Science in China Series F: Information Sciences

© 2007 Science in China Press

Springer-Verlag

A trinomial type of σ-LFSR oriented toward software implementation

ZENG Guang[†], HE KaiCheng & HAN WenBao

Department of Applied Mathematics, Information Engineering University, Zhengzhou 450002, China

In this paper, we introduce a new type of feedback shift register based on words, called σ-linear feedback shift register (σ-LFSR) which can make full use of the instructions of modern CPUs so that we can find good σ-LFSR with simple structure and fast software implementation. After analysis, we find a class of simple σ-LFSR with maximal period and give an algorithm of searching for those σ-LFSRs. As a result, we provide a new optional fast component in the design of modern word-based stream ciphers.

finite field, stream cipher, σ-LFSR, primitive polynomial, fast software implementation

The system of stream cipher is generally composed of several LFSRs and some nonlinear functions. Traditional stream cipher primarily adopted bit-based LFSR, which have profound theory and wide applications[1,2]. Bit-based LFSR extremely fits for hardware implementation, but many bit manipulations are required for software and only one bit output per step, while modern CPU can process 32 or 64 bits operations, even with MMX instructions which provide 4 or 5 32 bit instructions per a clock. With the application of modern CPU, software implementation efficiency of traditional LFSR is low.

Since the first "Fast Software Encryption (FSE)" conference in 1993, the design of crypto system suitable for software implementation is an important trend in the world of modern cryptology. In the following research, there appear to be many new ideas and methods of design of stream cipher for fast software implementation. Modern stream cipher is generally 4—5 times faster than block cipher in speed but lower in security, so how to design security stream cipher with high software efficiency attracts much attention. In FSE of 1994, Preneel[3] set forth a challenge to design LFSRs which exploit the parallelism offered by the word oriented operations of modern processors.

Considerable following up happened to the study of software implementation oriented stream

Received July 15, 2006; accepted January 18, 2007
doi: 10.1007/s11432-008-0036-y

[†]Corresponding author (email: sunshine_zeng@sina.com)

Supported by the National Natural Science Foundation of China (Grant No. 60503011), the National High-Tech Research and Development Program of China (863 Program) (Grant No. 2006AA01Z425) and the National Basic Research Program of China (973 Program) (Grant No. 2007CB807902)

cipher. Europe launched NESSIE project, calling for a broad set of cryptographic primitive algorithm. Finally, all candidates of stream ciphers did not pass. This also proposed a higher challenge for stream cipher designer: designing stream cipher for fast software implementation with security identical with block cipher. Japan and Korea also collected cryptographic algorithm standard one after another. In 2005, Europe's ECRYPT NoE eSTREAM project once again called for stream ciphers. In all 34 collected stream ciphers, 22 take software implementation as one of the design goals. Thus, it can be seen that designing fast software implementation stream cipher has become an active research topic at present.

It is clear that modern stream cipher designs, represented by proposals, such as Ssc2[4], Panama[5], Mugi[6], Seal[7], Scream[8], Rabbit[9], Helix[10], Sober[11,12], Turing[13], Snow[14,15], and many more, are far from classical designs like traditional nonlinear filter generator and nonlinear combination generators, etc. One major difference is that classical designs are bit-oriented, whereas modern designs tend to operate on (e.g., 32 or 64 bits) words to provide efficient software implementations. Modern stream ciphers use building blocks very similar to those used in block cipher. The transition from bit to word leads to introduce different operations in stream cipher design, even taking use of S boxes or other complicated linear transformations which are commonly used in block ciphers. It should be noticed that most proposals use word-based LFSR as the pseudorandom source. In 2003, Tsaban and Vishne[16] introduced the concept of linear transformation shift register (TSR) which is a word-based LFSR. Dewar and Panario[17] further studied searching algorithm of primitive TSR. In addition, the source part of stream cipher Sober[11,12], Turing[13], and Snow[14,15] is primitive word-based LFSR over finite field which has desirable pseudorandom properties and fast software implementation.

Word-based LFSR has already become the important constituent of modern stream cipher. Before word-based LFSRs wide use, we must research their pseudorandom properties and software suitability. In this paper, we introduce the concept of σ-linear feedback shift register (σ-LFSR)[18], which is the generalization of TSR proposed by Tsaban and Vishne. We focus the research on cryptographic properties, construction with few operations, and fast implementation of σ-LFSR. We give an overview of general word-based LFSR and analysis of cryptographic properties. As a result, we obtain an algorithm of searching for primitive trinomial σ-LFSR. Our results indicate that this kind of σ-LFSR has the good pseudorandom and fast software implementation.

1 Model of σ-LFSR

Let q be a prime, m a positive integer, $\mathbb{F}_q$ a finite field with q elements, and $\mathbb{F}_{q^m}$ an extension of $\mathbb{F}_q$ with degree m. Let n be a positive integer and $a_0,a_1,\cdots,a_n$ be given elements of $\mathbb{F}_{q^m}$. A sequence $s^\infty = s_0, s_1, \cdots$ such that

$$s_{k+n} = -(a_0 \cdot s_k + a_1 \cdot s_{k+1} + \cdots + a_{n-1} s_{k+n-1}) \quad k = 0,1,\cdots$$

is called nth order linear recurrence sequence in $\mathbb{F}_{q^m}$. It is obvious that many additions and multiplications of $\mathbb{F}_{q^m}$ are required in traditional LFSR. No matter whether we use polynomial basis or normal basis of $\mathbb{F}_{q^m}/\mathbb{F}_q$, addition is easy while multiplication is complicated, so table lookup technique is often used to speed up multiplication in software implementation. For modern popular 32 bit processor, in order to sufficiently exploit computation resources and avoid mass bit operations, m=32 is generally selected for q=2. In implementation, it is infeasible to have storage $2^{32}\times 2^{32}$ size table. Even if we can decrease storage amount, the time of table lookup will greatly decline the efficiency. The operations, including AND, OR, NOT, XOR, SHIFT, etc., are fundamental instructions of CPU, where circular rotation is not only simple and suitable for software or hardware, but also accelerates information diffusion in word. Therefore, we introduce the circular rotation to the word-based LFSR named σ-LFSR. Now, we define the circular rotation operation in a mathematical way.

Definition 1. Let $\alpha,\alpha^q,\cdots,\alpha^{q^{m-1}}$ be the normal basis of linear vector space $\mathbb{F}_{q^m}/\mathbb{F}_q$ and $\beta=k_0\alpha+k_1\alpha^q+\cdots k_{m-1}\alpha^{q^{m-1}}\in\mathbb{F}_{q^m}$ where $k_0,k_1,\cdots,k_{m-1}\in\mathbb{F}_q$, then circular rotation operation over $\mathbb{F}_{q^m}$ is defined as

$$\sigma(\beta)=\sigma(k_0\alpha+k_1\alpha^q+\cdots k_{m-1}\alpha^{q^{m-1}})\triangleq k_{m-1}\alpha+k_0\alpha^q+\cdots k_{m-2}\alpha^{q^{m-1}}.$$

Remark 1. In actual implementation, σ can be realized by circular rotation operation and it is a qth power function over $\mathbb{F}_{q^m}$.

It is obvious that σ is a linear map over $\mathbb{F}_{q^m}/\mathbb{F}_q$. Moreover, any element c of $\mathbb{F}_{q^m}$ can induce a linear map C over $\mathbb{F}_{q^m}/\mathbb{F}_q$, where $C:\mathbb{F}_{q^m}\to\mathbb{F}_{q^m}$ $C(\alpha)=c\alpha$ for $\alpha\in\mathbb{F}_{q^m}$. We define $\mathbb{F}_{q^m}[\sigma]=\sum_i a_i\sigma^{k_i}b_i$, where $a_i,b_i\in\mathbb{F}_{q^m}$, $k_i,i\in\mathbb{Z}$, and denotes by $\mathcal{A}_{q^m}=\mathbb{F}_{q^m}[\sigma]$.

Definition 2. If $\beta=k_0\alpha+k_1\alpha^q+\cdots k_{m-1}\alpha^{q^{m-1}}\in\mathbb{F}_{q^m}$ and $\gamma=l_0\alpha+l_1\alpha^q+\cdots l_{m-1}\alpha^{q^{m-1}}\in\mathbb{F}_{q^m}$, where $k_0,k_1,\cdots,k_{m-1}\in\mathbb{F}_q$ and $l_0,l_1,\cdots,l_{m-1}\in\mathbb{F}_q$, then

$$\beta+\gamma\triangleq((k_0+l_0),\cdots,(k_{m-1}+l_{m-1})).$$

In fact, this definition is the common addition in finite field, but it should be noticed that if q=2, above defined "+" is the XOR operation in computer and can be implemented by computer instruction.

Definition 3. Let n be a positive integer, and $c_0(\sigma),c_1(\sigma),\cdots,c_{n-1}(\sigma)$ are elements in $\mathcal{A}_{q^m}$. A sequence $s^\infty=s_0,s_1,\cdots$ over $\mathbb{F}_{q^m}$ such that

$$s_{i+n}=-(c_0(\sigma)\cdot s_i+c_1(\sigma)\cdot s_{i+1}+\cdots+c_{n-1}(\sigma)s_{i+n-1})\quad i=0,1,\cdots \tag{1}$$

is called nth order σ-linear recurrence sequence and polynomial

$$f(x)=x^n+c_{n-1}(\sigma)x^{n-1}+\cdots+c_1(\sigma)x+c_0(\sigma)\in\mathcal{A}_{q^m}[x]$$

is called σ-polynomial of sequence s^∞. The model of σ-LFSR is as in Figure 1.

Remark 2. If we take $c_0(\sigma)=a_0,\cdots,c_{n-1}(\sigma)=a_{n-1}$, where $a_0,a_1,\cdots,a_n\in\mathbb{F}_{q^m}$ in Definition 3, eq. (1) gives traditional nth order linear recurrence sequence over finite field.

2 Properties of σ-LFSR

It is easy to see that for any $\beta\in\mathbb{F}_{q^m}$, we have $\sigma\beta=\beta^q\sigma$, so $\mathcal{A}_{q^m}$ is a noncommutative algebra

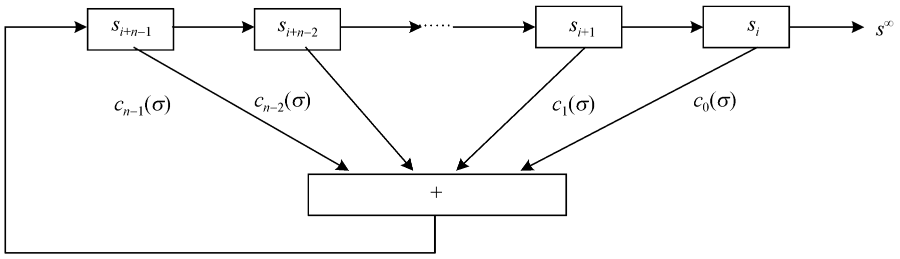

Figure 1 Model of σ-LFSR.

over $\mathbb{F}_q$. In order to utilize algebra theory to study σ-LFSR, we have

Theorem 1. Let $M_m(\mathbb{F}_q)$ be the $m\times m$ matrix ring over $\mathbb{F}_q$. We have $M_m(\mathbb{F}_q)\cong \mathcal{A}_{q^m}$.

Proof. Let $\alpha,\alpha^q,\cdots,\alpha^{q^{m-1}}$ be the normal basis of $\mathbb{F}_{q^m}/\mathbb{F}_q$. Because σ is a linear map over $\mathbb{F}_{q^m}/\mathbb{F}_q$ with order m and $\sigma\beta=\beta^q\sigma$, we have

$$\mathcal{A}_{q^m}=\left\{\sum_{j=0}^{m-1}\sum_{i=0}^{m-1}a_{ij}\alpha^{q^i}\sigma^j \mid a_{ij}\in\mathbb{F}_q\right\}.$$

Notice that elements in $\mathcal{A}_{q^m}$ are all linear map over $\mathbb{F}_{q^m}/\mathbb{F}_q$. Let γ be the natural map from $\mathcal{A}_{q^m}$ to $M_m(\mathbb{F}_q)$, which sends linear map to its associated matrix under the normal basis $\alpha,\alpha^q,\cdots,\alpha^{q^{m-1}}$. Obviously, γ is an injection and $\gamma(\mathcal{A}_{q^m})\subseteq M_m(\mathbb{F}_q)$. In fact, $|M_m(\mathbb{F}_q)|=q^{m^2}$. Thus, we only need to prove $|\mathcal{A}_{q^m}|=q^{m^2}$. If there exist $\theta_0,\theta_1,\cdots,\theta_{m-1}\in\mathbb{F}_{q^m}$ such that $\theta_0+\theta_1\sigma+\cdots+\theta_{m-1}\sigma^{m-1}=0$, then for any $\beta\in\mathbb{F}_{q^m}$, we have

$$\theta_0\beta+\theta_1\beta^q+\cdots+\theta_{m-1}\beta^{q^{m-1}}=0.$$

That is, the polynomial $\theta_0x+\theta_1x^q+\cdots+\theta_{m-1}x^{q^{m-1}}$ has q^m roots in $\mathbb{F}_{q^m}$ at least, so $\theta_0=\theta_1=\cdots=\theta_{m-1}=0$. Hence, $|\mathcal{A}_{q^m}|=q^{m^2}$ and the proof is finished.

In the following paper, we always denote the associated matrix of $c_i(\sigma)\in\mathcal{A}_{q^m}$ under the normal basis $\alpha,\alpha^q,\cdots,\alpha^{q^{m-1}}$ for $0\leqslant i\leqslant n-1$ by $C_i\in M_m(\mathbb{F}_q)$. Therefore, in σ-LFSR, the coefficients can be replaced by $m\times m$ matrix over $\mathbb{F}_q$, and the recurrence (1) is rewritten as

$$s_{i+n}=-(C_0\cdot s_i+C_1\cdot s_{i+1}+\cdots+C_{n-1}s_{i+n-1})\quad i=0,1,\cdots \tag{2}$$

Remark 3. Let $T\in M_m(\mathbb{F}_q)$ be the associated matrix with some linear map over $\mathbb{F}_{q^m}/\mathbb{F}_q$, if $C_0=C_1=\cdots=C_{n-1}=T$, then the σ-LFSR defined by (2) is the TSR[16,17], which is introduced by Tsaban and Vishne, so TSR is a special case of σ-LFSR.

Remark 4. Elements of $\mathbb{F}_{q^m}$ can be regarded as m-tuple vector over binary field when $q=2$, meanwhile CPU instructions AND operation can be introduced to σ-LFSR over $\mathbb{F}_{2^m}$. Let

$V=(a_1,a_2,\cdots,a_m)\in\mathbb{F}_{2^m}$ and symbol $\&_V$ be defined as AND operation with vector V.

By the definition of σ-LFSR, σ-linear recurrence sequence s^∞ is an ultimately periodic sequence, i.e., there exists integer $r>0$ and the least nonnegative integer $n_0\geqslant 0$ such that $s_{n+r}=s_n$ for all $n\geqslant n_0$, r is called the period of the sequence s^∞ and n_0 is called the preperiod. If $n_0=0$, the sequence s^∞ is called periodic. Similar to linear recurring sequence in finite fields, we can also establish a condition for the periodicity of σ-linear recurrence sequences.

Theorem 2. Let s^∞ satisfy the linear recurrence relation (1), then s^∞ is periodic if and only if $c_0(\sigma)$ is invertible of $\mathcal{A}_{q^m}$.

Proof. If the coefficient $c_0(\sigma)$ is an invertible in $\mathcal{A}_{q^m}$, r is the least period and n_0 its preperiod of sequence s^∞. Suppose s^∞ is not periodic, so we have $n_0\geqslant 1$. By definitions of period and preperiod, $s_{n+r}=s_n$ for all $n\geqslant n_0$. From (1) with $i=n_0+r-1$, we obtain

$$\begin{aligned}s_{n_0-1+r}&=-c_0^{-1}(\sigma)(s_{n_0-1+r+n}+c_1(\sigma)s_{n_0+r}+\cdots+c_{n-1}(\sigma)s_{n_0+r+n-2})\\&=-c_0^{-1}(\sigma)(s_{n_0-1+n}+c_1(\sigma)s_{n_0}+\cdots+c_{n-1}(\sigma)s_{n_0+n-2}).\end{aligned}$$

Using (1) with $i=n_0-1$, we find the same expression for s_{n_0-1+r} and s_{n_0-1}, so $s_{n_0-1+r}=s_{n_0-1}$. This is a contradiction to the definition of the preperiod, so $n_0=0$ and s^∞ is periodic.

On the other hand, suppose $c_0(\sigma)$ is noninvertible in $\mathcal{A}_{q^m}$, r is the least period and n_0 its preperiod of sequence s^∞. Then, associated matrix of $c_0(\sigma)$ is a singular in $M_m(\mathbb{F}_q)$ so that there exists $a\in\mathbb{F}_{q^m}$ $(a\neq 0)$ such that $c_0(\sigma)(a)=0$. Let $s_0=a$, $s_1=s_2=\cdots=s_{n-1}=0$, so we can obtain a σ-linear recurrence sequence with period r. According to recurrence relation (1),

$$s_n=-(0a+0s_1+\cdots+0s_{n-1})=0.$$

We have $s_1=s_2=\cdots=s_n=0$ and $s_i=0$ for all $i\geqslant n$. In particular, $s_r=0$. As a result $s_0=0$ since r is the period. This contradicts to $a\neq 0$, moreover, $c_0(\sigma)$ is invertible of $\mathcal{A}_{q^m}$.

Theorem 2 is the extension of traditional LFSR. It is easy to see that the conclusion is identical to the result about traditional LFSR when all the coefficients in linear recurrence relation (1) are in $\mathbb{F}_{q^m}$. In fact, Theorem 2 gives the condition for regularity of σ-LFSR. It is well-known that finding pseudorandom sequences with maximal period is of vital significance in research of stream cipher. Now, we give the definition of primitive σ-LFSR.

Definition 4. Let s^∞ be the sequence with the period $q^{mn}-1$ such that nth order σ-LFSR (1) over $\mathbb{F}_{q^m}$, we call s^∞ primitive sequence generated by σ-LFSR (1), its σ-polynomial $f(x)$ primitive σ-polynomial and σ-LFSR (1) primitive σ-LFSR.

In fact, there exists primitive σ-LFSR over $\mathbb{F}_{q^m}$. Moreover, we have

Theorem 3. Let s^∞ be the sequence such that nth order σ-LFSR (1) over $\mathbb{F}_{q^m}$ with σ-polynomial $f(x)=x^n+c_{n-1}(\sigma)x^{n-1}+\cdots+c_1(\sigma)x+c_0(\sigma)\in\mathcal{A}_{q^m}[x]$, where $c_0(\sigma)$ is invertible,

$F(x)= x^n + C_{n-1}x^{n-1} +\cdots+C_1x +C_0 \in M_m(F_q)[x]$ the corresponding matrix polynomial of $f(x)$ under given normal basis. Then, s^∞ is primitive $\Leftrightarrow$ the determinant $|F(x)|$ is primitive polynomial over $\mathbb{F}_q$ with degree mn.

Proof. Let $S_m=(s_m, s_{m+1},\cdots s_{n+m-1})$ be the mth state of σ-LFSR (1), we define state transition matrix from S_{m+1} to S_m as follows:

$$T=\begin{pmatrix} 0 & 0 & 0 & 0 & C_0 \\ E_m & 0 & 0 & 0 & C_1 \\ 0 & \ddots & 0 & 0 & \vdots \\ 0 & 0 & E_m & 0 & C_{m-2} \\ 0 & 0 & 0 & E_m & C_{m-1} \end{pmatrix}_{mn\times mn},$$

where E_m is an $m\times m$ identity matrix over $\mathbb{F}_q$. Thus, σ-LFSR is a primitive if and only if the multiplicative order of the $mn\times mn$ matrix $T \in GL_{mn}(\mathbb{F}_q)$ is $q^{mn}-1$. It is familiar that the order of invertible matrix over finite field equals the order of the last invariant factor of the corresponding characteristic matrix. Let $xE_{mn}+T$ be characteristic matrix of T, so sequence s^∞ is primitive if and only if the invariant factors of $xE_{mn}+T$ are $1,1,\cdots, g(x)$, where $g(x)$ is an mn degree primitive polynomial over $\mathbb{F}_q$. On the other hand, we know $g(x)=|xE_{mn}+T|=|F(x)|$, so s^∞ is primitive if and only if $|F(x)|$ is a primitive polynomial with degree mn over $\mathbb{F}_q$.

Primitive σ-LFSR is of considerable interest, and Theorem 3 gives the condition to determine primitivity. We expect to find such primitive σ-LFSR with simple σ-polynomial so that we can achieve both maximal period sequence and fast software implementation. One of the simplest σ-polynomial comes from all the coefficients which are circular rotations. Now, we discuss those σ-polynomials.

Theorem 4. Let $f(x)=x^n+\sigma^{k_{n-1}}x^{n-1}+\cdots+\sigma^{k_1}x+\sigma^{k_0}\in\mathcal{A}_{q^m}[x]$ with $m>1$ where k_i is an integer, $0\leqslant k_i\leqslant m-1$ and $0\leqslant i\leqslant n-1$, then $f(x)$ is not a primitive σ-polynomial.

Proof. Let $\alpha,\alpha^q,\cdots,\alpha^{q^{m-1}}$ be the normal basis of $\mathbb{F}_{q^m}/\mathbb{F}_q$, then the associated matrix of σ under the normal basis is

$$A=\begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}. \tag{3}$$

Then, the corresponding matrix polynomial of $f(x)$ is $F(x)=x^n+A^{k_n-1}x^{n-1}+\cdots+A^{k_1}x+A^{k_0}$. After converting the matrix polynomial to polynomial matrix, we can find $F(x)$ is an $m\times m$ circulant matrix over $\mathbb{F}_q$. Using the properties of determinant, add each row to the first row from second row to last and obtain the determinant of $F(x)$ as

$$|F(x)| = (x^n + k_{n-1}x^{n-1} + \cdots + k_1 x + k_0)g(x),$$

where $g(x) \in \mathbb{F}_q[x]$. Since $\deg(F(x)) = mn$, $\deg(g(x)) = (m-1)n$. So $|F(x)|$ is a reducible polynomial over $\mathbb{F}_q$. By Theorem 3, $f(x)$ is not primitive.

If σ-LFSR is constructed by circular rotation, then its σ-polynomial $f(x)$ is not primitive by Theorem 4, so this kind of σ-LFSR is not primitive.

3 σ-LFSR with trinomial over finite fields of characteristic 2

Traditional LFSRs over binary field have wide applications because of simple hardware implementation and high efficiency. However, implementation of traditional LFSR over $\mathbb{F}_{2^m}$ should take the structure of finite field into account and thus is more complicated. The following result shows that the implementation of σ-LFSR over $\mathbb{F}_{2^m}$ only need consider CPU instructions, but not the structure of finite field.

Theorem 5. Let $M_m(\mathbb{F}_2)$ be the $m \times m$ matrix ring over $\mathbb{F}_2$. Then $M_m(\mathbb{F}_2)$ is generated by σ and $\&_K$ over $\mathbb{F}_2$, where $K = (1,0,\cdots,0)$, that is $M_m(\mathbb{F}_2) = \mathbb{F}_2[\sigma, \&_K]$.

Proof. Let $E_{i,j}$ ($i=1,2,\cdots,m$, $j=1,2,\cdots,m$) be the basis of $M_m(\mathbb{F}_2)$, where $E_{i,j}$ is $m\times m$ matrix over $\mathbb{F}_2$ with 1 at ith row and jth column, others are 0. Under the normal basis $\alpha, \alpha^2, \cdots, \alpha^{2^{m-1}}$ of $\mathbb{F}_{2^m}/\mathbb{F}_2$, the associated matrix of circular rotation σ and AND operation $\&_K$ are

$$\begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}. \tag{4}$$

Notice that $\sigma^i \&_K = E_{i,1}$, hence we have $E_{i,j} = \sigma^i \&_K \sigma^{m+1-j}$. Thus, $M_m(\mathbb{F}_2) = \mathbb{F}_2[\sigma, \&_K]$.

This theorem shows every σ-LFSR over finite field with characteristic 2 can be implemented only by circular operation, AND operation, and XOR operation. However, it should be noticed that σ and $\&_K$ are non-commutative so that the expressions of σ-polynomial with σ and $\&_K$ are probably very complicated. Therefore, it is interesting to construct simple σ-LFSR with maximal period. Let us consider the σ-LFSR over $\mathbb{F}_{2^m}$ with trinomial σ-polynomial. For the sake of fast software implementation, we discuss the σ-polynomial with coefficients circular rotation or AND operation.

Now, we investigate the σ-polynomials as follows:

$$f(x) = x^n + \&_V x^r + \sigma^k, \text{ where } k,\ r \in \mathbb{Z}^+,\ 1 \leqslant k \leqslant m,\ 1 \leqslant r < n,\ 0 \neq V \in \mathbb{F}_2^m. \tag{5}$$

Example 1. Figure 2 shows trinomial σ-LFSR over $\mathbb{F}_{2^2}$ with σ-polynomial $f(x) = x^{14} + \&_{10}x^{11} + \sigma$.

In Figure 2 $\&_{10}$ is AND operation with vector (1,0). Let $F(x)$ be the corresponding matrix polynomial of $f(x)$ under normal basis. Then

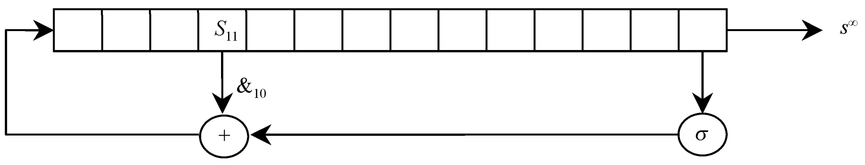

Figure 2 Example of trinomial σ-LFSR.

$$F(x)=\begin{pmatrix} x^{14}+x^{11} & 1 \\ 1 & x^{14} \end{pmatrix} \text{ and } |F(x)|=x^{28}+x^{25}+1\in\mathbb{F}_2[x].$$

$|F(x)|$ is precisely a primitive polynomial over $\mathbb{F}_2$. $f(x)$ is primitive σ-polynomial by Theorem 3, and the period of s^{∞} is $2^{28}-1$. From the point of software implementation, this trinomial σ-LFSR only required three basic operations: circular rotation, AND operation, and XOR operation. Because these operations are all fundamental instructions of CPU, the software efficiency of this example is extremely high.

Next, we discuss the general cases. In order to obtain maximal period sequence, one has to compute the determinant of the corresponding matrix polynomial $F(x)$ of σ-polynomial (5). We use $C_{m,k}$ to denote the following matrix over $\mathbb{F}_2$, where A is the matrix of σ c.f (3).

$$C_{m,k}=\begin{pmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_m \end{pmatrix}+A^k.$$

We compute the determinant $|F(x)|$ by using $x_i=x^n+a_ix^r$ $(1\leqslant i\leqslant m)$ in $C_{m,k}$. For convenience, we give following.

Lemma 1. If $k|m$, then $|C_{m,k}|=\prod_{i=1}^{k}(A_i+1)$ where $A_i=\prod_{t\equiv i \bmod k} x_t$.

Proof. Suppose $m=lk$. We partition $C_{m,k}$ into $k\times k$ submatrixes as follows.

$$C_{m,k}=\left(\begin{array}{ccc|ccc|c|ccc}
x_1 & & & 0 & & & & 1 & & \\
 & \ddots & & & \ddots & & & & \ddots & \\
 & & x_k & & & 0 & & & & 1 \\
\hline
 & & & x_{k+1} & & & & 0 & & \\
 & \ddots & & & \ddots & & & & \ddots & \\
 & & & & & x_{2k} & & & & 0 \\
\hline
 & & & & & & \ddots & & & \\
\hline
0 & & & 0 & & & & x_{m-k+1} & & \\
 & \ddots & & & \ddots & & & & \ddots & \\
 & & 0 & & & 0 & & & & x_m
\end{array}\right)$$

$$=\begin{pmatrix} X_1 & 0 & 0 & \cdots & 0 & 0 & E_k \\ E_k & X_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & E_k & X_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & X_{l-2} & 0 & 0 \\ 0 & 0 & 0 & \cdots & E_k & X_{l-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & E_k & X_l \end{pmatrix},$$

where X_i=diag($x_{(i-1)k+1},\cdots,x_{ik}$) is diagonal matrix, and E_k is the $k\times k$ identity matrix. To compute the determinant of $C_{m,k}$, we define the following procedure illustrated by X_1.

Procedure 1.

I. Multiply X_1 with the 2nd submatrix row (E_k, $X_2,\cdots$,0) and add it to the 1st submatrix row (X_1, 0,$\cdots$, E_k).

II. Compute determinant according to E_k at the 2nd submatrix row and the 1st submatrix column using Lapalace theorem and obtain a $(m-k)\times(m-k)$ determinant.

In fact, Procedure 1 is as follows:

$$\begin{vmatrix} X_1 & 0 & \cdots & E_k \\ E_k & X_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X_l \end{vmatrix}_{m\times m} \overset{\text{(I)}}{=\!=} \begin{vmatrix} 0 & X_1X_2 & \cdots & E_k \\ E_k & X_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X_l \end{vmatrix}_{m\times m} \overset{\text{(II)}}{=\!=} \begin{vmatrix} X_1X_2 & 0 & \cdots & E_k \\ E_k & X_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X_l \end{vmatrix}_{(m-k)\times(m-k)}.$$

Repeatedly apply Procedure 1 to the top-left submatrix until $C_{m,k}$ comes to a diagonal matrix.

$$|C_{m,k}|=\begin{vmatrix} X_1X_2 & 0 & \cdots & E_k \\ E_k & X_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X_l \end{vmatrix}=\begin{vmatrix} X_1X_2X_3 & 0 & \cdots & E_k \\ E_k & X_4 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X_l \end{vmatrix}=\cdots=\begin{vmatrix} \prod_{i=1}^{l-1} X_i & E_k \\ E_k & X_l \end{vmatrix}=|\prod_{i=1}^{l} X_i + E_k|.$$

Notice the diagonal elements of $\prod_{i=1}^{l} X_i + E_k$ is A_i+1, where $A_i=\prod_{t\equiv i \bmod k} x_t$, so we finish the proof.

Theorem 6. If d is the greatest common divisor of m and k, then $|C_{m,k}|=\prod_{i=1}^{d}(\prod_{t\equiv i \bmod d} x_t+1)$.

Proof. If $k|m$, the conclusion holds by Lemma 1. Now, we consider the cases when m is not divided by k and suppose $m=ku_1-k_1$, where $u_1\in\mathbb{Z}^+, k_1\in\mathbb{N}$ and $0\leqslant k_1<k$. We partition $C_{m,k}$ into the following form by $k\times k$ submatrix

$$|C_{m,k}|=\left|\begin{array}{cccc:c} X_1 & 0 & \cdots & T_k & Y'_{k,k-k_1} \\ E_k & X_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & X_{u_1-1} & 0 \\ \hdashline 0 & 0 & \cdots & Y_{k-k_1,k} & R_k \end{array}\right|, \tag{6}$$

where X_i, R_k are $k\times k$ and $(k-k_1)\times(k-k_1)$ diagonal square, respectively. $X_i=\mathrm{diag}(x_{(i-1)k+1},\cdots,x_{ik})$, $R_k=\mathrm{diag}(x_{(l_1-1)k+1},\ldots,x_m)$, E_k is $k\times k$ identity matrix, and $Y_{k-k_1,k}, Y'_{k,k-k_1}, T_k$ is as follows:

$$(y_{i,j})=\begin{cases}1, & i=j,\\ 0, & i\neq j.\end{cases}\quad (y'_{i,j})=\begin{cases}1, & i=k_1+j, i>k_1,\\ 0, & \text{otherwise.}\end{cases}\quad (t_{i,j})=\begin{cases}1, & i=k-k_1+i, j>k-k_1\\ 0, & \text{otherwise.}\end{cases}$$

By Lemma 1 and (6), we have

$$\left|C_{m,k}\right|=\begin{vmatrix}X_1X_2\cdots X_{u_1-1}+T_k & Y'_{k,k-k_1}\\ Y_{k-k_1,k} & R_k\end{vmatrix}.$$

Above equation can be repartitioned as follows, where $s_i=x_ix_{k+i}\cdots x_{(u_1-2)k+i}$, $(1\leqslant i<k)$:

$$\left(\begin{array}{ccc:cccc}
s_1 & & 0 & 1 & & & \\
 & \ddots & & & \ddots & & \\
 & & s_{k-k_1} & & & & \\
 & & & s_{k-k_1+1} & & & \\
 & & & & \ddots & & \ddots \\
 & & & & & s_k & 1 \\ \hdashline
1 & & & & x_{(u_1-1)k+1} & & \\
 & \ddots & & & & \ddots & \\
 & & 1 & & & & x_m
\end{array}\right)\triangleq\begin{pmatrix}\overline{S}_{k-k_1} & E_k+\overline{Q}_{k_1}\\ E_{k-k_1} & \overline{R}_k\end{pmatrix}, \qquad (7)$$

where $S_{k-k_1}=\mathrm{diag}(s_1,\cdots,s_{k-k_1})$, $\overline{S}_{k-k_1}=(S_{k-k_1},0)^T$, $\overline{R}_k=(0,R_k)$ and symbol T is the transpose of the matrix. $\overline{Q}_{k_1}$ is the following matrix:

$$\overline{Q}_{k_1}=\begin{pmatrix}0 & & & \\ \vdots & \ddots & & \\ s_{k-k_1+1} & & 0 & \\ & \ddots & & \ddots \\ 0 & & s_k & 0\end{pmatrix}. \qquad (8)$$

To compute determinant of (7), we give the following procedure illustrated by S_{k-k_1}.

Procedure 2.

I. Multiply the submatrix composed of the last $k-k_1$ rows in (7) with S_{k-k_1} and add it to the submatrix composed of the rows in (7), where S_{k-k_1} stands.

II. Compute the resulted (7) by step I according to left-bottom block E_{k-k_1}, using Lapalace theorem.

III. Adjust the column sequence of determinant by moving the first k_1 columns to the end.

The details of Procedure 2 are as follows.

$$\begin{vmatrix} s_1 & & 1 & & & & & & \\ & \ddots & & \ddots & & & & & \\ & & s_l & & & & & & \\ & & & s_{l+1} & & & & & \\ & & & & \ddots & & \ddots & & \\ & & & & & s_k & & 1 & \\ 1 & & & & & & x_{(u_1-1)k+1} & & \\ & \ddots & & & & & & \ddots & \\ & & 1 & & & & & & x_m \end{vmatrix} \overset{\text{(I)}}{=} \begin{vmatrix} 0 & & 1 & & & M_1 & & \\ & \ddots & & \ddots & & & \ddots & \\ & & 0 & & & & & M_l \\ & & & s_{l+1} & & & & \\ & & & & \ddots & & \ddots & \\ & & & & & s_k & & 1 \\ 1 & & & & & & x_{(u_1-1)k+1} & \\ & \ddots & & & & & & \ddots \\ & & 1 & & & & & x_m \end{vmatrix}$$

$$\overset{\text{(II)}}{=} \begin{vmatrix} 1 & & M_1 & \\ & \ddots & & \ddots \\ s_{l+1} & & & M_l \\ & \ddots & & \ddots \\ & & s_k & 1 \end{vmatrix} \overset{\text{(III)}}{=} \begin{vmatrix} M_1 & & & 1 & & \\ & \ddots & & & \ddots & \\ & & M_{k_1} & & & 1 \\ 1 & & & M_{k_1+1} & & \\ & \ddots & & & \ddots & \\ & & 1 & & & M_k \end{vmatrix}, \tag{9}$$

where $l = k - k_1$, $M_i = \prod_{t_1 \equiv i \bmod k} x_{t_1}$. We observe that the right equality in (9) is a $k \times k$ matrix similar to C_{k,k_1} except the diagonal elements $(M_1, M_2, \ldots, M_k)$.

After $u_1 - 2$ times Procedure 1 and one Procedure 2, $C_{m,k}$ is converted to (9) whose form is similar to C_{k,k_1}, and we call this procedure a recursive transformation. Suppose $d = g.c.d(m,k)$, u_i is positive integer, and $1 \leqslant k_i < k_{i-1}$, $(1 \leqslant i \leqslant n-1)$, $k_0 = k$. Thus, we have

$$\begin{cases} m = u_1 k - k_1, \\ k = u_2 k_1 - k_2, \\ \quad \vdots \\ k_{n-2} = u_n k_{n-1} - d, \\ k_{n-1} = u_{n+1} d. \end{cases} \tag{10}$$

After one recursive transformation, $C_{m,k}$ is converted to C_{k,k_1} with diagonal elements $M_i = \prod_{t_1 \equiv i \bmod k} x_{t_1}$; after two recursive transformation, the matrix is similar to C_{k_1,k_2} with diagonal elements $D_i = \prod_{t_2 \equiv i \bmod k_1} (\prod_{t_1 \equiv t_2 \bmod k} x_{t_1})$, where $1 \leqslant i \leqslant k_1$. Repeatedly applying recursive transformation for n time, we can obtain a matrix similar to $C_{k_{n-1},d}$ with diagonal elements $K_i = \prod\limits_{t_n \equiv i \bmod k_{n-1}} \cdots \prod\limits_{t_1 \equiv t_2 \bmod k} x_{t_1}$, where $1 \leqslant i \leqslant k_{n-1}$. With the fact that $d | k_{n-1}$ and Lemma 1, we have

$$\left| C_{m,k} \right| = \prod_{i=1}^{d} \Big(\prod_{t \equiv i \bmod d} K_t + 1 \Big). \tag{11}$$

For $d = g.c.d(m,k)$, we see that $d | k_{n-1}$, $d | k_{n-2}, \cdots, d | k$ by eq. (10). Hence,

$$\prod_{t\equiv i \bmod d} K_t = \prod_{t\equiv i \bmod d} \Big(\prod_{t_n \equiv t \bmod k_{n-1}} \cdots \prod_{t_1 \equiv t_2 \bmod k} x_{t_1} \Big) = \prod_{t\equiv i \bmod d} x_t . \tag{12}$$

Combining eqs. (11) and (12), we prove $|C_{m,k}| = \prod_{i=1}^{d} \Big(\prod_{t\equiv i \bmod d} x_t + 1 \Big)$.

Corollary 1. Let $d=g.c.d(m,k)$, $f(x) = x^n + \&_V x^r + \sigma^k$ be a σ-polynomial, where k,m are integers and $0 \leqslant k \leqslant m-1$, $V=(a_1,a_2,\cdots,a_m) \in \mathbb{F}_{2^m}$. Let s be Hamming weight of vector V. Then

A. If $d\neq 1$, $f(x)$ is not a primitive.

B. If $d=1$, n and r are both even, $f(x)$ is not a primitive.

C. If $d=1$, n and r are not both even, but s and m are both even, $f(x)$ is not primitive.

Proof. A. If $d\neq 1$, then $d\geqslant 2$. By Theorem 6, it is obvious that $f(x)$ is reducible, so $f(x)$ is not primitive.

B. If $d=1$, by Theorem 6, $F(x) = \prod_{i=1}^{m} (x^n + a_i x^r) + 1 = (x^n)^{m-s}(x^n + x^r)^s + 1$. If $n = 2n'$ and $r = 2r'$, then $F(x) = (x^{n'(m-s)}(x^{n'} + x^{r'})^s + 1)^2$ is a square, so $f(x)$ is not primitive.

C. If $s = 2s'$ and $m = 2m'$, then $F(x) = (x^{n(m'-s')}(x^n + x^r)^{s'} + 1)^2$ is also a square, so $f(x)$ is not primitive.

4 Searching for primitive trinomial σ-LFSR

According to the analysis of trinomial σ-LFSR, here we describe an algorithm to summarize the above conclusions.

Algorithm 1 (Searching for Primitive Trinomial σ-LFSR over $\mathbb{F}_{2^m}$).

```
1. Choose degree of circular rotation $k$ such that $g.c.d(m,k)=1$.
2. Choose $n$ and $r$ such that not both are even.
3. Choose $s$ such that not both $m$ and $s$ are even.
4. Check whether $g(x)=(x^n)^{m-s}(x^n+x^r)+1$ is a primitive polynomial over binary field, if false go to step 1.
```

By Algorithm 1, we made a largely exhaustive search to find primitive trinomial, and Table 1 lists some primitive σ-trinomial with $nm\leqslant 100$. Although k has a different value, as long as k and m are coprime, the primitivity of σ-trinomial is the same, so k is omitted in Table 1. The 10th row of $\mathbb{F}_{2^2}$ in Table 1, 14 1 11 is Example 1. It can be seen from the table, for every positive m, there not always exists a primitive σ-trinomial of any degree. In particular, the number of primitive σ-trinomial in cases $m=4$ or 6 are apparently fewer than other cases. Further explanation for this phenomenon would be of interest. For σ-pentanomial $f(x) = x^n + a_1 x^{j_1} + a_2 x^{j_2} + a_3 x^{j_3} + \sigma^k$, where $n>j_1>j_2>j_3>0$, and the coefficient a_i is either $\&_V$ ($V \in \mathbb{F}_{2^m}$) or a power of σ. We also simulate in the computer and list the results in Table 2. Similar to the open problem on primitive pentanomial of any degree over finite fields, it is interesting to ask for the solution about primitive σ-pentanomial.

Table 1 Primitive σ-trinomial $f(x)=x^n+\&_V x^r+\sigma^k$ over $\mathbb{F}_{2^m}$ a)

n	s	r	n	s	r	n	s	r	n	s	r	n	s	r	n	s	r
	$\mathbb{F}_{2^2}$		34	1	1	13	1	5	28	1	15	4	1	1	14	1	1
2	1	1	34	1	25	13	2	6	29	1	16	5	1	2		$\mathbb{F}_{2^7}$	
3	1	2	42	1	29	13	1	9	31	1	29	5	3	4	3	1	1
5	1	2	47	1	26	13	2	9	31	2	30	7	1	5	3	2	2
9	1	2	49	1	22	13	2	11		$\mathbb{F}_{2^4}$		7	2	6	4	1	1
10	1	7	49	1	38	19	2	8	5	1	2	11	3	3	5	1	3
11	1	10		$\mathbb{F}_{2^3}$		19	1	12	7	1	4	11	4	5	5	2	4
14	1	1	2	1	1	20	1	9	13	1	10	12	1	1	7	2	1
14	1	5	3	2	1	20	1	19	15	1	4	12	1	11	7	3	2
14	1	11	5	1	1	21	2	5	15	1	14	13	3	2	7	3	3
18	1	7	5	2	1	21	1	16	17	1	8	13	2	4	7	3	4
26	1	5	5	2	3	21	1	20	21	1	8	13	4	5	7	4	4
26	1	7	5	1	4	27	2	4		$\mathbb{F}_{2^5}$		13	3	7	7	5	4
26	1	23	7	1	5	27	1	11	3	2	1	19	1	2	7	6	5
29	1	10	7	2	6	27	2	19	3	4	1	19	1	8	9	4	1
30	1	19	11	2	1	27	1	23	3	1	2		$\mathbb{F}_{2^6}$		9	1	4
30	1	29	12	1	1	27	2	25	3	4	2	10	1	9	9	1	8

a) $2\leqslant m\leqslant 7$, k and m are coprime satisfying $nm\leqslant 100$, $1\leqslant r<n$, $1\leqslant s\leqslant m$.

Table 2 Primitive σ-pentanomial $f(x)=x^n+a_1x^{j_1}+a_2x^{j_2}+a_3x^{j_3}+\sigma^k$ over $\mathbb{F}_{2^m}$ a)

	n	a_1	j_1	a_2	j_2	a_3	j_3	k	n	a_1	j_1	a_2	j_2	a_3	j_3	k
	4	3	3	1	2	3	1	(1)	4	3	3	(2)	2	1	1	(1)
	5	3	3	3	2	1	1	(1)	5	1	4	(1)	3	1	2	(1)
$\mathbb{F}_{2^2}$	6	1	3	1	2	3	1	(1)	6	1	5	(1)	3	1	2	(2)
	7	1	3	1	2	3	1	(1)	7	1	5	(2)	3	3	2	(1)
	8	1	3	1	2	3	1	(1)	8	1	7	(1)	5	1	4	(2)
	4	3	3	1	2	7	1	(1)	4	1	3	(3)	2	7	1	(2)
	5	7	3	3	2	7	1	(2)	5	1	4	(1)	2	7	1	(2)
$\mathbb{F}_{2^3}$	6	7	3	1	2	1	1	(1)	6	1	5	(1)	2	7	1	(3)
	7	3	4	7	2	1	1	(2)	7	7	4	(3)	3	1	1	(1)
	8	1	6	3	5	1	2	(1)	8	1	5	(3)	2	1	1	(2)
	4	1	3	3	2	f	1	(3)	4	f	3	(1)	2	7	1	(4)
	5	7	4	1	3	7	1	(3)	5	7	4	(4)	2	3	1	(3)
$\mathbb{F}_{2^4}$	6	7	4	3	3	7	1	(1)	6	1	4	(1)	3	f	1	(2)
	7	3	4	1	2	1	1	(3)	7	1	4	(4)	3	1	2	(3)
	8	7	6	f	5	f	2	(1)	8	7	5	(1)	4	f	3	(4)
	4	1	3	7	2	3	1	(1)	4	3	3	(3)	2	1	1	(4)
	5	1	3	f	2	3	1	(2)	5	3	3	(5)	2	f	1	(2)
$\mathbb{F}_{2^5}$	6	1	3	3	2	3	1	(4)	6	1	4	(2)	2	7	1	(4)
	7	1	3	1	2	7	1	(3)	7	7	4	(2)	2	3	1	(1)
	8	1f	3	7	2	1	1	(1)	8	7	4	(1)	2	1f	1	(3)
	4	1	3	3	2	1f	1	(1)	4	1	3	(1)	2	3f	1	(2)
	5	1	3	7	2	3f	1	(5)	5	3f	3	(3)	2	1f	1	(4)
$\mathbb{F}_{2^6}$	6	3	3	7	2	3f	1	(5)	6	f	3	(6)	2	1	1	(1)
	7	1	3	7	2	7	1	(1)	7	1	4	(3)	2	1f	1	(5)
	8	7	6	1f	5	1f	3	(1)	8	7	7	(4)	3	3f	2	(1)

a) $n>j_1>j_2>j_3>0$, ( ) denotes power of σ, other coefficient is AND item in hexadecimal representation.

5 Conclusions

In this paper, we introduce the concept of σ-LFSR composed of CPU instructions, such as circular rotation, AND, XOR, etc. Some basic cryptographic properties are presented, and in particular, a type of trinomial σ-LFSR is found which can be implemented only by a few computer basic instructions circular rotation and AND operation. Moreover, this type of trinomial σ-LFSR not only has desirable cryptographic properties, but also has high software efficiency. We believe that as we move from 32 to 64 bit processor, word-based σ-LFSR, which can be implemented with few instructions and high efficiency, will become an attractive alternative to stream cipher design.

1 Golomb S W. Shift Register Sequences. San Francisco: Holden-Day, 1967

2 Lidi R, Niederreiter H. Finite Fields. In: Encyclopedia of Mathematics and its Applications 20. Cambridge: Cambridge University Press, 1983

3 Preneel B. Introduction to the Proceedings of the Fast Software Encryption 1994 Workshop. In: LNCS, Vol. 1008. Berlin, Heiderberg: Springer-Verlag, 1995. 1－5

4 Zhang M, Carroll C, Chan A. The Software-Oriented Stream Cipher SSC2. Fast Software Encryption 2000 Workshop. In: LNCS, Vol. 1978. Berlin, Heiderberg: Springer-Verlag, 2001. 31－48

5 Daemen J, Craig S, Clapp K. Fast Hashing and Stream Encryption with PANAMA. Fast Software Encryption 1998 Workshop. In: LNCS, Vol. 1372. Berlin, Heiderberg: Springer-Verlag, 1999. 60－74

6 Watanabe D, Furuya S, Yoshida H, et al. A New Keystream Generator MUGI. Fast Software Encryption 2002 Workshop. In: LNCS, Vol. 2365. Berlin, Heiderberg: Springer-Verlag, 2003. 179－194

7 Rogaway P, Coppersmith D. A software-optimized encryption algorithm. Fast Software Encryption 1993 Workshop. In: LNCS, Vol. 809. Berlin, Heiderberg: Springer-Verlag, 1994. 53－63

8 Halevi S, Coppersmith D, Charanjit S. Jutla. Scream: A Software-Efficient Stream Cipher. Fast Software Encryption 2002 Workshop. In: LNCS, Vol 2365. Berlin, Heiderberg: Springer-Verlag, 2003. 195－209

9 Boesgaard M, Vesterager M, Pedersen T, et al. Rabbit: A New High-Performance Stream Cipher. Fast Software Encryption 2003 Workshop. In: LNCS, Vol. 2887. Berlin, Heiderberg: Springer-Verlag, 2004. 307－329

10 Ferguson N, Whiting D, Schneier B, et al. Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive. Fast Software Encryption 2003 Workshop. In: LNCS, Vol. 2887. Berlin, Heiderberg: Springer-Verlag, 2004. 330－346

11 Hawkes P, Rose G. Primitive Specification and Supporting Documentation for SOBER-t16 Submission to NESSIE, Proceedings of the first NESSIE Workshop, Heverlee, Belgium, 2000

12 Hawkes P, Rose G. Primitive Specification and Supporting Documentation for SOBER-t32 Submission to NESSIE, Proceedings of the first NESSIE Workshop, Heverlee, Belgium, 2000

13 Hawkes P, Rose G. Turing: A Fast Stream Cipher. Fast Software Encryption 2003 Workshop. In: Johansson T, ed. LNCS, Vol. 2887. Berlin, Heiderberg: Springer-Verlag, 2003. 290－306

14 Ekdahl P, Johansson T. SNOW──a new stream cipher. In: Proceedings of the first NESSIE Workshop, Heverlee, Belgium, 2000

15 Ekdahl P, Johansson T. A New Version of the Stream Cipher SNOW. Selected Areas in Cryptography 2002 Workshop. In: Nyberg K, Heys H, eds. LNCS, Vol. 2595. Berlin, Heidelberg: Springer-Verlag, 2003. 47－61

16 Tsaban B, Vishne U. Efficient linear feedback shift registers with maximal period. Finite Fields Their Appl, 2002, 8: 256－267

17 Dewar M, Panario D. Linear transformation shift registers. IEEE Trans Infor Theory, 2003, 49: 2047－2052

18 Zeng G, Han W B, He K C. High efficiency feedback shift register: σ-LFSR. Cryptology ePrint Archive, Report 2007/114. 2007. http://eprint.iacr.org/