A Fast Stream Cipher with Huge State Space and Quasigroup Filter for Software*

Makoto Matsumoto[1], Mutsuo Saito[2], Takuji Nishimura[3], and Mariko Hagita[4]

[1] Dept. of Math., Hiroshima University
m-mat@math.sci.hiroshima-u.ac.jp
[2] Dept. of Math., Hiroshima University
saito@math.sci.hiroshima-u.ac.jp
[3] Dept. of Math. Sci., Yamagata University
nisimura@sci.kj.yamagata-u.ac.jp
[4] Dept. of Info. Sci., Ochanomizu University
hagita@is.ocha.ac.jp

Abstract. Recent personal computers have high-spec CPUs and plenty of memory. The motivation of this study is to take these advantages in designing a tough and fast key-stream generator. Natural controversies on using a large state space for a generator are (1) effectiveness is unclear, (2) slower generation speed, (3) expensive initialization, and (4) costs in a hardware implementation.

Our proposal is to combine a linear feedback shift register (LFSR) and a uniform quasigroup filter with memory of wordsize. We prove theorems which assure the period and the distribution property of such generators, answering to (1). As for (2), the generation speed of a LFSR is independent of the state size. In addition, we propose a filter based on integer multiplication, which is rather fast in modern CPUs. We analyze the algebraic degree of such filters. We answer to (3) by a simple trick to use another small generator to initialize LFSR while outputting. We have no answer to (4), but comment that recent hardwares tend to have larger memory and sophisticated instructions.

As a concrete example, we propose CryptMT stream generator with period (no less than) $2^{19937}-1$, 1241-dimensional equidistribution property, which is sometimes faster than SNOW2.0 in modern CPUs.

Keywords: stream cipher, combined generator, filter with memory, quasigroup filter, multiplicative filter, CryptMT, eSTREAM, period, distribution.

1 Stream Cipher

In this article, we pursue a fast stream cipher in software. We assume that the machine has plenty of memory, and a fast integer multiplication instruction.

* This work is supported in part by JSPS Grant-In-Aid #16204002, #18654021, #18740044, #19204002 and JSPS Core-to-Core Program No.18005.

C. Adams, A. Miri, and M. Wiener (Eds.): SAC 2007, LNCS 4876, pp. 246–263, 2007.

Let B be the set of symbols. Throughout this article, we assume B to be the set of one byte integers, which is identified with ${\mathbb{F}_2}^8$, where $\mathbb{F}_2 = \{0, 1\}$ is the two-element field. We consider a stream cipher based on a key-stream generator over B. A generator receives a key k in the set of possible keys K, then generates a sequence of elements

$$b_0(k), b_1(k), \ldots, b_n(k), \ldots, \in B.$$

A plain text (a sequence of elements of B) is encrypted by taking bitwise exor with the sequence $(b_n(k))$, and then decrypted by the same method.

1.1 Combined Generator

Such a sequence is typically generated by a finite state automaton.

Definition 1. *A finite state automaton A without input is a quadruple $A = (S, f, O, o)$, where S is a finite set (the set of states), $f : S \to S$ is a function (the state transition function), O is a set (the set of the output symbols), and $o : S \to O$ is the output function.*

For a given initial state s_0, A changes the state by the recursion $s_n := f(s_{n-1})$ $(n = 1, 2, 3, \ldots)$ and generates the sequence

$$o(s_0), o(s_1), o(s_2), \ldots \in O.$$

For a stream cipher, we prepare an *initializing* function $\mathrm{init} : K \to S$, and take $O := B$. By setting $s_0 := \mathrm{init}(k)$, the automaton A generates a sequence of elements in B. Its period is bounded by $\#(S)$.

To obtain a secure generator, larger $\#(S)$ and complicated f and o are desirable. However, if f is complicated, then the analysis of the sequence (such as computing the period and the distribution) often becomes difficult. A typical choice is to choose an $\mathbb{F}_2$-linear transition function. We take $S := {\mathbb{F}_2}^d$ and choose a linear transition function f. Then, the period can be computed by the linear algebra and polynomial calculus. In particular, the following linear feedback shift register generators (LFSRs) are widely used: $S := ({\mathbb{F}_2}^w)^n$ where w is the word size of the machine (e.g. $w = 32$ for 32-bit machines), and the transition is

$$f(x_1, x_2, \ldots, x_{n-1}, x_n) := (x_2, x_3, \ldots, x_n, g(x_1, \ldots, x_n)). \tag{1}$$

Here $g : ({\mathbb{F}_2}^w)^n \to {\mathbb{F}_2}^w$ is a linear function called the feedback function. This state transition is equivalent to the recursion

$$x_{i+n} := g(x_i, x_{i+1}, \ldots, x_{i+n-1}) \quad (i = 0, 1, 2, \ldots).$$

The output of LFSR is given by

$$o : S \to {\mathbb{F}_2}^w, \quad (x_1, \ldots, x_n) \mapsto x_1,$$

which is not secure as it is. A software implementation technique using a cyclic array ([9, P.28 Algorithm A]) reduces the computation of f to that of g and an

index change. Consequently, the computation time is independent of the size of n, which allows a fast generator with huge state space. This type of generator is common for the pseudorandom number generation in Monte Carlo method (PRNG for MC), such as Mersenne Twister (MT19937) [10], whose period is $2^{19937}-1$.

As a stream cipher, any linear recurring sequence is vulnerable, so we need to introduce some non-linearity. A conventional method is to choose a "highly non-linear" $o : S \to O$. In this context, o is called *a filter*.

One of the estimators of the non-linear property of a function is the *algebraic degree*.

Definition 2. *Let $h(c_1, c_2, \ldots, c_n)$ be a boolean function, i.e.,*

$$h : {\mathbb{F}_2}^n \to \mathbb{F}_2.$$

Then, the function h can be represented as a polynomial function of n variables $c_1, c_2, \ldots, c_n$ with coefficients in $\mathbb{F}_2$, namely as a function

$$h = \sum_{T \subset \{1,2,\ldots,n\}} a_T c_T$$

holds, where $a_T \in \mathbb{F}_2$ and $c_T := \prod_{t \in T} c_t$. This representation is unique, and called the algebraic normal form *of h. Its degree is called the algebraic degree of h.*

Let $h_{i,n}(s_0)$ denote the i-th bit of the n-th output $b_n(s_0)$ of the generator for the initial state s_0. This is a boolean function, when we consider $s_0 \in S = {\mathbb{F}_2}^d$ as d variables of bit. Thus, an adversary can obtain s_0 by solving the simultaneous equations $h_{i,n}(s_0) = o_{i,n}$ for unknown s_0 for various i and n, where $o_{i,n}$ are the outputs of the generator observed by the adversary. This is the *algebraic attack* (see for example [4], [3]).

A problem of a linear generator with filter is the following. Since any s_n is a linear function of the bits in $s_0 = \mathrm{init}(k)$, the algebraic degree of $h_{i,n}(s_0)$ is bounded from the above by the algebraic degree of the i-th bit of the filter function o, namely that of the function

$$o_i : S \xrightarrow{o} {\mathbb{F}_2}^8 \xrightarrow{i\mathrm{th}} \mathbb{F}_2.$$

To attain the high-speed generation, o_i cannot access so many bits in S, and its algebraic degree is bounded by the number of accessed bits. This decreases the merits of the large state space. A *filter with memory*, which is just a finite state automaton with input, solves this conflict (see §2.3 for its effect on the algebraic degree).

Definition 3. *A finite state automaton A with input is a five-tuple $A = (S, I, f, O, o)$. The data S, O, o are same with Definition 1. The difference is that it has another component I (the set of input symbols), and that the state transition function is of the form $f : I \times S \to S$. For an initial state s_0 and an input sequence $i_0, i_1, \ldots \in I$, A changes the state by $s_n = f(i_{n-1}, s_{n-1})$ $(n = 1, 2, 3, \ldots)$.*

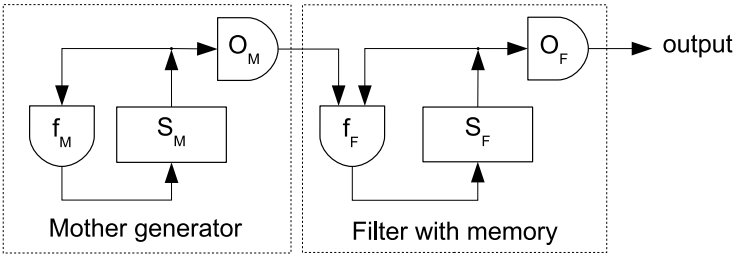

Fig. 1. Combined generator

Definition 4. *(A combined generator with filter with memory.)*
Let $A_M := (S_M, f_M, O_M, o_M)$ *be an automaton without input (called the* mother generator*, M for mother). Let* $A_F := (S_F, I_F, f_F, O_F, o_F)$ *be an automaton with input (called the* filter with memory*, F for the filter). We assume that* $O_M = I_F$*. Consider a pair of initial states* $s_{M,0} \in S_M$ *and* $s_{F,0} \in S_F$*. We generate a sequence of* $O_M = I_F$ *by* A_M *with initial state* $s_{M,0}$*, and pass it to* A_F *with initial state* $s_{F,0}$*, to obtain a sequence of* O_F *as the output sequence. This amounts to considering an automaton C without input, named the* combined generator*: the state space* S_C *of C is* $S_M \times S_F$*, the transition function is*

$$f_C : (s_M, s_F) \mapsto (f_M(s_M), f_F(o_M(s_M), s_F)),$$

and the output function is

$$o_C : (s_M, s_F) \mapsto o_F(s_F) \in O_F.$$

Figure 1 describes a combined generator.

Example 1. The output function o_C in the above definition depends only on S_F, but we may consider a function depending both S_M and S_F.

Such an example is famous SNOW stream cipher [5] [6]. The mother generator of SNOW2.0 is a LFSR with 512-bit state space, and its filter has 64-bit state space. Non-linearity is introduced by four copies of one same S-box of 8-bit size, based on arithmetic operations in 2^8-element filed $\mathbb{F}_{2^8}$.

SNOW has no rigorous assurance on the period and the distribution of the generated sequence. We shall introduce the notion of quasigroup filter, which allows to compute the period and distribution property.

2 Quasigroup Filter

Definition 5. *A function* $f : X \times Y \to Z$ *is said to be bi-bijective if* $f(-, y) : X \to Z,\ x \mapsto f(x, y)$ *is bijective for any fixed* y*, and so is* $f(x, -) : Y \to Z,\ y \mapsto f(x, y)$ *for any fixed* x*. If* $X = Y = Z$*, this coincides with the notion of a quasigroup.*

A quasigroup filter is an automaton in Definition 3 where the state transition function $f : I \times S \to S$ *is bi-bijective.*

Example 2. (Multiplicative filter)
Let $I = S$ be the set of odd integers in the ring $\mathbb{Z}/2^{32}$ of integers modulo 2^{32}. Let $f : I \times S \to S$ be the integer multiplication modulo 2^{32}. This is a quasigroup (actually the multiplicative group of the ring $\mathbb{Z}/2^{32}$).

We choose $o_F : S \to O_F = B$ as the function taking the 8 MSBs from the 32-bit integer.

Example 3. If we correspond a 32-bit integer x to a 33-bit odd integer $2x + 1$ modulo 2^{33}, then the multiplication formula

$$(2x + 1) \times (2y + 1) = 2(2xy + x + y) + 1$$

gives a quasigroup structure

$$\tilde{\times} : (x, y) \mapsto x \tilde{\times} y := 2xy + x + y \mod 2^{32}$$

on the set of 32-bit integers. We can consider the corresponding multiplicative filter with $I = S$ being the set of 32-bit integers.

Modern CPUs often have a fast integer multiplication for 32-bit integers. We shall discuss mathematical property of such filters in §2.3.

Example 4. (CryptMT1: MT with multiplicative filter)
We choose a LFSR described in (1) as the mother generator A_M, with $O_M = {\mathbb{F}_2}^w$. We can choose its parameters so that the period is a large Mersenne prime $Q = 2^p - 1$ (e.g. $p = 19937$ as in the case of MT19937 [10]). By identifying O_M as the set of w-bit integers, we can use the multiplicative filter A_F described in Example 3. We call this generator as *MT19937 with multiplicative filter*. The output function $o_F : S_F \to O_F = {\mathbb{F}_2}^8$ is extracting 8 MSBs. This generator is called CryptMT Version 1 (CryptMT1) [11].

2.1 *k*-Dimensional Distribution

Let k be an integer, and let A be an automaton without input as in Definition 1. We define its *k-tuple output function* $o^{(k)}$ by

$$o^{(k)} : S \to O^k \quad s \mapsto (o(s), o(f(s)), o(f^2(s)), \ldots, o(f^{k-1}(s))) \tag{2}$$

(i.e. $o^{(k)}$ maps the state to the next k outputs). Consider the multi-set of the possible output k-tuples for all states:

$$O^{(k)} := \{o^{(k)}(s) \mid s \in S\}.$$

This is the image of S by $o^{(k)}$ counted with multiplicities.

Definition 6. *The output of the automaton A is said to be k-dimensionally equidistributed if the multiplicity of each element in $O^{(k)}$ is same.*

This type of criteria is commonly used for PRNG for MC: MT19937 as a 32-bit integer generator has this property with $k = 623$. This criterion is equivalent to the uniformness of the function $o^{(k)}$ defined below.

Definition 7. *A mapping $g : X \to Y$ is* uniform *if the cardinality of $g^{-1}(y)$ is independent of $y \in Y$. A bijection is uniform, and the composition of uniform mappings is uniform.*

A filter with memory is uniform *if its output function is uniform.*

Example 4 is uniform. The next proposition shows that a uniform quasigroup filter increases the dimension of equidistribution by 1. A proof is in Appendix B.1.

Proposition 1. *We keep the set-up of Definition 4. Assume that A_F is a uniform quasigroup filter. Suppose that the output of A_M is k-dimensionally equidistributed. Then, the combined generator C is $(k+1)$-dimensionally equidistributed.*

Corollary 1. *CryptMT1 explained in Example 4 is 624-dimensionally equidistributed.*

We mean by a *simple distinguishing attack of order N* to choose a real function F with N variables and to detect the deviation of the distribution of the values of F applied to the consecutive N-outputs. If N does not exceed the dimension of the equidistribution, then one can observe no deviation from the true randomness, under the assumption of uniform choice of the initial state.

By this reason, it seems very difficult to apply a correlation attack or a distinguishing attack to such generators. For example, to observe some deviation of MT19937 with multiplicative filter in Example 4, one needs to observe the correlation of outputs with the lag more than 624. Because of the high nonlinearity of the multiplicative filter discussed below, we guess that this would be infeasible.

2.2 A Theorem on the Period

Theorem 1. *Consider a combined generator C as in Definition 4. Let $s_{M,0}$ be the initial state of the mother generator A_M, and assume that its state transition is purely periodic with period $P = Qq$ for a prime Q and an integer q. Let $S^o \subset S_M$ be the orbit of the state transition. Let k be an integer. Assume that the k-tuple output function of the mother generator $o_M^{(k)} : S^o \to O_M^k$ as defined in (2) is surjective when restricted to S^o. Suppose that A_F is a quasigroup filter as in Definition 1.*

Let r be the ratio of the occupation of the maximum inverse image of one element by $o_F : S_F \to O_F$ in S_F, namely

$$r = \max_{b \in O_F} \{\#(o_F^{-1}(b))\}/\#(S_F).$$

If

$$r^{-(k+1)} > q \cdot (\#(S_F))^2,$$

then the period of the output sequence of C is a nonzero multiple of Q.

A proof is given in Appendix B.2.

Example 5. For MT19937 with multiplicative filter, this theorem shows that any bit in the output sequence has a period being a multiple of the prime $2^{19937}-1$, as follows.

We have $Q = 2^{19937}-1$ and $q = 1$. If $o_F : S_F \to O_F = {\mathbb{F}_2}^m$ is extracting some m bits from the 32-bit integers, then $r = 2^{w-m}/2^w = 2^{-m}$. The inequality condition in the theorem is now

$$2^{m(k+1)} > 2^{2w},$$

and hence if this holds, then the m-bit output sequence has a period which is a multiple of Q.

In the case of MT19937 and the multiplicative filter, since $k = 623$ and $w = 32$, the above inequality holds for any $m \geq 1$, hence any bit of the output has a period at least $2^{19937}-1$.

2.3 A Proposition on the Algebraic Degree of Integer Products

Definition 8. *Let us define a boolean function $m_{s,N}$ of $(s-1)N$ variables, as follows. Consider N of s-bit integer variables $x_1, \ldots, x_N$. Let*

$$c_{s-1,i}c_{s-2,i}\cdots c_{0,i}$$

be the 2-adic representation of x_i, hence $c_{j,i} = 0, 1$. We fix $c_{0,i} = 1$ for all $i = 1, \ldots, N$, i.e. assuming x_i odd. The boolean function $m_{s,N}$ has variables $c_{j,i}$ $(j = 1, 2, \ldots, s-1, i = 1, 2, \ldots, N)$, and its value is defined as the s-th digit (from the LSB) of the 2-adic expansion of the product $x_1x_2\cdots x_N$ as an integer.

Proposition 2. *Assume that $N, s \geq 2$. The algebraic degree of $m_{s,N}$ is bounded from below by*

$$\min\{2^{s-2}, 2^{\lfloor \log_2 N \rfloor}\}.$$

A proof is given in Appendix B.3. This proposition gives the algebraic degree of the multiplicative filter, with respect to the inputs $x_1, \ldots, x_N$.

This proposition implies that we should use MSBs of the multiplicative filter. On the other hand, using 8 MSBs among 32-bit integers as in Example 2 seems to have enough high algebraic degree. We check this using a toy model in Appendix A.

3 A Fast Initialization of a Large State Space

Consider LFSR in (1) as a mother generator. Its state space is an array of w-bit integers with size n. We need to give initial values to such a large array in the initialization. If one wants to encrypt a much shorter message than n, then this is not efficient. A possible solution is to use a PRNG with relatively small state space (called *the booter*) which can be quickly initialized, and use it to generate the initial array $x_0, x_1, \ldots, x_{n-1}$, and at the same time, its outputs are passed

to the filter for key-stream generation. If the message length is smaller than n, then the mother generator is never used: the outputs of the booter are used as the output of the mother generator. If the message length exceeds n, then the first n outputs of the booter are used as the outputs of the mother generator, and at the same time for filling up the state space of the mother generator. After the state space is filled up, the mother generator starts to work. See Appendix C for more detail.

The first outputs come from the booter. One may argue why not using the booter forever, without using the mother generator. The answer is that we do not need to care about the attacks to the booter based on a long output stream.

4 A Concrete Example Using 128-Bit Instructions

Recent CPUs often have Single Instruction Multiple Data (SIMD) instructions. These instructions treat a quadruple of 32-bit integers at one time. We propose a LFSR and a uniform quasigroup filter, based on 128-bit instructions, named CryptMT Version 3 (CryptMT3) in the rest of this paper. CryptMT3 is one of the phase 3 candidates in eSTREAM stream cipher competition [13]. We shall describe the generation algorithm below.

4.1 SIMD Fast MT

In the LFSR (1), we assume that each x_i is a 128-bit integer or equivalently a vector in ${\mathbb{F}_2}^{128}$. We choose the following recursion: $n = 156$ and

$$x_{156+j} := (x_{156+j-1} \,\&\, \texttt{ffdfafdf f5dabfff ffdbffff ef7bffff}) \oplus (x_{108+j} >>_{64} 3) \oplus x_{108+j}[2][0][3][1] \oplus (x_j[0][3][2][1]). \tag{3}$$

Here, & denotes the bit-wise-and operation, and the hexadecimal integer is a constant 128-bit integer for the bit-mask. The notation $\oplus$ is bitwise exor. The notation

$$(x_{108+j} >>_{64} 3)$$

means that x_{108+j} is considered as two 64-bit integers, and each of them is shifted to the right by 3 bits. The notation $x_{108+j}[2][0][3][1]$ is a permutation of four 32-bit integers. The 128-bit integer x_{108+j} is considered as a quadruple of (0th, 1st, 2nd, 3rd) 32-bit integers, and then they are permuted by $2 \to 0, 1 \to 0, 2 \to 3, 3 \to 1$. The next notation $x_j[0][3][2][1]$ is a similar permutation. These instructions are available both in SSE2 SIMD instructions for Intel processors and in AltiVec SIMD instructions in PowerPC. We call this generator SIMD Fast MT (SFMT) (This is a variant of [14]). A description is in Figure 2. We proved its 155-dimensional equidistribution property. We proved that, if the third component $x_0[3]$ of x_0 is `0x4d734e48`, then the period of the generated sequence of the SFMT is a multiple of the Mersenne prime $2^{19937} - 1$. Note that since 19937 is a prime, there is no intermediate field of $\mathbb{F}_{2^{19937}}$. This is in contrast to SNOW1.0, where the existence of the intermediate field $\mathbb{F}_{2^{32}}$ introduces some weakness (see [6]).

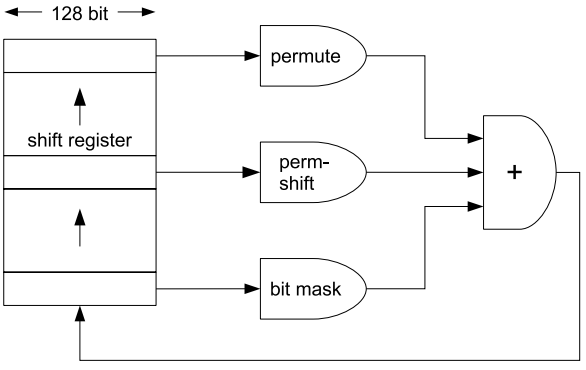

Fig. 2. The mother generator of CryptMT3: SIMD Fast Mersenne Twister
permute: $\mathbf{y} \mapsto \mathbf{y}[0][3][2][1]$.
perm-shift: $\mathbf{y} \mapsto \mathbf{y}[2][0][3][1] \oplus (\mathbf{y} >>_{64} 3)$.
bit-mask: `ffdfafdf f5dabfff ffdbffff ef7bffff`

4.2 A Modified Multiplicative Filter

Our filter A_F has $I_F = S_F$ being the set of 128-bit integers, and O_F being the set of 64-bit integers, as described below.

For given 128-bit integers $y \in I_F$ and $x \in S_F$, we define

$$f_F(y, x) := (y \oplus (y[0][3][2][1] >>_{32} 1))\tilde{\times}_{32}x. \tag{4}$$

Here, the notation "$>>_{32} 1$" means to consider a 128-bit integer as a quadruple of 32-bit integers, and then shift each of them to the right by 1 bit. The binary operator $x\tilde{\times}_{32}y$ means that 32-bit wise binary operation $\tilde{\times}$ (see Example 3) is applied for each 32-bit components, namely, i-th 32-bit integer of $x\tilde{\times}_{32}y$ is $x[i]\tilde{\times}y[i]$ $(i = 0, 1, 2, 3)$.

The operation applied to y is an invertible linear transformation, hence is bijective. Since $\tilde{\times}$ is bi-bijective, so is f_F. The purpose to introduce the permutation-shift is to mix the information among four 32-bit memories in the filter, and to send the information of the upper bits to the lower bits. This supplements the multiplication, which lacks this direction of transfer of the information.

The output function is

$$o_F(y) := \mathrm{LSB}_{32}^{16}(y \oplus (y >>_{32} 16)). \tag{5}$$

This means that y is considered as a quadruple of 32-bit integers, and for each of them, we take the exor of the MSB 16 bits and LSB 16 bits. Thus we obtain four 16-bit integers, which is the output 64-bit integer (see Figure 3). To obtain 8-bit integers, we dissect it into 8 pieces.

CryptMT3 is the combination of the SIMD Fast MT (§4.1) and this filter. Initialization by the booter is explained in Appendix C.

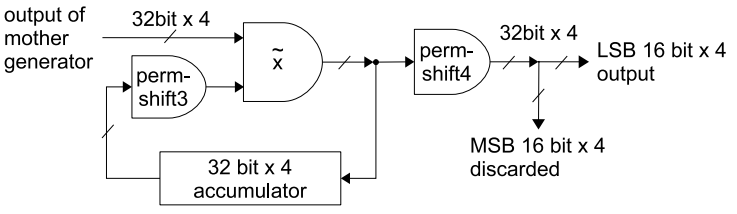

Fig. 3. Filter of CryptMT3.
perm-shift3: $\mathbf{y} \mapsto \mathbf{y} \oplus (\mathbf{y}[0][3][2][1] >>_{32} 1)$.
perm-shift4: $\mathbf{y} \mapsto \mathbf{y} \oplus (\mathbf{y} >>_{32} 16)$.
$\tilde{\times}$: multiplication of 33-bit odd integers.

Table 1. Summary from eSTREAM benchmark [2]

Primitive	Core 2 Duo			AMD Athlon 64 X2			Motorola PowerPC G4		
	Stream	Key setup	IV setup	Stream	Key	IV	Stream	Key	IV
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

4.3 Speed Comparison

Comparison of the speed of generation for stream ciphers is a delicate problem: it depends on the platform, compilers, and so on. Here we compare the number of cycles consumed per byte, by CryptMT3, HC256, SOSEMANUK, Salsa20, Dragon (these are the five candidates in eSTREAM software cipher phase 3 permitting 256-bit Key), SNOW2.0 and AES (counter-mode), in three different CPUs: Intel Core 2 Duo, AMD-Athlon X2, and Motorola PowerPC G4, using eSTREAM timing-tool [7]. The data are listed in Table 1. Actually, they are copied from Bernstein's page [2]. The number of cycles in Key set-up and IV set-up are also listed.

CryptMT3 is the fastest in generation in Intel Core 2 Duo CPU, reflecting the efficiency of SIMD operations in this newer CPU. CryptMT3 is slower in Motorola PowerPC. This is because AltiVec (SIMD of PowerPC) lacks 32-bit integer multiplication (so we used non-SIMD multiplication instead).

5 Conclusions

We proposed combination of a LFSR and a uniform quasigroup filter as a stream cipher in software. As a concrete example, we implemented CryptMT3 generator. CryptMT3 is as fast as SNOW2.0 and faster than AES counter-mode for recent CPUs. CryptMT3 satisfies the conditions of Theorem 1 and Proposition 1, and it can be proved to have the astronomical period $\geq 2^{19937} - 1$ and the

156-dimensional equidistribution property as a 64-bit integer generator (and hence 1241-dimensional equidistribution property as a 8-bit integer generator).

CryptMT3 uses integer multiplication instead of an S-box. This is an advantage over generators with large look-up tables for fast software implementation of the S-box, such as SNOW or AES, where cache-timing attacks might be applied [1].

A toy model of CryptMT3 shows high algebraic degrees and nonlinearity for the multiplicative filter, which supports its effectiveness.

References

1. Bernstein, D.J.: Cache-timing attack on AES `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`
2. Bernstein, D.J.: Software timings. `http://cr.yp.to/streamciphers/timings.html`
3. Courtois, N.: Cryptanalysis of Sfinks, `http://eprint.iacr.org/2005/243`
4. Courtois, N.: Fast algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)
5. Ekdahl, P., Johansson, T.: SNOW-a new stream cipher. In: Proceedings of First Open NESSIE Workshop, KU-Leuven (2000)
6. Ekdahl, P., Johansson, T.A.: A New Version of the Stream Cipher SNOW. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 47–61. Springer, Heidelberg (2003)
7. eSTREAM – The ECRYPT Stream Cipher Project – Phase 3. `http://www.ecrypt.eu.org/stream/index.html`
8. Golomb, S.: Shift Register Sequences. Aegean Park Press (1982)
9. Knuth, D.E.: The Art of Computer Programming. In: Seminumerical Algorithms, 3rd edn., vol. 2, Addison-Wesley, Reading, Mass. (1997)
10. Matsumoto, M., Nishimura, T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation 8, 3–30 (1998)
11. Matsumoto, M., Saito, M., Nishimura, T., Hagita, M.: Cryptanalysis of CryptMT: Effect of Huge Prime Period and Multiplicative Filter, `http://www.ecrypt.eu.org/stream/cryptmtfubuki.html`
12. Matsumoto, M., Saito, M., Nishimura, T., Hagita, M.: CryptMT Version 2.0: a large state generator with faster initialization, `http://www.ecrypt.eu.org/stream/cryptmtfubuki.html`
13. Matsumoto, M., Saito, M., Nishimura, T., Hagita, M.: CryptMT Stream Cipher Version 3. eSTREAM stream cipher proposals (submitted), `http://www.ecrypt.eu.org/stream/cryptmtp3.html`
14. Saito, M., Matsumoto, M.: SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. In: The proceedings of MCQMC (2006) (to appear)

Appendix

A Simulation by Toy Models

Since the filter has a memory, it is not clear how to define the algebraic degree or non-linearity of the filter. Instead, if we consider all bits in the initial state as

variables, then each bit of the outputs is a boolean function of these variables, and algebraic degree and non-linearity are defined.

However, it seems difficult to compute them explicitly for CryptMT3, because of the size. So we made a toy model and obtained experimental results. Its mother generator is a linear generator with 16-bit internal state, and generates a 16-bit integer sequence defined by

$$\mathbf{x}_{j+1} := (\mathbf{x}_j >> 1) \oplus ((\mathbf{x}_j \& 1) \cdot \mathbf{a}),$$

where $>> 1$ denotes the one-bit shift-right, $(\mathbf{x}_j \& 1)$ denotes the LSB of $\mathbf{x}_j$, $\mathbf{a} = 1010001001111000$ is a constant 16-bit integer, and $(\mathbf{x}_j \& 1) \cdot \mathbf{a}$ denotes the product of the scalar $(\mathbf{x}_j \& 1) \in \mathbb{F}_2$ and the vector $\mathbf{a}$.

Then it is filtered by

$$y_{j+1} = (\mathbf{x}_j | 1) \times y_j \mod 2^{16},$$

where $(\mathbf{x}_j | 1)$ denotes $\mathbf{x}_j$ with LSB set to 1. We put $y_0 = 1$, and compute the algebraic degree of each of the 16 bits in the outputs $y_1 \sim y_{16}$, each regarded as a polynomial function with 16 variables being the bits in $\mathbf{x}_0$. The result is listed in Table 2. The lower six bits of the table clearly show the pattern $0, 1, 1, 2, 4, 8$, which suggests that the lower bound 2^{s-2} for $s \geq 2$ given in Proposition 2 would be tight, when the iterations are many enough. On the other hand, eighth bit and higher are "saturated" to the upper bound 16, after 12 generations.

We expect that the algebraic degrees for CryptMT3 would behave even better, since its filter is modified. So, if we consider each bit of the internal state of CryptMT3 as a variable, then the algebraic degree of the bits in the outputs will be near to 19937, after some steps of generations.

Also, we computed the non-linearity of the MSB of each y_i $(i = 1, 2, \ldots, 8)$ of this toy model. The result is listed in Table 3, and each value is near to 2^{16-1}. This suggests that there would be no good linear approximation of CryptMT3.

Table 2. Table of the algebraic degrees of output bits of a toy model

y_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
y_2	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
y_3	15	15	14	13	12	11	10	9	8	6	4	3	2	1	0
y_4	15	16	15	14	13	12	11	10	9	7	5	4	2	1	0
y_5	16	16	15	15	14	13	12	11	10	7	5	4	2	1	0
y_6	16	16	15	15	15	14	13	11	10	9	7	4	2	1	0
y_7	16	15	16	16	15	15	14	13	12	9	7	4	2	1	0
y_8	15	15	15	16	16	15	15	14	13	10	8	4	2	1	0
y_9	16	15	16	15	15	16	15	15	13	10	8	4	2	1	0
y_{10}	15	16	16	16	16	16	15	15	14	12	8	4	2	1	0
y_{11}	15	16	16	15	15	15	16	15	15	12	8	4	2	1	0
y_{12}	15	16	16	16	16	15	16	16	15	13	8	4	2	1	0
y_{13}	16	15	15	15	15	15	16	15	16	13	8	4	2	1	0
y_{14}	15	15	16	15	15	16	16	15	16	15	8	4	2	1	0
y_{15}	15	16	16	16	15	16	16	16	15	14	8	4	2	1	0
y_{16}	16	15	16	15	15	15	15	15	16	14	8	4	2	1	0

Table 3. The non-linearity of the MSB of each output of a toy model

output	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
nonlinearity	0	32112	32204	32238	32201	32211	32208	32170	32235

B Proof of Theorems and Propositions

B.1 Proof of Proposition 1

Proof. Consider the k-tuple output function of the mother generator $o_M^{(k)} : S_M \to O_M^k$ as in (2). Then, the k-dimensional equidistribution property is equivalent to the uniformness of $o_M^{(k)}$. The $(k+1)$-tuple output function $o_C^{(k+1)}$ of the combined generator C is the composite

$$o_C^{(k+1)} : S_M \times S_F \xrightarrow{o_M^{(k)} \times \mathrm{id}_{S_F}} O_M^k \times S_F \xrightarrow{\mu} S_F^{k+1} \xrightarrow{o_F^{k+1}} O_F^{k+1},$$

where the second map μ is given by

$$\mu : ((x_k, x_{k-1}, \ldots, x_1), y_1) \mapsto (y_{k+1}, y_k, \ldots, y_1)$$

where y_i's are inductively defined by $y_{i+1} := f_F(x_i, y_i) \quad (i = 1, 2, \ldots, k)$. The last map o_F^{k+1} is the direct product of $k+1$ copies of o_F. The quasigroup property of f_F implies the bijectivity of μ. The last map is uniform. Since the composition of uniform mappings is uniform, we obtain the proof.

B.2 Proof of Theorem 1

Proof. We may replace S_M with the orbit starting from s_0. Then, replace S_M with its quotient set where two states are identified if the output sequences from them are identical. Thus, we may assume $\#(S_M) = P$, where P is the period of the output sequence of A_M. In this proof, we do not consider multi-sets. Consider the k-tuple output function $o_C^{(k+1)}$ as in the proof of Proposition 1. Since $o_M^{(k)}$ is surjective and μ is bijective (by the quasigroup property), the image $I \subset O_M^{n+1}$ of $S_M \times \{y_0\}$ by $\mu \circ (o_M^{(k)} \times \mathrm{id}_Y)$ has the cardinality $\#(O_M)^k$. By the assumption of the pure periodicity of x_i and the bijectivity of f_F, the output sequence $o_F(y_i)$ $(i = 0, 1, 2, \ldots)$ is purely periodic. Let p be its period. Then, $o_F^{k+1}(I) \subset O_F^{k+1}$ can have at most p elements. Thus, by the assumption on o_F and the definition of r,

$$\#(I) \le p(r\#(S_F))^{k+1}.$$

Since $\#(O_M)^k = \#(I)$ and $\#(O_M) = \#(S_F)$, we have an inequality

$$r^{-(k+1)} \le p\#(S_F).$$

The period P' of the state transition of C is a multiple of $P = Qq$. Since the state size of C is $P \times \#(S_F)$, $P' = Qm$ holds for some $m \le q\#(S_F)$. Consequently, p

is a divisor of Qm. If p is not a multiple of Q, then p divides m and $p \leq q\#(S_F)$. Thus we have

$$r^{-(k+1)} \leq q\#(S_F)^2,$$

contradicting to the assumption.

B.3 Proof of Proposition 2

Let $h(c_1, c_2, \ldots, c_n)$ be a boolean function as in Definition 2, and $h = \sum_{T \subset \{1,2,..,n\}} a_T c_T$ be its algebraic normal form.

The following lemma is well known.

Lemma 1. *It holds that* $a_T = \sum_{U \subset T} h(U)$, *where* $h(U) := h(c_1, \ldots, c_n)$ *with* $c_i = 0, 1$ *according to* $i \notin U$, $\in U$, *respectively.*

Proof of Proposition

Proof. For $s = 2$, the claim is easy to check. We assume $s \geq 3$.
Case 1. $s - 2 \leq \log_2 N$. In this case, it suffices to prove that the algebraic degree is at least 2^{s-2}. Take a subset T of size 2^{s-2} from $\{1, 2, \ldots, N\}$, say $T = \{1, 2, \ldots, 2^{s-2}\}$. Then, we choose $c_{1,1}, c_{1,2}, \ldots, c_{1,2^{s-2}}$ as the $\#T$ variables "activated" in Lemma 1, and consequently, the coefficient of $c_{1,1}c_{1,2}\cdots c_{1,2^{s-2}}$ in the algebraic normal form of $m_{s,N}$ is given by the sum in $\mathbb{F}_2$:

$$a_T := \sum_{U \subset T} (s\text{-th bit of } x_1 \cdots x_n, \text{ where } c_{j,i} = 1 \text{ if and only if } j = 1 \text{ and } i \in U).$$

Note that $c_{0,i} = 1$. It suffices to prove $a_T = 1$. Now, each term in the right summation is the s-th bit of the integer $3^{\#U}$, so the right hand side equals to

$$\sum_{m=0}^{2^{s-2}} \left[\binom{2^{s-2}}{m} \times \text{ the } s\text{-th bit of } 3^m \right].$$

However, the well-known formula

$$(x+y)^{2^{s-2}} \equiv x^{2^{s-2}} + y^{2^{s-2}} \mod 2$$

implies that the binary coefficients are even except for the both end, so the summation is equal to the s-th bit of $3^{2^{s-2}}$.

A well-known lemma says that if $x \equiv 1 \mod 2^i$ and $x \not\equiv 1 \mod 2^{i+1}$ for $i \geq 2$, then $x^2 \equiv 1 \mod 2^{i+1}$ and $x^2 \not\equiv 1 \mod 2^{i+2}$. By applying this lemma inductively, we know that

$$3^{2^{s-2}} = (1+8)^{2^{s-3}} \equiv 1 \mod 2^s, \quad \not\equiv 1 \mod 2^{s+1}.$$

This means that s-th bit of $3^{2^{s-2}}$ is 1, and the proposition is proved.

Case 2. $s - 2 > \lfloor \log_2(N) \rfloor$. In this case, we put $t := \lfloor \log_2(N) \rfloor + 2$, and hence $s > t$ and $2^{t-2} \leq N$. We apply the above arguments for $T = \{1, 2, \ldots, 2^{t-2}\}$, but this time instead of $c_{1,i}$, we activate

$$\{c_{s-t+2,i} \mid i \in T\}.$$

The same argument as above reduces the non-vanishing of the coefficient of the term $c_{s-t+2,1} \cdots c_{s-t+2,2^{t-2}}$ to the non-vanishing of

$$\sum_{m=0}^{2^{t-2}} \left[\binom{2^{t-2}}{m} \times \text{the } s\text{-th bit of } (1 + 2^{s-t+2})^m \right].$$

Again, only the both ends $m = 0$ and $m = 2^{t-2}$ can survive, and the above summation is the s-th bit of $(1 + 2^{s-t+2})^{t-2}$. Since $s - t + 2 \geq 2$, the lemma mentioned above implies that

$$(1 + 2^{s-t+2})^{2^{t-2}} \equiv 1 \bmod 2^s, \quad \not\equiv 1 \bmod 2^{s+1},$$

which implies that its s-th bit is 1.

C The Key, IV, and the Booter

The design of the booter (see §3) goes independently of the key-stream generator. However, as the referees pointed out, we need to specify one to have a complete description of the generator. Thus, we here include the booter of CryptMT3 for self-containedness. The booter is described in Figure 4.

We choose an integer H later in §C.1. The state space of the booter is a shift register consisting of H 128-bit integers. We choose an initial state $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{H-1}$ and the initial value $\mathbf{a}_0$ of the accumulator (a 128-bit memory) as described in the next section. Then, the state transition is given by the recursion

$$\begin{aligned} \mathbf{a}_j &:= (\mathbf{a}_{j-1} \mathbin{\tilde{\times}}_{32} \texttt{perm-shift2}(\mathbf{x}_{H+j-1})) \\ \mathbf{x}_{H+j} &:= \texttt{perm-shift1}(\mathbf{x}_j +_{32} \mathbf{x}_{H+j-2}) -_{32} \mathbf{a}_j, \end{aligned}$$

where

$$\begin{aligned} \texttt{perm-shift1}(\mathbf{x}) &:= (\mathbf{x}[2][1][0][3]) \oplus (\mathbf{x} >>_{32} 13) \\ \texttt{perm-shift2}(\mathbf{x}) &:= (\mathbf{x}[1][0][2][3]) \oplus (\mathbf{x} >>_{32} 11). \end{aligned}$$

The notation $+_{32}$ ($-_{32}$) denotes the addition (subtraction, respectively) modulo 2^{32} for each of the four 32-bit integers in the 128-bit integers. The output of the j-th step is $\mathbf{x}_j +_{32} \mathbf{x}_{H+j-2}$.

As described in Figure 4, the booter consists of an automaton with three inputs and two outputs of 128-bit integers, together with a shift register. In the implementation, the shift register is taken in an array of 128-bit integers with the length $2H + 2 + n$, where $n = 156$ is the size of the state array of SFMT. This redundancy of the length is for the idling, as explained below.

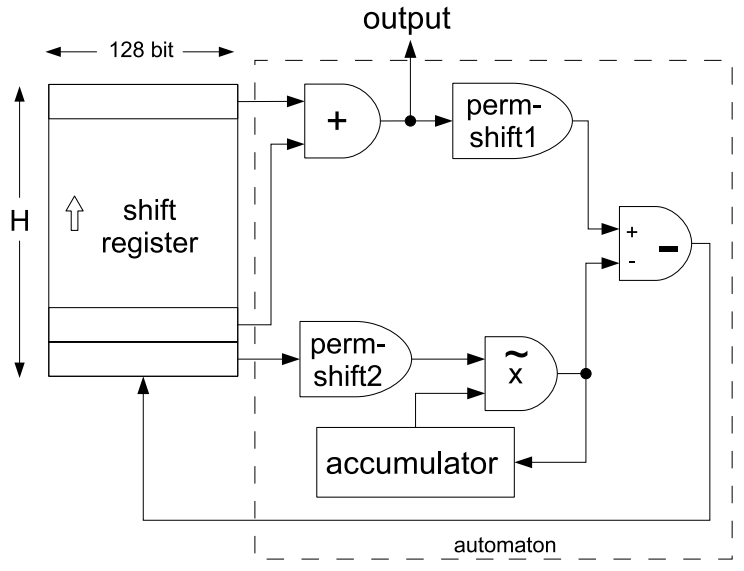

Fig. 4. Booter of CryptMT3.
`perm-shift1`: $\mathbf{x} \mapsto (\mathbf{x}[2][1][0][3]) \oplus (\mathbf{x} >>_{32} 13)$.
`perm-shift2`: $\mathbf{x} \mapsto (\mathbf{x}[1][0][2][3]) \oplus (\mathbf{x} >>_{32} 11)$.
$\tilde{\times}$: multiplication of (a quadruple of) 33-bit odd integers.
$+$, $-$: addition, subtraction of four 32-bit integers modulo 2^{32}.

C.1 Key and IV Set-Up

We assume that both the IV and the Key are given as arrays of 128-bit integers. The size of each array is chosen by user, from 1 to 16. Thus, the Key-size is chosen from 128 bits to 2048 bits, as well as the IV-size. We claim the security level that is the same with the minimum of the Key-size and the IV-size.

We concatenate the IV and the Key to a single array, and then it is copied twice to an array, as described in Figure 5. To break the symmetry, we add a constant 128-bit integer $(846264, 979323, 265358, 314159)$ (denoting four 32-bit integers in a decimal notation, coming from π) to the bottom row of the second copy of the key (add means $+_{32}$). Now, the size H of the shift register in the booter is set to be $2 \times$ (IV-size + Key-size (in bits))/128, namely, the twice of the number of 128-bit integers contained in the IV and the Key. For example, if the IV-size and the Key-size are both 128 bits, then $H = 2 \times (1 + 1) = 4$. The automaton in the booter described in Figure 4 is equipped on this array, as shown in Figure 5. The accumulator of the booter-automaton is set to

$$(\text{the top row of the key array}) \mid (1, 1, 1, 1),$$

that is, the top row is copied to the accumulator and then the LSB of each of the 32-bit integers in the accumulator is set to 1.

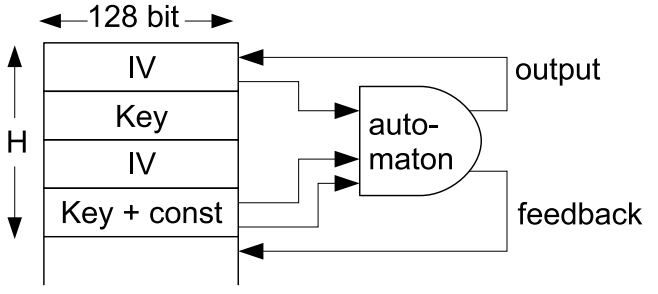

Fig. 5. Beginning of Key and IV set-up. The IV-array and Key-array are concatenated and copied to an array twice. Then, a constant is added to the bottom of the second copy of the key to break a possible symmetry. The automaton is described in Figure 4.

At the first generation, the automaton reads three 128-bit integers from the array, and write the output 128-bit integer at the top of the array. The feedback to the shift register is written into the $(H+1)$-st entry of the array. For the next generation, we shift the automaton downwards by one, and proceed in the same way.

For idling, we iterate this for $H+2$ times. Then, the latest modified row in the array is the $(2H+2)$-nd row, and it is copied to the 128-bit memory in the filter of CryptMT3. We discard the top $H+2$ entries of the array. This completes the Key and IV set-up. Figure 6 shows the state after the set-up.

After the set-up, the booter produces 128-bit integer outputs, at most n times. Let L be the number of bits in the message. If $L \leq n \times 64$, then we do not need the

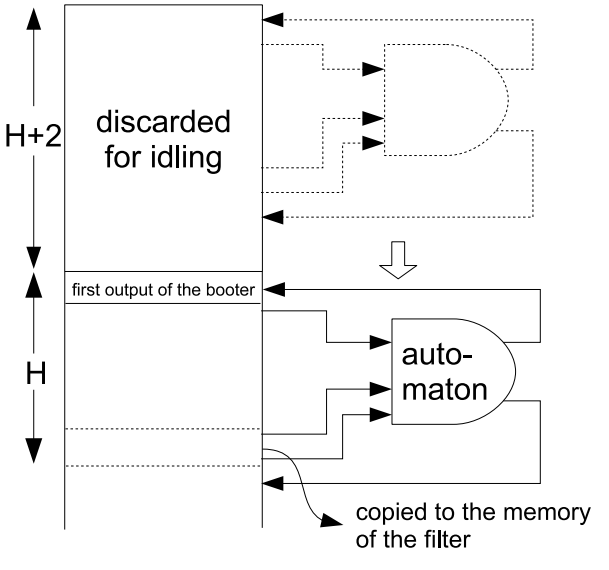

Fig. 6. After the Key and IV set-up

mother generator. We generate the necessary number of 128-bit integers by the booter, and pass them to the filter to obtain the required outputs. If $L \geq n \times 64$, then, we generate n 128-bit integers by the booter, and pass them to the filter to obtain n 64-bit integers, which are used as the first outputs. At the same time, these n 128-bit integers are recorded in the array, and they are passed to SFMT as the initial state.

To eliminate the possibility of shorter period than $2^{19937} - 1$, we set the 32 MSBs of the first row of the state array of SFMT to the magic number `0x4d734e48` in the hexadecimal representation, as explained in §4.1. That is, we start the recursion (3) of SFMT with $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$ being the array of length n generated by the booter (with 32 bits replaced with a magic constant), and then SFMT produces $\mathbf{x}_n, \mathbf{x}_{n+1}, \ldots$. Since $\mathbf{x}_n$ might be easier to guess because of the constant part in the initial state, we skip $\mathbf{x}_n$ and pass the 128-bit integers $\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \ldots$ to the filter.