High Efficiency Feedback Shift Register: $\sigma-$LFSR*

Guang Zeng[1], Wenbao Han[2], and Kaicheng He[3]

[1] Department of Applied Mathematics,
Institute of Information and Science ,
Zhengzhou 450002, PR.CHINA,
sunshine_zeng@sina.com
[2] wb.han@263.net
[3] hekaicheng_jxy@163.com

Abstract. We introduce a new kind of word-oriented linear feedback shift register called $\sigma-$LFSR which is constructed with the instructions of the modern processor and have fast software implementation. We offer an algorithm to search for good primitive $\sigma-$LFSR. In particular, we give two examples HHZ-1 and HHZ-2 and compare their efficiency and security with those of the LFSRs appearing in stream ciphers such as SNOW, SOBER and Turing. Our results show that replacing the LFSRs in SNOW, SOBER and Turing with HHZ-1 will improve security and the efficiency of fast software implementation.

Keywords. Linear Feedback Shift Register, Finite Field, Stream Cipher

1 Introduction

Linear feedback shift registers (LFSRs) over finite fields are widely exploited and play a important role in cryptography and coding theory, see Golomb [1], Lidl and Niederreiter [2] etc. Most LFSRs used in traditional stream cipher are based on binary field $\mathbb{F}_2$, which have good cryptographic properties but produce only one bit per step. It is well-known that hardware implementations of traditional binary LFSR are simple and efficient, but their software implementations are inefficient for modern processors with word operations.

Generally, there are two principles to evaluate FSR sequences: 1). security; 2). efficiency and resource consumption. These two principles have same importance, in other words, if a FSR has extremely excellent cryptographic properties but its implementation efficiency is low and resource consumption is large, their application value is limited. In fact, modern computer processors provide many fundamental word operations: *a*). logic operations such as Xor, And, Or, complementary operation, left shift, right shift, cycle shift etc; *b*). arithmetic operations such as addition, substraction, multiplication, division etc. So it is interesting to

* This work is supported by NSF of China with contract No.19971096 and No.90104035

research on how to use the word operation above to design word-oriented feedback shift registers (FSRs) with good security, easy hardware and fast software implementations.

In FSE of 1994, Preneel [3] set forth the following problem: **how to design fast and secure FSRs with the help of the word operations of modern processors and the techniques of parallelism**. In the stream ciphers such as SOBER [4, 5], SNOW [6, 7], and Turing [8], word-oriented primitive LFSRs over finite field were used, which were carefully chosen so that the Hamming weights of the generating primitive polynomials of the component sequences are large and have fast software implementation.

This paper is arranged as following. In section 2, we introduce the concept of $\sigma-$linear feedback shift register ($\sigma-$LFSR) based on logic operations on words, which is the generalization of TSR introduced by Tsaban and Vishne [9], and give basic properties of $\sigma-$LFSR. In Section 3, we discuss the 32-bit $\sigma-$LFRS and give an algorithm to search for the primitive $\sigma-$LFRS with few logic operations on words, as a result we give two examples: HHZ-1 and HHZ-2. In Section 4, we compare HHZ-1 and HHZ-2 with the LFSRs appeared in SNOW1.0, SNOW2.0, SOBERt-32 and Turing by the point of view from security and efficiency of software implementation.

2 Background of $\sigma-$LFSR

2.1 Definition of $\sigma-$LFSR

Let m be a positive integer, $\mathbb{F}_{2^m}$ the finite field with 2^m elements, α_0, α_1, $\cdots$, α_{m-1} a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Let $M_m(\mathbb{F}_2)$ denotes $m \times m$ matrix ring over $\mathbb{F}_2$ and $GL_m(\mathbb{F}_2) \subseteq M_m(\mathbb{F}_2)$ the general linear group.

Let $c \in \mathbb{F}_{2^m}$, then c can induce a linear transformation $\mathcal{C}$ on vector space $\mathbb{F}_{2^m}/\mathbb{F}_2 :\ \mathcal{C}(\alpha) = c\alpha,\ \forall\ \alpha \in \mathbb{F}_{2^m}$. For convenience, we will use the same symbol throughout the paper without distinguishing elements in finite field, induced linear transformations and corresponding matrixes under the basis α_0, α_1, $\cdots$, α_{m-1}. So we have $\mathbb{F}_{2^m} \subseteq M_m(\mathbb{F}_2)$.

Let $A \in M_m(F_2)$, $\alpha \in \mathbb{F}_{2^m}$, there exists vector $(a_0, a_1, \cdots, a_{m-1}) \in \mathbb{F}_2^m$ such that $\alpha = \sum_{i=0}^{m-1} a_i\alpha_i$. We define linear transformation $\mathcal{A}$ induced by A as

$$\mathcal{A}(\alpha) = \sum_{i=0}^{m-1} b_i\alpha_i,\ \textit{where}\ (b_0, b_1, \cdots, b_{m-1}) = (a_0, a_1, \cdots, a_{m-1}) \cdot A \in \mathbb{F}_2^m$$

Later, we denote $\mathcal{A}$ by A and $A(\alpha)$ is abbreviated to $A\alpha$ for $\alpha \in \mathbb{F}_{2^m}$.

Definition 1. *Let n be a positive integer, $C_0, C_1, \ldots, C_{n-1} \in M_m(\mathbb{F}_2)$. If the sequence $s^\infty = s_0, s_1, \cdots$ over $\mathbb{F}_{2^m}$ satisfies:*

$$s_{i+n} = -\big(C_0 s_i + C_1 s_{i+1} + \cdots + C_{n-1} s_{i+n-1}\big) \qquad \textit{for } i = 0, 1, \cdots, \tag{1}$$

we call (1) $\sigma-$linear feedback shift register ($\sigma-$LFSR) of order n, s^∞ the sequence generated by the $\sigma-$LFSR (1), matrix polynomial

$$f(x) = x^n + C_{n-1}x^{n-1} + \cdots + C_1 x + C_0 \in M_m(\mathbb{F}_2)[x]$$

$\sigma-$polynomial of $\sigma-$LFSR (1).

The following figure is the model of $\sigma-$LFSR

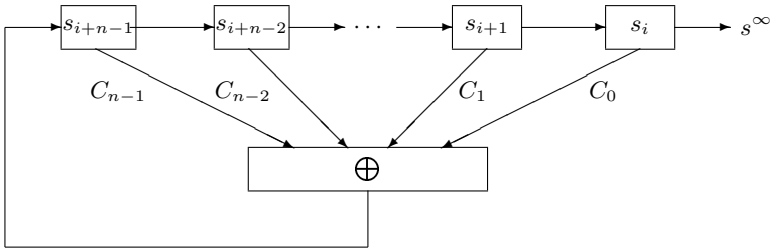


Fig. 1. $\sigma-$LFSR over $\mathbb{F}_{2^m}$

2.2 Basic Properties

It is easy to see that s^∞ generated by $\sigma-$LFSR (1) over $\mathbb{F}_{2^m}$ must be ultimately periodic sequence, that is, there exists integers $r > 0$ and the least nonnegative integer $n_0 \geq 0$ such that $s_{n+r} = s_n$ for all $n \geq n_0$, r is the so-called period and n_0 the preperiod of s^∞. If $n_0 = 0$, the sequence is periodic precisely. Similar to linear recurring sequences in finite fields, we have

Theorem 1. *[12] Let the sequence $s^\infty = s_0, s_1 \cdots$ be generated by $\sigma-$LFSR (1). Then s^∞ is periodic if and only if $C_0 \in M_m(\mathbb{F}_2)$ is invertible, that is $C_0 \in GL_m(\mathbb{F}_2)$.*

Theorem 1 is the generalization of traditional LFSR, same as the conclusion of traditional LFSR over finite fields if all the coefficients in $\sigma-$LFSR (1) are in $\mathbb{F}_{2^m}$. In research of stream cipher, it is familiar that using simple structure to construct pseudorandom sequences with good cryptographic properties is of vital significance. It is hopeful that $\sigma-$LFSR can serve those good sequences. Now we give the definition of primitive $\sigma-$LFSR as following.

Definition 2. *Let the sequence $s^\infty = s_0, s_1 \cdots$ be generated by $\sigma-$LFSR (1). If the period of s^∞ is $2^{mn} - 1$, we call $\sigma-$LFSR (1) primitive $\sigma-$LFSR and its $\sigma-$polynomial primitive $\sigma-$polynomial.*

We observe that the period of s^∞ with $n-$th $\sigma-$LFSR (1) is $\leq 2^{mn} - 1$, so $2^{mn} - 1$ is the possible maximal period, in fact it is.

Theorem 2. *[12] Let the sequence $s^\infty = s_0, s_1 \cdots$ be generated by $\sigma-$LFSR (1) with $\sigma-$polynomial $f(x) = x^n + C_{n-1}x^{n-1} + \cdots + C_1 x + C_0 \in M_m(\mathbb{F}_2)[x]$ where $C_0 \in GL_m(\mathbb{F}_2)$ and $C_l = (c_l^{ij})_{m\times m}$ for $l = 0, 1, \cdots, n-1$,*

$$F(x) = (f^{ij}(x))_{m\times m} \in M_m(\mathbb{F}_2[x])$$

be the corresponding polynomial matrix of $f(x)$ where

$$f^{ij}(x) = \delta^{ij}x^n + \sum_{l=0}^{n-1} c_l^{ij}x^l \in \mathbb{F}_2[x],\ \delta^{ij} = \begin{cases} 1, & i = j; \\ 0, & i \neq j. \end{cases}$$

Then $\sigma-$LFSR (1) is a primitive $\sigma-$LFSR if and only if the determinant $|F(x)|$ is a primitive polynomial of degree mn over $\mathbb{F}_2$.

Theorem 2 gives a condition to determine whether $\sigma-$LFSR is primitive or not. We expect to find such primitive $\sigma-$LFSR whose $\sigma-$polynomial has simple form and fast software implementation. For the number of primitive $\sigma-$LFSR, we did massive tests and find the following formula although we do not prove.

Conjecture 1. The number of all $n-$th order primitive $\sigma-$LFSR over $\mathbb{F}_{2^m}$ is

$$\Upsilon(m, n) = \frac{|GL_m(\mathbb{F}_2)|}{2^m - 1} \cdot \frac{\varphi(2^{mn} - 1)}{mn} \cdot 2^{m(m-1)(n-1)}$$

where $GL_m(\mathbb{F}_2)$ is the general linear group over $\mathbb{F}_2$ and $|GL_m(\mathbb{F}_2)| = \prod_{i=0}^{m-1}(2^m - 2^i)$.

By Conjecture 1, we conclude that the number of $n-$th order primitive $\sigma-$LFSR over $\mathbb{F}_{2^m}$ is much greater than $\frac{\varphi(2^{mn}-1)}{n}$, the number of $n-$th primitive LFSR over $\mathbb{F}_{2^m}$. This provides much broader space for choosing $\sigma-$LFSR with good cryptographic properties to satisfy different requirements.

Now we discuss the primitive problem on coordinate sequences of primitive $\sigma-$LFSR. Under the basis $\alpha_0, \alpha_2, \cdots, \alpha_{m-1}$ of $\mathbb{F}_{2^m}/\mathbb{F}_2$, we can rewrite the sequence s^∞ generated by $\sigma-$LFSR over $\mathbb{F}_{2^m}$ as follows:

$$s^\infty = s_0^\infty\alpha_0 + s_1^\infty\alpha_1 + \cdots + s_{m-1}^\infty\alpha_{m-1}.$$

We call binary sequence s_i^∞ the i-th coordinate sequence of s^∞ $(0 \leq i \leq m-1)$.

Definition 3. *Let $f(x) \in M_m(\mathbb{F}_2)[x]$. For $0 \leq i \leq m-1$, define*

$$\begin{aligned} G(f(x)) &= \{s^\infty \in \mathbb{F}_{2^m}^\infty | s^\infty \textit{ is generated by } \sigma-\textit{LFSR with } \sigma-\textit{polynomial } f(x)\} \\ G_i(f(x)) &= \{s_i^\infty \in \mathbb{F}_2^\infty | \textit{there exists a } s^\infty \in G(f(x)) \textit{ such that } s_i^\infty \textit{ is the } i\textit{-th} \\ &\qquad \textit{coordinate sequence of } s^\infty\} \end{aligned}$$

Now we discuss the minimal polynomial of the binary sequences in $G_i(f(x))$ when $f(x)$ is a primitive $\sigma-$polynomial.

Theorem 3. *[12] Let $f(x) \in M_m(\mathbb{F}_2)[x]$ be a primitive $\sigma-$polynomial of degree n, $F(x)$ the polynomial matrix of $f(x)$. Then all the nonzero binary sequences in $G_i(f(x))$ for $i = 0, 1, \cdots, m-1$ is primitive linear recurrence sequences over $\mathbb{F}_2$ with the minimal polynomial $p(x) = |F(x)|$.*

Tsaban and Vishne [9] introduce the concept of TSR, discuss its basic properties and give an algorithm to search for primitive TSRs. Dewar and Panario [10] further improve the algorithm. In fact, TSR is a special case of $\sigma-$LFSR as described in the following figure, where $(a_0, a_1, \cdots, a_{n-1}) \in \mathbb{F}_2^n,\ T \in GL_m(\mathbb{F}_2)$.

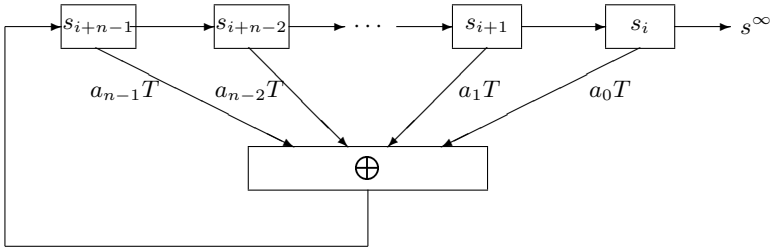


Fig. 2. Model of TSR

In fact, we have more general conclusion as following.

Corollary 1. *(cf.[9]) Let $T \in GL_m(\mathbb{F}_2)$, $q_T(x) = |xE + T| \in \mathbb{F}_2[x]$ be the characteristic polynomial of T, $f(x) = w_1(x)I + w_2(x)T \in M_m(\mathbb{F}_2)[x]$ where E is $m \times m$ identity matrix, $w_1(x), w_2(x) \in \mathbb{F}_2[x]$ such that $w_2(x) \neq 0$, $\deg w_1(x) = n, \deg w_2(x) < n$. Then $\sigma-$polynomial $f(x)$ is primitive $\Leftrightarrow w_2(x)^m q_T(\frac{w_1(x)}{w_2(x)})$ is a primitive polynomial over $\mathbb{F}_2$.*

Proof. Let $F(x)$ be the polynomial matrix of $f(x)$, we have

$$|F(x)| = |w_1(x)I + w_2(x)T| = w_2(x)^m |\frac{w_1(x)}{w_2(x)} I + T| = w_2(x)^m q_T(\frac{w_1(x)}{w_2(x)})$$

By Theorem 2, we prove the corollary. □

If $w_1(x) = x^n$ in Corollary 1, then we obtain the conclusion of Tsaban and Vishne[9]. Furthermore, Let $m = 32$, $T \in GL_{32}(\mathbb{F}_2)$ the corresponding matrix of linear transformation induced by a suitable primitive element of $\mathbb{F}_{2^{32}}$ under the basis $\alpha_0, \alpha_1, \cdots, \alpha_{31}$ of $\mathbb{F}_{2^{32}}$ over $\mathbb{F}_2$.

1. Let $n = 17$, $w_1(x) = x^{17} + x^{15} + x^4, w_2(x) = 1$, then the $\sigma-$LFSR with $\sigma-$polynomial $f(x) = w_1(x)I + w_2(x)T$ appears in SOBER-t32 and Turing;
2. Let $n = 16$, $w_1(x) = x^{16}, w_2(x) = x^9 + x^3 + 1$, then the $\sigma-$LFSR with $\sigma-$polynomial $f(x) = w_1(x)I + w_2(x)T$ is the part of SNOW1.0 where T is the matrix of a primitive element of $\mathbb{F}_{2^{32}}$ under the basis $\alpha_0, \alpha_1, \cdots, \alpha_{31}$.

In fact, the LFSR component of SNOW2.0 is also a simple $\sigma-$LFSR, but a little more complex than the examples above. In next section, we'll discuss $\sigma-$LFSRs over $\mathbb{F}_{2^{32}}$ to obtain simple primitive $\sigma-$LFSR with fast software implementation, lower resource consumption and stronger security.

3 Software Oriented 32 bit $\sigma-$LFSR

Traditional LFSR over $\mathbb{F}_{2^m}$ uses addition and multiplication over finite field $\mathbb{F}_{2^m}$, but multiplication reduces software implementation efficiency greatly. For the sake of seeking for $\sigma-$LFSR suitable for high speed implementation in software, we use some special linear transformations, namely the word operations provided by modern processor, such as, and operation, circular rotation operation, left(right) shift operation and left right shift combination operation.

For $\alpha \in \mathbb{F}_{2^m}$, let $\alpha = \sum_{i=0}^{m-1} a_i\alpha_i$, where $a_i \in \mathbb{F}_2$ for $i = 0, 1, \cdots, m-1$.

1. **And Operation $\wedge_\gamma$.** Let $\gamma \in \mathbb{F}_{2^m}$, $\gamma = \sum_{i=0}^{m-1} c_i\alpha_i$, where $c_i \in \mathbb{F}_2$ for $i = 0, 1, \cdots, m-1$. For $\alpha \in \mathbb{F}_{2^m}$ we define $\wedge_\gamma(\alpha) = \sum_{i=0}^{m-1} a_ic_i\alpha_i$. In fact, linear transformation $\wedge_\gamma$ over $\mathbb{F}_{2^m}/\mathbb{F}_2$ is "and" operation of coordinates of γ and α.

$$\wedge_\gamma = \begin{pmatrix} c_0 & 0 & \cdots & 0 \\ 0 & c_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{m-1} \end{pmatrix}_{m\times m} \qquad \sigma = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{m\times m} \tag{2}$$

2. **Circular Rotation Operation σ.** Defined in (2) as above, in fact, σ is right circular rotation, so left circular rotation is σ^{-1}. Let k be an integer, $\alpha \in \mathbb{F}_{2^m}$, we have

$$\sigma^k(\alpha) = \sum_{i=0}^{m-1} a_{i+k \ (\mathrm{mod}\ m)}\ \alpha_i$$

   If the basis α_0, α_1, $\cdots$, α_{m-1} of $\mathbb{F}_{2^m}/\mathbb{F}_2$ is a normal basis β, β^2, $\cdots$, $\beta^{2^{m-1}}$, then the linear transformation induced by σ is the Frobenius automorphism over $\mathbb{F}_{2^m}$, that is $\sigma(x) = x^2$. Moreover, we have $M_m(\mathbb{F}_2) = \mathbb{F}_{2^m}[\sigma]$, that is why we name $\sigma-$LFSR.
3. **Left Shift Operation L and Right Shift Operation R**. For $\alpha \in \mathbb{F}_{2^m}$, we define $\mathbf{L}(\alpha) = \sum_{i=0}^{m-2} a_i\alpha_{i+1}$, $\mathbf{R}(\alpha) = \sum_{i=1}^{m-1} a_{i+1}\alpha_i$.

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{m\times m} \qquad \mathbf{R} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}_{m\times m}$$

4. **Left Right Shift Combination Operation** $\sqcup_{s,t}$, where s,t are positive integers such that $0 < s,t < m$. Define

$$\sqcup_{s,t} = \mathbf{L}^s + \mathbf{R}^t$$

In the linear transformations above, σ is invertible; $\wedge_\gamma(\gamma \neq \sum_{i=0}^{m-1} \alpha_i)$, $\mathbf{L}$, $\mathbf{R}$ is noninvertible; for $\sqcup_{s,t}$, we have

Lemma 1. *Let $0 < s,t < m$ be positive integer, $\sqcup_{s,t}$ is invertible $\Leftrightarrow (s+t)|m$.*

Finite field with characteristic 2 is most convenient and widespread in practical application such as communication, cryptographic technique, etc. In addition, we generally let m be consistent with the word size (such as 8, 16, 32 and 64 bit, etc) of processor to exert the performance. For the sake of comparing $\sigma-$LFSR with the LFSR components of SOBER-t32, Turing, SNOW1.0 and SNOW2.0, we concentrate on constructing software oriented 32 bit $\sigma-$LFSR. It should be noticed that the theory of $\sigma-$LFSR and searching algorithm proposed in this section also be applicable to any finite field and any word length processor.

Now we review some symbols before. Let $M_{32}(\mathbb{F}_2)$ be the 32×32 matrix ring over $\mathbb{F}_2$, $GL_{32}(\mathbb{F}_2)$ the invertible matrix of $M_{32}(\mathbb{F}_2)$. Let $f(x) = x^n + C_{n-1}x^{n-1} + \cdots + C_1 x + C_0 \in M_m(\mathbb{F}_2)$ is the $\sigma-$polynomial of $\sigma-$LFSR where $C_{n-1}, C_{n-2}, \cdots, C_1 \in M_{32}(\mathbb{F}_2)$ and $C_0 \in GL_{32}(\mathbb{F}_2)$.

As mentioned in Section 1, And Operation $\wedge_\gamma, \gamma \in \mathbb{F}_{2^{32}}$, Left (Right) Shift Operation $\mathbf{L}^k, 0 < k < 32$ ($\mathbf{R}^k, 0 < k < 32$), Circular Rotation Operation $\sigma^k, -32 < k < 32$ and Left and Right Shift Combination Operation $\sqcup_{s,t}, 0 < s,t < 32$ are all linear transformations over $\mathbb{F}_2^{32}$. Circular Rotation Operation σ^k is invertible operation, the corresponding matrix is a invertible matrix, totals 63. And Operation $\wedge_\gamma, \gamma \in \mathbb{F}_{2^{32}}, \gamma \neq \sum_{i=0}^{31} \alpha_i$, Left (Right) Shift Operation are noninvertible operation, totals $2^{32} + 60$. For Left Right Shift Combination Operation $\sqcup_{s,t}$ there are 5 invertible and 416 noninvertible.

Now we define two sets W and V as follows.

$$\begin{aligned} W &= \{\wedge_\gamma, \mathbf{L}^k, \mathbf{R}^{k'}, \sqcup_{s,t} | \gamma \in \mathbb{F}_{2^{32}}, \gamma \neq 0 \text{ or } \textstyle\sum_{i=0}^{31} \alpha_i;\ 0 < k, k', s, t < 32\} \\ V &= \{\sigma^k, \sqcup_{s,t} | -32 < k < 32;\ 0 < s,t < 32, (s+t)|32\} \end{aligned} \tag{3}$$

It is obvious that the primitive $\sigma-$LFSRs of simple forms with coefficients in W and V are suitable to fast software implementation. Therefor we try to find primitive $\sigma-$polynomial with few terms and coefficients in $W \bigcup V$. On the other hand, we observe that $\#W = 2^{32} + 476$, $\#V = 68$, so we have rich space for choosing good primitive $\sigma-$polynomial.

By the previous results, we have the necessary condition to determine whether $\sigma-$LFSR is primitive so that we can give a (exhaustion) algorithm which need to improve further in the future work. Now we give the search algorithm in detail:

Algorithm 1 (Searching for Simple Primitive $\sigma-$LFRS).

1. **Choose Coefficients of $\sigma-$polynomial.** *First determine the number of word states and nonzero coefficients of $\sigma-$polynomial $f(x)$, usually starting from trinomial. The positions of nonzero coefficients are chosen randomly and the constant term is from V, others from $W \bigcup V$.*

2. **Compute Determinant.** *Computing the determinant* $g(x) = |F(x)| \in \mathbb{F}_2[x]$*, where* $F(x)$ *is the characteristic polynomial matrix of* $f(x)$*.*
3. **Determine Primitive.** *Check whether* $g(x)$ *is a primitive polynomial of degree* mn *over* $\mathbb{F}_2$*.*

By the searching algorithm above, we find massive primitive $\sigma-$polynomials with few terms. Consequently, one can further pick out good primitive $\sigma-$polynomial through software speed test and comparison of cryptographic properties. In this paper, we give two examples HHZ-1 and HHZ-2 with figure below, which are both primitive with 16 word states. HHZ-1 is a pentanomial with primitive $\sigma-$polynomial $h_1(x) = x^{16} + \wedge_{\gamma_1} x^{10} + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$ and HHZ-2 is a trinomial with primitive $\sigma-$polynomial $h_2(x) = x^{16} + \wedge_{\gamma_2} x^9 + \sqcup_{3,1}$, in which $\gamma_1 = \texttt{0x5e8491f8}, \gamma_2 = \texttt{0xbffffe4f}$.

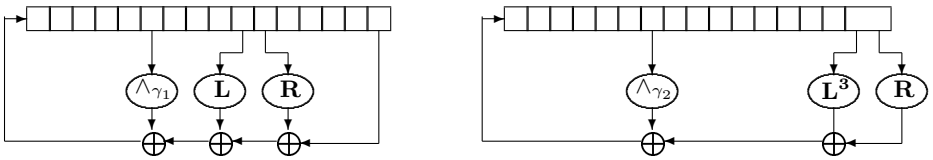


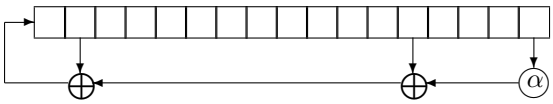
Fig. 3. Model of HHZ-1(left) and HHZ-2(right)

4 Software Implementation and Comparison

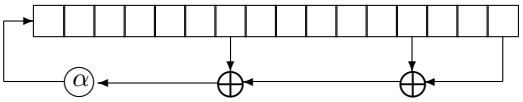
4.1 LFSR Components of Four Stream Cipher

For the sake of comparison, we inspect four stream cipher SOBER-t32, Turing, SNOW1.0 and SNOW2.0, whose LFSR components all come from primitive LFSR over finite field. In fact, they are carefully selected with high software efficiency and good cryptographic properties.

SOBER-t32 and Turing. SOBER-t32 is a candidate of NESSIE and its LFSR is 17th-order primitive LFSR over $\mathbb{F}_{2^{32}}$ with the recurrence relation: $s_{t+17} = s_{t+15} \oplus s_{t+4} \oplus \alpha s_t$ where $\alpha = \texttt{0xc2db2aa3}$, $t = 0, 1, ...$ Turing's LFSR is based on the foundation of SOBER-t32 which is identical with SOBER-t32 in form. It adopts the design rationale of SNOW2.0 so that α is a root of irreducible polynomial $y^4 \oplus \texttt{0xD0}y^3 \oplus \texttt{0x2B}y^2 \oplus \texttt{0x43}y \oplus \texttt{0x67}$ over $\mathbb{F}_8$ of degree 4.



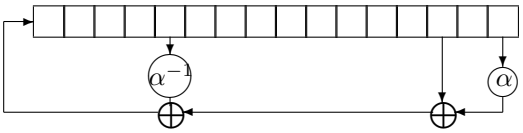
SNOW1.0 and SNOW2.0. SNOW1.0 is also a candidate of NESSIE and its LFSR is 16th-order primitive LFSR over $\mathbb{F}_{2^{32}}$ with the recurrence relation: $s_{t+16} = (s_{t+9} \oplus s_{t+3} \oplus s_t) * \alpha$ where $t = 0, 1, \cdots$, $\alpha = \texttt{0x20108403}$.



SNOW2.0 is a new version of SNOW1.0. The designer declared SNOW2.0 overcome certain latent weakness in the design and become more secure and faster in software. It's new recurrence relation is:

$$s_{t+16} = \alpha^{-1} s_{t+11} \oplus s_{t+2} \oplus \alpha s_t \quad t = 0, 1, ...$$

where α is a root of $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239} \in GF(2^8)[x]$, β is a root of $x^8 + x^7 + x^5 + x^3 + 1 \in \mathbb{F}_2[x]$ and figure is below.



4.2 Implementation

Fast software implementations. Software implementation of LFSR is very simple, but still has some skills to speed up. We describe two implementations of LFSR [6] and take SNOW1.0 as example to explain. Let $R[0], R[1], \cdots, R[15]$ be the 16 word states.

- Basic Mode: We simply calculate the feedback value, shift all the other values of states and then renew last state. The code is as below:

  $new = (R[0] \oplus R[3] \oplus R[9]) * \alpha;\ R[0] = R[1];\ \cdots;\ R[14] = R[15];\ R[15] = new;$

  The advantage of this mode is the compact, clear and perspicuous code; the shortcoming is low efficiency since most time is consumed on state shift.
- Fast Mode: We use position shift to implement sequence generation. This mode is the fastest, but increase the size of code. We'll illustrate the mode as follows:

  $$\begin{aligned} &R[0] = (R[0] \oplus R[3] \oplus R[9]) * \alpha; \\ &R[1] = (R[1] \oplus R[4] \oplus R[10]) * \alpha; \\ &\cdots\cdots \\ &R[15] = (R[15] \oplus R[2] \oplus R[8]) * \alpha; \end{aligned}$$

Software implementations of detail. In fact, the difference of code will have great influence on the efficiency for a concrete algorithm. LFSRs of the four stream ciphers involves multiplications $x \cdot \alpha$ over $\mathbb{F}_{2^{32}}$, where α is fixed. In SNOW2.0 and Turing, table look-up is used to implement the multiplication, that is $x \cdot \alpha = (x \ll 8) \oplus table[x \gg 24]$ where table is uniquely decided by α. In SOBER-t32 and SNOW1.0, $x \cdot \alpha$ can be implemented in three kinds of methods. The first uses branch sentence, namely $\texttt{if}((\texttt{x} \gg 31) == 1)\ \ \texttt{x} \cdot \alpha = (\texttt{x} \ll 1) \oplus \alpha;\ \ \texttt{else}\ \ \texttt{x} \cdot \alpha = (\texttt{x} \ll 1)$. The second uses shift and multiplication,

that is $x \cdot \alpha = (x \ll 1) \oplus ((x \gg 31) * \alpha)$. And the third is table look-up, let $table[0] = 0$, $table[1] = \alpha$, so $x \cdot \alpha = (x \ll 1) \oplus table[x \gg 31]$. But in the modern processor, there still exists other skills to improve the speed in software, that is, if the size of table is not large, it may be completely placed in cache of processor so that it is not necessary to visit the memory each time, as a result the third way is quicker than the second way.

For our $\sigma-$LFSRs only involved Xor, And, Shift, Circular Rotation, etc which are the fundamental machine instructions, their software implementations are sure to be simple and fast with little resources. In fact, the hardware implementations of the our $\sigma-$LFSR is also easy.

4.3 Cryptographic Properties of $\sigma-$LFSR

LFSRs used in stream ciphers are generally primitive sequences so that they have pseurandom properties including the maximal period, same distributions of element frequency within a block and good autocorrelation properties, etc. In addition, in order to resist some cryptanalysis such as correlation attack, etc, the generating polynomial of LFSR is required to be selected carefully. However, complicated generating polynomial of LFSR will inevitably lead to decrease the efficiency in software. Designers must carefully take care of the balance between efficiency and security. In fact SNOW2.0 is a extremely splendid model in the design and has obtained a desirable result.

Divisibility. In order to resist correlation attack [14] and distinguishing attack [13], the minimal polynomial of bitwise primitive sequence should not be a divisor of a polynomial with few terms and low degree. SNOW2.0 uses a inverse element α^{-1} as coefficient to strengthen the resistance against certain correlation attack, as discussed in [14], but the speed of software slows down half of SNOW1.0. SOBER-t32 and Turing haven't solved this problem since the designers think that the two kinds of attacks have practical significance only when the correlation of nonlinear function cannot be neglected.

In HHZ-1 and HHZ-2, because the Left Shift and Right Shift Operations are noncommutative and the two operations at different positions so that the coefficients of the primitive $\sigma-$polynomials $h_1(x), h_2(x)$ are noncommutative. As a result, it is difficult to find a multiple polynomial with few terms divisible by $h_1(x), h_2(x)$ separately. So we give a solution to resist correlation and distinguishing attacks.

Weight of minimal polynomial. Weight of minimal polynomial is defined as the number of nonzero terms in minimal polynomial except the highest degree item. It is the best choice that the weight of the minimal polynomial is about half of the polynomial degree because this can achieve maximal confusion effect and resistance to correlation attack. Weight of SNOW2.0, SOBER-t32, Turing and HHZ-1 are all about half of degree which achieved the best effect on security. SNOW1.0 and HHZ-2 are not, but have excellent efficiency. Therefore HHZ-2 is also a pretty choice in the circumstance that speed is of specially importance.

4.4 Comparison Result

For the six $\sigma-$LFSRs above, we used the standard C language programming to carry on the speed test in basic mode and fast mode. Test machine is Pentium IV 1.5G CPU with 256M memory, Microsoft Windows 2000 Professional operation system, VC6.0 compiler system and iterative step is $2^{30} = 1073741824$. Testing results come from Table 1.

Algorithm	Order	Memory	Weight	Divisible	Fast Speed (Gbits/s)	Speed Ratio	Basic Speed (Mbits/s)
Snow1.0	16	(16+3)*4	82/512	Y	11.38	100	587
Snow2.0	16	(16+512)*4	250/512	N	7.21	63	587
Sober-t32	17	(17+2)*4	272/544	N	10.07	88	599
Turing	17	(17+256)*4	266/544	Y	9.84	86	599
HHZ-1	16	16*4	256/512	N	11.45	101	599
HHZ-2	16	16*4	74/512	N	13.13	115	603

Table 1. Test Result

Speed is scaled with throughput, every clock $\sigma-$LFSR outputs 32 bit. So in fast mode, fast speed=iteriative step $\times 32\times$ state order/(time consuming); in basic mode, basic speed=iteriative step $\times 32$ /(time-consuming). For convenience of comparison, using SNOW1.0 as standard, speed ratio is the percentage of respective speed division by speed of SNOW1.0. Memory refers to the need of every algorithm in implementation.

From Table 1, we see that the memories of SNOW2.0 and Turing are larger, 2112 byte and 1092 byte respectively. Certainly, several thousands byte memory in modern computer are not a problem; but in embedded systems or chips, it is necessary to take into account the resource consumption.

As a result, HHZ-1 achieves the best feature in divisibility and weight of minimal polynomial. Moreover, it is much faster than SNOW2.0. In security and efficiency, HHZ-1 is superior to SOBER and Turing. Although HHZ-2 does not achieve best weight of minimal polynomial, its speed holds the obvious superiority.

5 Conclusions

We introduce the concept of $\sigma-$LFSR, a word oriented LFSR, which can be used in the design of modern stream cipher. We offer an algorithm to search for primitive $\sigma-$LFSR and obtain many primitive $\sigma-$LFSRs such as HHZ-1 or HHZ-2, etc, which use less memory and computer instructions. In addition, some $\sigma-$LFSRs buildup the ability against fast correlation attack and distinguish attack and have fast software implementation so that they could be used as the LFRS component in many cryptographic schemes based on software.

References

1. S.W. Golomb, Shift Register Sequences. Holden-Day, San Francisco, 1967.
2. R. Lidi and H. Niederreiter, Finite Fields, in: Encyclopedia of Mathematics and its Applications **20** (1983), Cambridge University Press.
3. B. Preneel, Introduction to the Proceedings of the Fast Software Encryption 1994 Workshop (Ed. Bart Preneel), LNCS **1008** (1995) 1–5
4. P. Hawkes and G.G. Rose, Primitive Specification and Supporting Documentation for SOBER-t16 Submission to NESSIE, Proceedings of the first NESSIE Workshop (2000)
5. P. Hawkes and G.G. Rose, Primitive Specification and Supporting Documentation for SOBER-t32 Submission to NESSIE, Proceedings of the first NESSIE Workshop, (2000)
6. P. Ekdahl and T. Johansson, SNOW – a new stream cipher, Proceedings of the first NESSIE Workshop, (2000)
7. P. Ekdahl and T. Johansson, A New Version of the Stream Cipher SNOW, Selected Areas in Cryptography, LNCS **2595** (2002) 47–61
8. P. Hawkes and G.G. Rose. Turing: A Fast Stream Cipher, Fast Software Encryption, LNCS **2887** (2003) 24–26
9. B. Tsaban and U. Vishne, Efficient Linear Feedback Shift Registers with Maximal Period, Finite Fields and Their Applications **8(2)** (2002) 256–267
10. M. Dewar and D. Panario, Linear transformation shift registers, IEEE Transactions on Information Theory **49(8)** (2003) 2047–2052
11. K. Chen, W. Millan and L.R. Simpson, Perspectives on Word Based Stream Ciphers, Cryptographic Algorithms and their Uses (2004) 14–27
12. Guang Zeng, Kaicheng He and Wenbao Han, Word-oriented Feedback Shift Registers: $\sigma-$LFSR. (to appear)
13. P. Ekdahl and T. Jonhansson, Distinguishing Attacks on SOBER-t16 and t32, Fast Software Encryption, LNCS **2365** (2002) 210–224
14. P. Hawkes and G.G. Rose, Guess-and-Determine Attacks on SNOW, Selected Areas in Cryptography, LNCS **2595** 37–46
15. A. Klimov and A. Shamir, A New Class of Invertible Mappings, Cryptographic Hardware and Embedded Systems, LNCS **2523** (2002) 470–483
16. A. Klimov and A. Shamir, Cryptographic Applications of T-Functions, Selected Areas in Cryptography, LNCS **3006** (2003) 248–261

A Minimal Polynomial of HHZ-1 and HHZ-2

HHZ-1 is equivalent to 32 parallel binary LFSRs with minimal polynomial $p_1(x)$, as shown in binary, with the first bit being the leading term and decreasing exponent:
10000010000001000000100110100100100110101100011110100101110110101100011O
0110011111000000101000101100110111000101000111100101001100011000111101
1010110001001000100100001111000000001011101010110010101000110100100011 1
0111011011110101000010101100011110001111110000101000101100111100111111
1110111010111000110001010011010111011101110000001100100001110010111111 1
1011110110111101111100111010001110011100110100110100011110101011010110
1101111100011101010101100010011111110000011100010100110010000011000100 0

11110000000110000000001
That is, $p_1(x) = x^{512} + x^{506} + x^{500} + x^{494} + \cdots + x^{31} + x^{30} + x^{26} + x^{22} + x^{21} + x^{20} + x^{11} + x^{10} + 1$. This polynomial has 257 nonzero terms.

HHZ-2 is equivalent to 32 parallel binary LFSRs with minimal polynomial $p_2(x)$:
10000001000000100000010000000000000000000000000000000010000001100000
11000001100000010000001000000100000010000011000001000000101000010000000
00000000000000000000000100001010000111000011000001101000010100001010000
01000000101000010100000010000011000000100000011000001100000100000111000
00110000011100000100000001000000000000010000010000000100000000000000110
00000000000100000010000000000000010000001000000100000100000001000000001
00000010000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000001
That is, $p_2(x) = x^{512} + x^{505} + x^{498} + x^{491} + \cdots + x^{144} + x^{128} + x^{121} + x^{114} + x^{108} + x^{100} + x^{93} + x^{86} + 1$. This polynomial has 75 nonzero terms.

B Some Experiment Results

Here we list some results of experiments in finding HHZ-1 and HHZ-2.

$\sigma-$polynomial	Weight	$\sigma-$polynomial	Weight
$x^{16} + \sigma x^3 + \wedge\texttt{0x5437af9e}x^2 + \sigma^2$	66	$x^{16} + \sigma^2 x^3 + \wedge\texttt{0x5437ad31}x^2 + \sigma$	38
$x^{16} + \sigma x^3 + \wedge\texttt{0x5437d77f}x^2 + \sigma^2$	76	$x^{16} + \sigma^2 x^3 + \wedge\texttt{0x5437b48d}x^2 + \sigma$	44
$x^{16} + \sigma x^3 + \wedge\texttt{0x5437b148}x^2 + \sigma^2$	48	$x^{16} + \sigma^2 x^3 + \wedge\texttt{0x5437ada4}x^2 + \sigma$	58
$x^{16} + \sigma x^3 + \wedge\texttt{0x5437b89c}x^2 + \sigma^2$	58	$x^{16} + \sigma^2 x^3 + \wedge\texttt{0x5437afc1}x^2 + \sigma$	64
$x^{16} + \sigma x^3 + \wedge\texttt{0x5437af9e}x^2 + \sigma$	62	$x^{16} + \sigma^2 x^3 + \wedge\texttt{0x5437b4c8}x^2 + \sigma$	74

Table 2. Primitive $\sigma-$LFSR I

$\sigma-$polynomial	W	$\sigma-$polynomial	W
$x^{16} + \wedge\texttt{0x5e8491f8}x^3 + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$	256	$x^{16} + \wedge\texttt{0xfe8492a4}x^3 + \mathbf{L^{13}}x^6 + \mathbf{R}x^5 + 1$	164
$x^{16} + \wedge\texttt{0xfe84a29b}x^3 + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$	230	$x^{16} + \wedge\texttt{0xfe849301}x^3 + \mathbf{L^3}x^6 + \mathbf{R}x^5 + 1$	176
$x^{16} + \wedge\texttt{0xfe84a4a2}x^3 + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$	224	$x^{16} + \wedge\texttt{0x5e8491c6}x^3 + \mathbf{L^3}x^6 + \mathbf{R}x^5 + 1$	176
$x^{16} + \wedge\texttt{0xfe84a2dd}x^3 + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$	180	$x^{16} + \wedge\texttt{0x5e84914b}x^3 + \mathbf{L^5}x^6 + \mathbf{R}x^5 + 1$	126
$x^{16} + \wedge\texttt{0xfe84a6b7}x^3 + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$	232	$x^{16} + \wedge\texttt{0xfe84a27a}x^3 + \mathbf{L}x^6 + \mathbf{R}x^5 + 1$	222

Table 3. Primitive $\sigma-$LFSR II

$\sigma-$polynomial	Weight	$\sigma-$polynomial	Weight
$x^{16} + \wedge\texttt{0x7ceabddf}x^9 + \sqcup_{1,1}$	124	$x^{16} + \wedge\texttt{0x7ceabf4}x^9 + \sqcup_{1,1}$	100
$x^{16} + \wedge\texttt{0xfffffff5e}x^9 + \sqcup_{1,1}$	80	$x^{16} + \wedge\texttt{0xbffffe4f}x^9 + \sqcup_{3,1}$	74
$x^{16} + \wedge\texttt{0xbffffea8}x^9 + \sqcup_{3,1}$	68	$x^{16} + \wedge\texttt{0xfffffff81}x^9 + \sqcup_{3,1}$	54
$x^{16} + \wedge\texttt{0xbfffffac}x^9 + \sqcup_{5,3}$	40	$x^{16} + \wedge\texttt{0xfffffff13}x^9 + \sqcup_{7,1}$	42
$x^{16} + \wedge\texttt{0xffff1113}x^9 + \sqcup_{5,3}$	34	$x^{16} + \wedge\texttt{0xfffff16}x^9 + \sqcup_{15,1}$	24

Table 4. Primitive $\sigma-$LFSR III