also presents a table showing values for different parameters of 32-bit MT. Section 2.3 talks about the limitations of MT such as initialization and security, and also the modification made to overcome the limitations. Section 3 introduces SFMT and 3.1 shows the theories and algorithms for SFMT, and also contains two figures of a circuit like description of SFMT19937 and block generation scheme. Next we talk about the choice of parameters of SFMT for the calculation of period and dimension of equidistribution in section 3.2. Then in 3.3 its limitation such as recovery from 0-excess state and the improvements made are discussed. Section 4 introduces CryptMT, we discuss its first version, the theories behind it and presents a block diagram of CryptMT ver. 1. Section 4.1 talks about its latest version CryptMT ver. 3, its theories and contains a figure showing combined generator. It also shows a table containing the parameters for the speed comparison of CryptMT with other stream ciphers. Then its advantages and shortcomings are discussed in section 4.2. The appendix of the paper shows the inversive-decimation method for primitivity testing.

2. Mersenne Twister:

Mersenne Twister, the name derives from the fact that it uses a period which is a Mersenne prime. It is a modification of a Twisted Generalized Feedback Shift Register(TGFSR) which takes in an incomplete array to realize a Mersenne prime as its period and uses an inversive-decimation method for primitivity testing of a characteristic polynomial of a linear recurrence with a computational complexity of $O(p^2)$ where p is the degree of the polynomial. MT has a long period of 2^{19937}-1and a 623 dimensional equidistribution up to 32-bit accuracy, generating an output which is free of long-term correlations. It is considered to be fast, as it avoids multiplications and divisions and uses the advantages of cashes and pipelines and efficient in memory use as only 624 words needed for the working area.

2.1 MT Theories:

MT generates a sequence of word vectors considered as uniform pseudorandom integers between 0 and 2^w-1, where w is the dimension of row vectors over the finite binary field F_2. It is based on the following recurring equation:

$$x_{k+n} := x_{k+m} \oplus (x_k^u \mid x_{k+1}^l)A \qquad (1)$$

Here:

Integer n is the degree of recurrence.
Integer m, $1 \leq m \leq n$.
A is a constant $w \times w$ matrix chosen to ease the matrix multiplication shown below

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

k is 0,1,2……
x_n is a row vector of a word size w which is generated with k =0.
x_0, x_1, …., x_{n-1} are initial seeds.

x_{k+1}^l are lower or rightmost r bits of x_{k+1}.

x_k^u are upper or leftmost $w - r$ bits of x_k.

$\oplus$ denotes a bitwise XOR.
| denotes a concatenating operation.

$(x_k^u \mid x_{k+1}^l)$ is the concatenation vector obtained by concatenating the upper w - r bits of x_k and the lower r bits of x_{k+1}^l in the order. Then the matrix A is multiplied from the right by this vector. Finally bitwise addition $\oplus$ is performed to add x_{k+m}, and then generate the next vector x_{k+n}. The multiplication of $(x_k^u \mid x_{k+1}^l)A$ can be performed using simple bit shift operations as shown below.

$$xA = \begin{cases} x >> 1 & if\ x_0 = 0 \\ (x >> 1) \oplus a & if\ x_0 = 1 \end{cases}$$

Where x is $(x_{w-1}, x_{w-2}, ..., x_0)$,

a is the bottom row vector of A and

‘ >>’ denotes bitwise right shifts.

Thus the calculation of the recurrence (1) is realized with bitshift, bitwise EXCLUSIVE-OR, bitwise OR, and bitwise AND operations.

MT has (n-1) dimensional equidistribution, which is a good characteristic of PRNG’s. k-distribution tests are considered to be a good way to measure the randomness of a PRNG. From the definition, a periodic sequence $X := x_0, x_1, ..., x_{p-1}$ with period p of w-bit integers is said to have k-dimensional equidistribution property if it has the maximum period and if any kw-bit pattern equally likely occurs as k-tuple $(x_l, x_{l+1}, ..., x_{l+k-1})$ for $l = 0,1,2,...$. All the overlapping k-tuples appearing in one period, then the output k-tuples are uniformly distributed over the set of possible output numbers. A largest value of such k is called the dimension of equidistribution. To improve the k-distribution to v-bit accuracy of the raw sequences generated from the recursion (1), a method called tempering is used to produce the final pseudo

random number. Each generated word is multiplied by a $w \times w$ invertible matrix T from right yielding a result of tempering matrix x into $z := xT$. The matrix T is chosen such that binary operations can be performed as:

$$y := x \oplus (x >> u)$$

$$y := y \oplus ((y << s) \,\&\, b)$$

$$y := y \oplus ((y << t) \,\&\, c)$$

$$z := y \oplus (y >> l)$$

where $u, s, t \text{ and } l$ are tempering bit shifts.
b and c are tempering bitmasks.
'<<' denotes a bitwise left shift and
'&' denotes a bitwise AND operation.

MT works in two parts: recurring and tempering. The recurring is a form of Linear Feedback Shift Register(LFSR) in which each bit of the state derives from the recursion and each bit of the output also satisfies the recurring of the bits forming the states. Figure 1 shows a high level block diagram of MT.

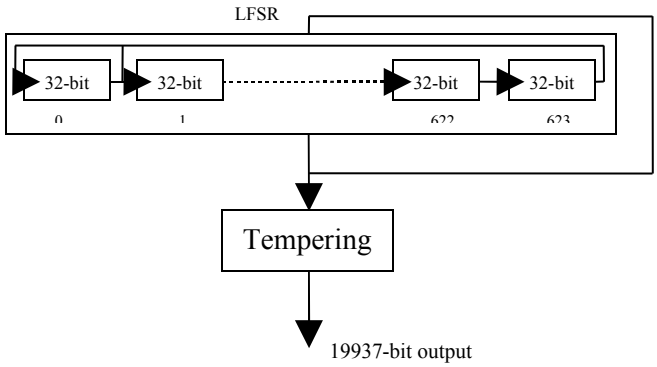

Fig 1. Block diagram of MT

The shift register is composed of 624 elements and a total of 19937 cells. Each element has a 32-bit length except for the first element which has only 1bit due to the bit-discarding. Following steps shows the working of an MT algorithm:

Step 0. Create bitmask for upper and lower bits

$u \leftarrow \underbrace{1.....1}_{w-1}\underbrace{0\cdots0}_{r},$ bit mask of upper w-r bits,

$ll \leftarrow \underbrace{0\cdots0}_{w-r}\underbrace{1.....1}_{r},$ bit mask of lower r bits,

$a \leftarrow a_{w-1}a_{w-2}.....a_1a_0,$ the last row of matrix A.

Step 1. Initialize the x array with seeds of nonzero values.
$x[0], x[1], ..., x[n-1]$

Step 2. Compute $(x_i^u \mid x_{i+1}^l)$, where the upper bits of $x[i]$ are concatenated with the lower bits of $x[i+1]$

$y \leftarrow (x[i]\ AND\ u)\ OR\ (x[(i+1) \bmod n]\ AND\ ll)$

Step 3. Calculate the next state

$$x[i] \leftarrow x[(i+m) \bmod n] \oplus (y >> 1) \oplus \begin{cases} 0 & if\ LSB\ of\ y = 0 \\ a & if\ LSB\ of\ y = 1 \end{cases}$$

Step 4. Multiply $x[i]$ by the tempering matrix T for better equidistribution

$y \leftarrow x[i]$

$y \leftarrow y \oplus (y >> u)$

$y \leftarrow y \oplus ((y << s)\ \&\ b)$

$y \leftarrow y \oplus ((y << t)\ \&\ c)$

$z \leftarrow y \oplus (y >> l)$

output y

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

Step 6. Repeat the process
Go to step 2.

2.2 Choice of parameters:

The choice of parameters for MT are carefully made so as to satisfy the above mentioned properties. There are two classes of parameters to be chosen: period parameters for determining the period and tempering parameters for k-distribution. The parameters n and r are selected so that the characteristic polynomial is primitive or $nw-r$ is equal to the Mersenne exponent of 19937. The value *w* is the word size of the computer and here it is 32-bit. The value of the last row of matrix *A* is chosen randomly and the values $n, m, r\ and\ w$ are fixed. The tempering parameters must be chosen to satisfy the k-distribution test. *k*(*v*) has to be near a multiple of n, $((nw-r)/v)$ being its upper bound. Table I shows the list parameters for 32-bit MT

2.3 Limitation and Improvement:

One of the limitations of MT is the initialization. If the initial state has too many zeros then the generated sequence may also contain many zeros for more than 10000 generations and if the seeds are chosen systematically such as 0, 20, 30….. the output sequences will be correlated. The use of 32-bit words for initial seeds in MT poses a problem known as birthday paradox. Four years later the inventors corrected these limitations in their implementation code. To fix the latter problem they proposed an array of arbitrary length as an initial seed and also manipulated the initializing routine so that any one bit change in the initial seed will result in dramatic changes in the initial state.

Another limitation of original MT is not preferred for cryptographic purposes. It is easy to predict the next state given the present outputs in MT, because of its linearity and the large output of 19937 bits. To overcome this limitation the inventors recommended to have the outputs of MT gone through a hash function such as SHA-1. Based on this aspect, a cryptographically secure generator called CryptMt was designed. CryptMT is discussed in section 4 of this paper.

TABLE I
PARAMETERS OF 32-BIT MT19937

Parameter	Quantity
n	624
w	32
r	31
m	397
a	99083B0DF
u	11
s	7
t	15
l	18
b	9D2C5680
c	EFC60000

3. SIMD-Oriented Fast Mersenne Twister:

SFMT is the 128-bit variant of MT, which takes the advantage of new features of CPU's such as parallel processing, namely Single Instruction Multiple Data(SIMD) and multi-stage pipelines. It is twice as fast as original MT from fact that it uses SIMD operations and a block-generation function, which fills an array of 32-bit integers in one call makes it two times faster than original MT if implemented with SIMD. It supports various periods from 2^{607}-1 to 2^{216091}-1 with a better dimensional equidistribution property of *v*-bit accuracy than MT.

3.1 SFMT Theories:

SFMT is a LFSR generator based on a recursion over finite binary field. It uses the following recurrence equation:

$$x_{k+n} := x_k A \oplus x_{k+m} B \oplus x_{n-2} C \oplus x_{n-1} D$$

Where A, B, C and D are 128 × 128 matrices over F_2 field perform the computations with few SIMD bit-operations. The choice of the suffixes n-2 and n-1 are for speed considerations. Reading x_k and x_{k+m} from memory takes time. SFMT benefits from the fact that it computes xn-2C and xn-1D in the mean time, because copies of x_{n-2} and x_{n-1} are kept in registers. The linear transformations of A, B, C and D are as follows:

1. $x_k A := (x_k \overset{128}{<<} d) \oplus x_k$

The result x_kA is the left shift of x_k by d(=8) bits ex-ored with x_k itself and x_k is a single 128-bit integer.

2. $x_{k+m} B := (x_k \overset{32}{>>} e) \,\&\, g$

x_{k+m} is a quadruple of 32-bit integers, and each 32-bit integer is shifted to right by e(=11) bits. The notation & means the bitwise AND with a constant g, having a value of BFFFFFF6 BFFAFFFF DDFECB7F DFFFFFEF in its hexadecimal form.

3. $x_{n-2} C := (x_k \overset{128}{>>} d)$

The result is the right shift of x_k, a 128-bit integer by d(=8) bits.

4. $x_{n-1} D := (x_k \overset{32}{<<} h)$

x_{n-1} is a quadruple of 32-bit integers. The result is obtained by left shift of x_k by h(=18) bits.
Figure 2 shows the functioning of SFMT19937 with period a multiple of 2^{19937}-1.

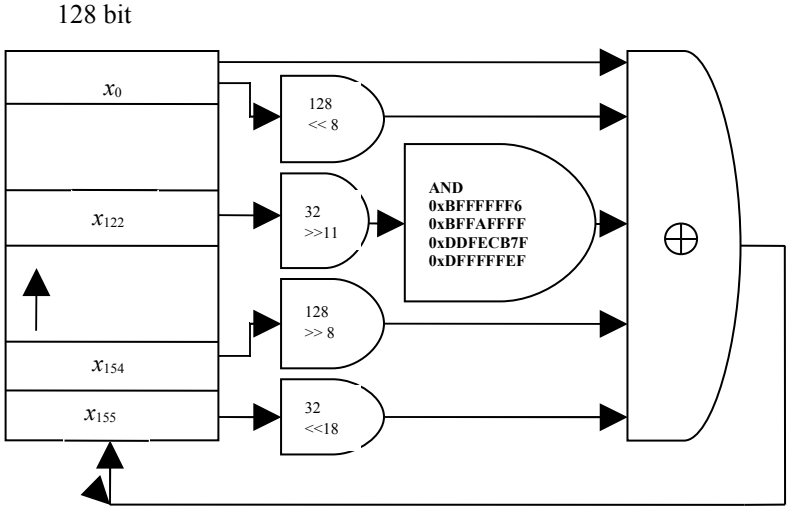

Fig 2. A circuit like description of SFMT19937

The SFMT implements a scheme called block generation where a w-bit array is filled up with pseudorandom integers in one call. Block generation is introduced to avoid delay of function call like branch conditions. In this scheme an array of w-bit integers of length L is specified, where w 32 or 64 bits

and L is chosen such that wL is a multiple of 128 and no less than $N \times 128$ (N being 156). Figure 3 describes the block generation scheme. An internal array of 128-bit integers of length 156 is concatenated with the state array using the indexing technique. Then a user-specified array of 128-bit integers is generated by recursion until it fills up the array, and the last 156 128-bit integers are copied back to the internal array.

3.2 Choice of Parameters:

Computation of period and the dimension of equidistribution are the two critical factors in SFMT. The choice of parameters must be made such that the calculated period and the dimension of equidistribution must satisfy the requirements of the generator. An LFSR with the state transition function f: $S \rightarrow S$, S being the state space and X_f the characteristic polynomial is considered an automaton. If X_f is primitive, then the period of the state transition takes the maximal value $2^{\dim(S)} - 1$. To check for primitivity, factorization of this huge number is hard, so SFMT adopts a method called the reducible transition method (RTM) which avoids integer factorization. The parameters for the recursion of LFSR are chosen randomly. All the factors of small degree are from X_f are removed until it has no irreducible factor of degree p, or that it has a factor of degree p, where p being the Mersenne exponent. Hence by definition, the period of SFMT19937 as a 128-bit integer generator is a nonzero multiple of $2^{19937} - 1$, if the 32 MSBs of x_0 are set to the value 6d736d6d in hexadecimal form. The value of the dimension of equdistribution depends on the defect ratio of the system. By definition, the defect ratio up to $1/(P+1)$ is allowed to claim the dimension of equidistribution. If $P = 2^{19937} - 1$, then $1/(P+1) = 2^{-19937}$. In the following, the dimension of equidistribution allows the defect ratio up to 2^{-19937}.

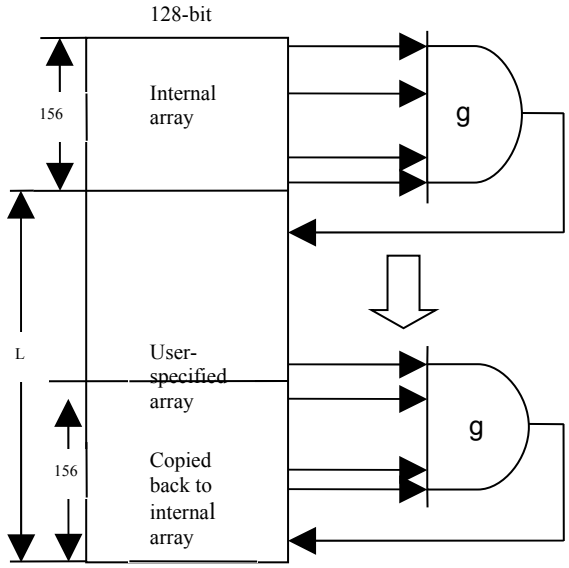

Fig 3. Block-generation scheme

3.3 Limitation and Improvement:

SFMT has a limitation of recovery from 0-excess states. By definition a LFSR with a sparse feedback function has the phenomenon: if the bits in the state space contain too many 0's and few 1's (called a 0-excess state), then this tendency continues for considerable generations, since only a small part is changed in the state array at one generation, and the change is not well-reflected to the next generation because of the sparseness. A method was introduced in [11] to measure the recovery time from 0-excess states: generate k pseudorandom numbers from an initial state with only one bit being 1 and discard them. Compute the ratio of 1's among the next 1000 pseudorandom numbers of 32-bit integers and take the average of them. The speed of recovery from 0-excess states is a trade-off with the speed of generation.
The number of initializations are much smaller than the generations in typical simulations and hence by well-designed initialization, 0-excess initial state can be avoided. A new version of SFMT was introduced in 2007 known as Double precision SIMD-oriented Fast Mersenne Twister (DSFMT). It is the faster version of SFMT and also supports various periods from 2^{607}-1 to 2^{132049}-1. It directly generates double precision floating point pseudorandom numbers by avoiding the conversion of integer to double(floating point).

4 CryptMT:

CryptMT internally uses MT. It is stream cipher which has been submitted to the eSTREAM project of the eCRYPT network. CryptMT ver. 1 is the first implementation of MT as a cryptographic secure generator. It contains a combination of MT as a mother generator and a multiplicative filter with 32-bit memory. It computes the accumulative product of the output of MT, and uses the most significant 8 bits. The concatenation of key and IV is passed to the initialization scheme of MT. A variable accum of word size is used with its initial value set to 1. This type of filter with memory, based on multiplication and use of MSB's is called as a multiplicative filter. Then the following set of steps is processed to obtain a cryptographically secure sequence of pseudorandom numbers of 8-bit integers.
(1) Generate one pseudorandom word gen rand by MT.
(2) Multiply it to accum: accum ← accum × (gen rand | 1).
(3) Output the most significant 8 bits of accum. Go to Step 1.
"|" denotes bitwise-OR operation.
The internal state of the MT is sufficient enough to take care of time-memory-trade-off attacks making CryptMT fast. It is 1.5 – 2.0 times faster than the optimized counter-mode AES. The reason that only most significant 8-bits of the output are used is because it will be hard to obtain the internal state of MT at any time. The most significant bits are safe, since the bit-diffusion pattern of the multiplication is from right to left, and most significant bits gather information of all the less significant bits of the two operands, accum and the output of MT. And hence it is clear that the security of CryptMT largely depends on the mother generator MT. Figure 4 shows the block diagram of CryptMT.
One of the advantages of CryptMT over other ciphers is that the key size and IV size are variable and can be specified by the users, both up to 2048 bits. And the other is its internal size of 19937+32 bits with a period of 2^{19937}-1.
The latest version of CryptMT is CryptMT ver. 3. It is proposed to have faster speed than CryptMT by using the new features of CPU's like SIMD and parallel pipelining.

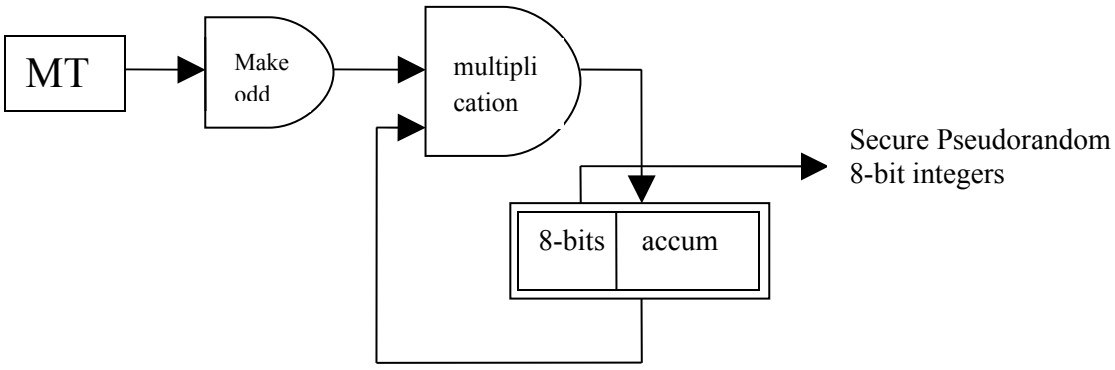

Fig 4. Block diagram of CryptMT ver. 1

4.1 CryptMT ver. 3:

It uses a variant of SFMT as its LFSR, and a non-linear multiplicative filter and a booter. Its period is a non-zero multiple of 2^{19937}-1 and a dimension of equidistribution of at least 1241. It also possesses a property that, any bit of the 8-bit integer stream generated by CryptMT Ver.3 has a period that is a multiple of 2^{19937} -1. It uses the version of SFMT that generates 128-bit integers by the following recursion:

$$x_{N+j} := (x_{N+j-1} \,\&\, \text{128-bit MASK}) \oplus (x_{M+j} >> 64\ S) \oplus x_{M+j}[2][0][3][1]) \oplus x_j[0][3][2][1]) \qquad (2)$$

The first term is a result of the bit wise AND of x_{N+j-1} and a constant 128-bit MASK given by ffdfafdf f5dabfff ffdbffff ef7bffff in its hexa-decimal form. The second term is the concatenation of two 64-bit integers (x_{M+j} [3][2] >> S) and (x_{M+j} [1][0] >> S), where S = 3, N = 156 and M = 108. Third term is a permutation of four 32-bit integers in x_{M+j}, and the last term is a rotation of those in x_j.

The filter is given by:

$$f(y; x) := y \times (x|1) \bmod 2^{32}; \quad g(y) := 8 \text{ MSBs of } y \qquad (3)$$

(x|1) shows that x is set with its LSB as 1.
8 MSBs means 8 most significant bits of y.
By setting the LSB to 1, the multiplication is restricted to odd integers in the ring of 2^{32}, avoiding degenerations.
Figure 5 shows a block diagram of the combination generator.

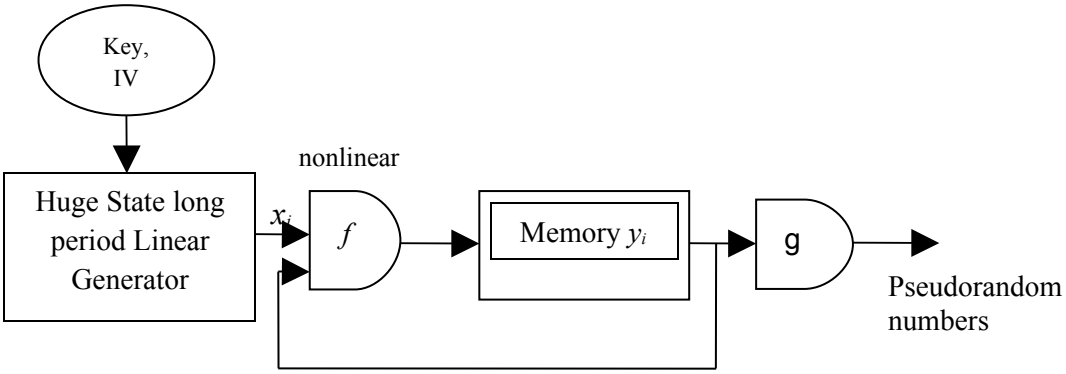

Fig. 5 Combined generator = linear generator + filter with memory.

As CryptMT is a stream cipher the key and IV initialization is the critical part in its functioning. The key and IV used for the initialization has to be expanded to a sequence of 128-bit integers, which happens to be expensive when the message length is much less than $N \times 128$. Hence the new aspect introduced, is to use a smaller PRNG called the Booter. It generates a shorter sequence efficiently. The output of the booter is simultaneously used to pass to the filter and also fill the state of SFMT.

Speed is one of the important property that has to be possessed by a stream cipher. Table II shows the comparison of number of cycles consumed per byte, by CryptMT and some of the other stream ciphers in three different CPU's. It also shows the number of cycles consumed for Key and IV set-up.

Table II. Summary from eSTREAM benchmark by Bernstein[18]

Primitive	Core 2 Duo			AMD Athlon 64 X2			Motorola PowerPC G4		
	Stream	Key setup	IV setup	Stream	Key setup	IV setup	Stream	Key setup	IV setup
CryptMT3	2.95	61.41	514.42	4.73	107.00	505.64	9.23	90.71	732.80
HC-256	3.42	61.31	83805.33	4.26	105.11	88726.20	6.17	87.71	71392.00
SOSEMANUK	3.67	848.51	624.99	4.41	1183.69	474.13	6.17	1797.03	590.47
SNOW-2.0	4.03	90.42	469.02	4.86	110.70	567.00	7.06	107.81	719.38
Salsa20	7.12	19.71	14.62	7.64	61.22	51.09	4.24	69.81	42.12
Dragon	7.61	121.42	1241.67	8.11	120.21	1469.43	8.39	134.60	1567.54
AES-CTR	19.08	625.44	18.90	20.42	905.65	50.00	34.81	305.81	34.11

It can be seen that, CryptMT3 is the fastest in generation in Intel Core 2 Duo CPU, reflecting the efficiency of SIMD operations. It is slower in Motorola PowerPC due to the lack of 32-bit integer multiplication in Motorola.

4.2 Advantages and Shortcomings:

CryptMT ver. 3 has many advantages over many other stream ciphers proposed. It is 1.8 times faster than the first version. It has a variable Key and IV size from 128-bits to 2048-bits. The size of the state and the length of the period make time-memory-trade-off attacks infeasible, and the high non-linearity introduced by the integer multiplication would make any kind of algebraic attacks impossible. It has no look-up table and hence has resistance to cache timing attacks.

Despite of all the above mentioned advantages CryptMT ver. 3 is not selected for eSTREAM portfolio. It is not because of the defect of CryptMT. The concern is that the security of the cipher, in particular the non-linear filter component is not yet well understood and insufficient confidence in the design.

5. Conclusion:

Mersenne Twister is theoretically proven to be a good PRNG, with a long period and high equidistribution. It is extensively used in the fields of simulation and modulation. The defects found by the users have been corrected by the inventors. MT has been upgraded, to use and to be compatible with the newly emerging technologies of CPU's such as SIMD and parallel pipelines in its version of SFMT. SFMT is the much faster version of original MT, with a wide range of variable periods and better dimension of equidistribution. It has a good initialization scheme to overcome the limitations of original MT and has been theoretically proven. MT is also been upgraded to be cryptographically secure. CryptMT is the cryptographic secure version of MT. CryptMT ver. 1 uses MT as its mother generator and a multiplicative filter to generate pseudorandom numbers. It takes variable user specified values for key and IV. It is resistant to memory-trade-off-attacks and faster compared to other stream ciphers. CryptMT ver. 1 has been improved for faster speed using SFMT as its mother generator in its latest version CryptMT ver. 3. It is a 128-bit version of CryptMT ver.1 and also faster. It uses a component called booter in addition to the multiplicative filter, for the expansion of Key and IV to 128-bits. CryptMT is new and has not been widely used or truly evaluated it could be a good candidate for future researches and evaluations.

Appendix:

The inversive-decimation method:

The standard method of testing if a linear feedback shift register has maximum period is by first determining that it does, in fact, return to its initial state at the end of that period, and then establishing that it does not return to its initial state after any shorter period which is not a factor of the maximum period. The maximum period of an LFSR is always one less than a power of two.

A prime number is a number which has no factors, except, of course, one (so one must also test that the generator actually changes state after a single step).A prime number which is one less than a power of two is a Mersenne prime; hence, a shift register whose maximum period is a Mersenne prime is easier to test for maximum period. It is not so much the extra tests that are the problem as finding when to perform them by factoring the period when it is not a prime.

The same basic algorithm is used to run an LFSR for an enormous number of iterations as is used to raise numbers to large exponents when performing RSA; instead of thousands of repeated multiplications by the characteristic polynomial of the shift register, that polynomial can be squared, and the result squared, repeatedly.

The *inversive decimation* method achieves a modest, but important, increase in speed over the conventional method of testing an LFSR for maximum period; the time remains proportional to the length of the LFSR squared, but a factor proportional to the number of active taps on the LFSR is eliminated. This was sufficient to make testing the Mersenne Twister for maximum period practical. An essential step in this method is the ability to invert the operation of the LFSR; given a stream of bits produced by the shift register, it is necessary to be able to determine the state which gave rise to it. To make this easy, the bit which determines if the "magic number" A is XORed to the 32 bits of the array being processed is used as the output; thus, by knowing this output, one knows a very important piece of what is going on during the step that produced it. Then, to run the shift register backwards, we begin by noting that the bottom row of bits will be output without modification. The bit at position M in that row will then modify the last bit in that row as it moves to the next higher row, while the bits in the output sequence will modify what other bits in the shift register will need to have been, through applying the XOR vector A.

The truly bizarre step in this algorithm is that, instead of squaring the polynomial of the shift register, one merely needs to take every second bit in its output sequence, and then obtain the state which would generate those bits. This unusual result is said to be a consequence of the fact that only XOR, rather than modular arithmetic with a higher modulus, is used. To provide an example of this astounding property of binary LSFRs, we see the following:

0000001 1 1
1000000 0
0100000 0 0
0010000 0
0001000 0 0
0000100 0
0000010 1 1
1000001 1

1100000 0 0
0110000 0
0011000 0 0
0001100 0
0000110 1 1
1000011 0
0100001 1 1001001 1
1010000 0
0101000 0 1100100 0
0010100 0
0001010 1 0110010 1
1000101 1
1100010 1 1011001 1
1110001 1
1111000 0 1101100 0
0111100 0
0011110 1 0110110 1
1001111 0
0100111 0 1011011 0
0010011 0
0001001 1 0101101 1
1000100 0
0100010 1 1010110 1

Starting the generator from 0000001, we then take seven bits of the output of the 'twice as fast' generator to derive the contents which the same generator would need to produce that sequence, and indeed the sequences continue to correspond with the same polynomial. Squaring the sequence instead of the recurrence itself is what removes the dependence on the number of taps.

(From the paper: The Mersenne Twister. http://www.quadibloc.com/crypto/co4814.htm)

References:

[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] M. Matsumoto and Y. Kurita, "Twisted GFSR generators" ACM Trans. On Modeling and Computer Simulation, vol. 2, pp. 179-194, 1992.

[3] Mersenne Twister – A Very Fast Random Number Generator. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

[4] Paul C. Leopardi, "Mersenne Twister with improved initialization" (2002): default seed varies in different implementations. http://www.cygwin.com/ml/gsl-discuss/2006-q4/msg00014.html

[5] M. Matsumoto, M. Saito, H. Haramoto and T. Nishimura, "Pseudorandom number generation: impossibility and compromise" Journal of Universal Computer science, vol. 12, no.6, 2006.

[6] SIMD-oriented Fast Mersenne Twister (SFMT): twice faster than Mersenne Twister. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html

[7] M. Matsumoto and M. Saito, "SIMD-oriented fast Mersenne twister: a 128-bit pseudorandom number generator" in Proceeding of MCQMC, 2006.

[8] Papers on random number generators.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/earticles.html#sfmt
[10] Mutsuo Saito, "An Application of Finite Field: Design and Implementation of 128-bit Instruction-Based Fast Pseudorandom Number Generator", Master's Thesis, 2007.
[11] F. Panneton, P. L'Ecuyer, and M. Matsumoto. "Improved long-period generators based on linear reccurences modulo 2". ACM Transactions on Mathematical Software, 32(1):1–16, 2006.
[12] M. Matsumoto, M. Saito, T. Nishimura and M. Hatagi, "Cryptographic Mersenne Twister and Fubuki stream/block cipher" Cryptographic ePrint Archive, June 2005.
[13] M. Matsumoto, M. Saito, T. Nishimura and M. Hatagi, "CryptMT stream cipher version 3" eStream Proposal. http://www.ecrypt.eu.org/stream
[14] The cryptographic Mersenne Twister.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/CRYPTMT/index.html
[15] M. Matsumoto, M. Saito, T. Nishimura and M. Hatagi, "Crypt Stream Cipher Version 3", Cryptographic ePrint Archive, June 2006.
[16] Matsumoto, M., Saito, M., Nishimura, T. and Hagita, M. CryptMT Stream Cipher Version 3, SASC2007 Conference Volume.
[17] Matsumoto, M., Saito, M., Nishimura, T. and Hagita, M. CryptMT Version 2.0: a large state generator with faster initialization, SASC2006 Conference Volume.
http://www.ecrypt.eu.org/stream/cryptmtfubuki.html.
[18] Bernstein, D.J. http://cr.yp.to/streamciphers/timings.html.