An Efficient Algorithm for Determining the k-error Linear Complexity of Binary Sequences with Periods $2p^n$

Shimin Wei

Huaibei Coal Normal University, College of Computer Science & Technique, Huaibei 235000, Anhui, China

Summary
An efficient algorithm is presented for computing the k-error linear complexity of a binary sequence with period $2p^n$, where 2 is a primitive root modulo p^2. The new algorithm is a generalization of an algorithm for computing the k-error linear complexity of a binary sequence with period p^n presented by Wei, Chen, and Xiao.
Key words:
Cryptography, binary sequence, linear complexity, k-*error linear complexity*.

1. Introduction

Although linear complexity is a necessary index for measuring the unpredictability of a sequence, it is not sufficient. The linear complexity has a typical unstable property as a fast change (increase or decrease) by only a few bit change within one period of the original sequence, hence it is cryptographically weak. Ding, Xiao and Shan [2] introduced some measure indexes on the security of stream ciphers. One of them is the sphere complexity of periodic sequences. Stamp and Martin [6] proposed a measure index analogous with the sphere complexity, the k-error linear complexity, and gave an algorithm for determining the k-error linear complexity of binary sequences with period 2^n. Kaida, Uehara and Imamura [4] generalized the algorithm to an algorithm for determining the k-error linear complexity of sequences over GF(p^m) with period p^n. This algorithm is also a generalization of the algorithm presented by Ding [2]. In this paper, we present an efficient algorithm for determining the k-error linear complexity of a binary sequence with period $2p^n$, where 2 is a primitive root modulo p^2. The new algorithm is a generalization of an algorithm presented by Xiao, Wei, Imamura and Lam [7].In this paper we will consider binary sequences.

Let $s=(s_0, s_1, s_2, \cdots)$ be a binary sequence. s is called an L-order linear recusive sequence if there exists a positive integer L and $c_1, c_2, \cdots, c_L$ in GF(2) such that s satisfies $s_j+c_1s_{j-1}+\cdots+c_Ls_{j-L}=0$ for any $j\geq L$; and the minimal order is called the linear complexity of s, and denoted by $c(s)$. If there exists a positive number N such that $s_i=s_{i+N}$ for $i=1, 2, \cdots$, s is called a periodic sequence, and N is called a period of s. The generating function of s is defined as

$$s(x)=s_0+s_1x+s_2x^2+\cdots.$$

Let s be a binary sequence with the first period $s^N=(s_0, s_1, \cdots, s_{N-1})$. Then

$$s(x)=\frac{s^N(x)}{1-x^N}=\frac{s^N(x)/\gcd(s^N(x),1-x^N)}{(1-x^N)/\gcd(s^N(x),1-x^N)}=\frac{g(x)}{f_s(x)},$$

Where

$$f_s(x)=(1-x^N)/\gcd(s^N(x), 1-x^N),$$
$$g(x)= s^N(x)/\gcd(s^N(x), 1-x^N).$$

Obviously, $\gcd(g(x), f_s(x))=1$, $\deg g(x)<\deg f_s(x)$, $f_s(x)$ is the minimal polynomial of s, and $\deg f_s(x)=c(s)$ [3].

2. An Algorithm for Computing the Linear Complexity

Let us recall some results in finite field theory and number theory. Let p be a prime. Then $\phi(p^n)=p^n-p^{n-1}$, where n is a positive integer, ϕ is the Euler ϕ-function. Let $\Phi_n(x)$ be the n-th cyclotomic polynomial. Then $\Phi_n(x)$ is irreducible over GF(2) if and only if 2 is a primitive root modulo n, i.e. if 2 has order $\phi(n)$ modulo n.

From Theorem 3.1 in [8] we have the following theorem.

Theorem 1. Let s be a binary sequence with the first period $s^N=(s_0, s_1, \cdots, s_{N-1})$, and let 2 be a primitive root (mod p^2), and $N=2p^n$. Denote $l=p^{n-1}$, $A_i=(a_{(i-1)l}, a_{(i-1)l+1},\cdots, a_{il-1})$, $i=1, 2, \cdots, 2p$. Then

$$\gcd(s^N(x), 1+x^N)= \gcd(s^N(x), \Phi_{pl}(x)^2)\cdot\gcd(1+x^{2l}, (A_1(x)+A_3(x)+\cdots+A_{2p-1}(x))+(A_2(x)+A_4(x)+\cdots+A_{2p}(x))\, x^l),$$

and

1) $\Phi_{pl}(x)^2| s^N(x)$ if and only if
$$(A_1, A_2)=(A_3, A_4)=\cdots=(A_{2p-1}, A_{2p});$$
2) $\Phi_{pl}(x)| s^N(x)$ if and only if
$$A_1+A_{p+1}=A_2+A_{p+2}=\cdots=A_p+A_{2p}.$$

Let s be a binary sequence with period $N=2p^n$, 2 a primitive root (mod p^2), and $s^N=(s_0, s_1, \cdots, s_{N-1})$ the first period of s. From Algorithm 4.2 in [8] and Theorem 1 it immediately follows Algorithm 1.

Algorithm 1. Initial: $a \leftarrow s^N$, $l \leftarrow p^n$, $c \leftarrow 0, f \leftarrow 1$.

1) If $l=1$, go to 2); otherwise $l \leftarrow l/p$,
$A_i=(a_{(i-1)l}, a_{(i-1)l+1},\cdots, a_{il-1})$, $i=1, 2, \cdots, 2p$,
go to 4).
2) If $a=(0,0)$, stop; otherwise , go to 3).
3) If $a_0=a_1$, $c \leftarrow c+1$, $f \leftarrow (1+x)f$, otherwise $c \leftarrow c+2$, $f \leftarrow (1+x^2)f$, stop.
4) If $(A_1, A_2)= (A_3, A_4)= \cdots =(A_{2p-1}, A_{2p})$, $a \leftarrow (A_1,A_2)$, go to 1); otherwise go to 5).
5) If $A_1+A_{p+1}=A_2+A_{p+2}=\cdots=A_p+A_{2p}$, then
$a \leftarrow (A_1+A_2+\cdots+A_p, A_2+A_3+\cdots+A_{p+1})$,
$c \leftarrow c+(p-1)l, f \leftarrow f\Phi_{pl}(x)$,
go to 1); otherwise
$a \leftarrow (A_1+A_3+\cdots+ A_{2p-1}, A_2+A_4+\cdots+A_{2p})$,
$c \leftarrow c+2(p-1)l$, $f \leftarrow f\Phi_{pl}(x)^2$,
go to 1).
Finally, we have that the linear complexity $c(s)=c$ and the minimal polynomial $f_s(x)=f$ of s.

3. The New Algorithm

Let $s=(s_0, s_1, s_2, \cdots)$ be a binary sequence with period N. The smallest linear complexity that can be obtained when any k ($0 \le k \le N$) or fewer of the s_i's are altered in every period of s is called the k-error linear complexity of s [6], and denoted by $c_k(s)$, i.e. $c_k(s)=\min_{w(t)\le k}\{c(s+t)\}$, where t is a binary sequence with period N, $w(t)$ is the number of non-zero elements of the first period of t, $c(s)$ is the linear complexity of s. The k-error linear complexity of any sequence could be found by repeated application of the Berlekamp-Massey [5] Algorithm. But, to compute the k-error linear complexity of a binary sequence with period N would require $\sum_{j=1}^{k}\binom{N}{j}$ applications of the Berlekamp-Massey Algorithm. In the case of $N=2p^m$, 2 is a primitive root modulo p^2, to compute the k-error linear complexity of s would require $\sum_{j=1}^{k}\binom{N}{j}$ applications of Algorithm 1. The number $\sum_{j=1}^{k}\binom{N}{j}$ becomes very large even for moderate N and k.
In this section, we propose and prove an efficient algorithm for determining the k-error linear complexity of binary sequences with period $N=2p^n$, where 2 is a primitive root modulo p^2. The new algorithm is a generalization of Algorithm 1. Let $s=(s_0, s_1, s_2, \cdots)$ be a binary sequence with period $N=2p^n$, and let 2 be a primitive root modulo p^2, and $s^N=(s_0, s_1, \cdots, s_{N-1})$ the first period of s. An efficient algorithm for computing the k-error linear complexity of s is as follows.

Algorithm 2 Initial: $a \leftarrow s^N$, $l \leftarrow p^n$, $c \leftarrow 0$, $\text{cost}[i] \leftarrow 1$, $i=0, 1, 2, \cdots, 2l-1$, $K \leftarrow k$.

1) If $l=1$, then $T \leftarrow a_0\text{cost}[0]+ a_1\text{cost}[1]$, go to 2); otherwise,
$l \leftarrow l/p$, $A_i \leftarrow (a_{(i-1)l}, a_{(i-1)l+1},\cdots, a_{il-1})$, $i=1, 2, \cdots, 2p$,
$T_{i1} \leftarrow a_i\text{cost}[i]+a_{i+l}\text{cost}[i+l]+\cdots+a_{i+(p-1)l}\text{cost}[i+(p-1)l]$,
$T_{i0} \leftarrow \text{cost}[i]+ \text{cost}[i+l]+\cdots+\text{cost}[i+(p-1)l] -T_{i1}$,
$T_i \leftarrow \min\{T_{i1}, T_{i0}\}$, $T \leftarrow T_1+T_2+\cdots+T_{l-1}$,
go to 4).
2) If $T\le K$, stop; otherwise $T= (1+a_0)\text{cost}[0]+(1+a_1)\text{cost}[1]$, go to 3).
3) If $T\le K$, $c \leftarrow c+1$, stop; otherwise $c \leftarrow c+2$, stop.
4) If $T\le K$, $K \leftarrow K-T$, $\text{cost}[i] \leftarrow \max\{T_{i1}, T_{i0}\}-T_i$, $i=0, 1, \cdots, 2l-1$, go to 5); otherwise,
$B \leftarrow (A_1+A_{p+1}, A_2+A_{p+2}, \cdots, A_p+A_{2p})$,
$\text{cost}'[i] \leftarrow \min\{\text{cost}[i], \text{cost}[i+pl]\}$, $i=0, 1, 2, \cdots, pl-1$,
$T_{i1}'=b_i\text{cost}'[i]+b_{i+l}\text{cost}'[i+l]+\cdots+b_{i+(p-1)l}\text{cost}'[i+(p-1)l]$
$T_{i0}'=\text{cost}'[i]+ \text{cost}'[i+l]+\cdots+\text{cost}'[i+(p-1)l]-T_{i1}'$,
$T_i'=\min\{T_{i1}', T_{i0}'\}$, $T'=T_1'+T_2'+\cdots+T_{l-1}'$,
go to 6).
5) For $i=0, 1, 2, \cdots, 2l-1$, if $T_i = T_{ih}$, then $a_i \leftarrow h$, $h=0, 1$, $a \leftarrow (A_1, A_2)$, go to 1).
6) If $T' \le K$, $K \leftarrow K-T'$, $c \leftarrow c+(p-1)l$; for $i=0, 1, \cdots, pl-1$, do
$\delta(i) \leftarrow 1$ if $T_{i1}' \le T_{i0}'$ and $a_i+a_{i+pl}=1$, or $T_{i1}'>T_{i0}'$ and $a_i+a_{i+pl}=0$;
$\delta(i) \leftarrow 0$ if if $T_{i1}' \le T_{i0}'$ and $a_i+a_{i+pl}=0$, or $T_{i1}'>T_{i0}'$ and $a_i+a_{i+pl}=1$;
go to 7); otherwise, $c \leftarrow c+2(p-1)l$,
$\text{cos}t[i] \leftarrow \min_{0\le j\le p-1}\{\text{cos}t[i+2jl]\}$, $i=0,1, \cdots, 2l-1$,
$a \leftarrow (A_1+A_3+\cdots+A_{2p-1},A_2+A_4+\cdots+A_{2p})$,
go to 1).
7) Do
$a_i \leftarrow a_i +1$ if $\delta(i)=1$ and $\text{cost}[i] \le \text{cost}[i+pl]$,
$a_{i+pl} \leftarrow a_{i+pl} +1$ if $\delta(i)=1$ and $\text{cost}[i+pl] \le \text{cost}[i]$,
$\text{cos}t[i] \leftarrow \min_{0\le j\le p-1}\{|\text{cos}t[i+jl]+(-1)^{\delta(i+jl)}\text{cos}t[i+jl+pl]|\}$,
for $i=0, 1,\cdots, 2l-1$,
$a \leftarrow (A_1+A_2+\cdots+ A_p, A_2+A_3+\cdots+A_{p+1})$,
go to 1).
Finally, the k-error linear complexity $c_k(s)$ of s is equal to c.
In Algorithm 2, $\text{cost}[i]$ ($\text{cost}'[i]$) is the minimal number of changes in the initial sequence s^N necessary and sufficient for changing the current element a_i (b_i) without disturbing the results
$(A_1, A_2)= (A_3, A_4)= \cdots =(A_{2p-1}, A_{2p})$ and
$A_1+A_{p+1}=A_2+A_{p+2}=\cdots= A_p+A_{2p}$
of any previous step.

Theorem 2 Let $s=(s_0, s_1, s_2, \cdots)$ be a binary sequence with period $N=2p^n$, 2 a primitive root modulo p^2, and $0\le k\le 2p^n$. Then Algorithm 2 computes c, the k-error linear complexity of s, in n steps.

Proof: Obviously, when $k=0$, Algorithm 2 reduces to Algorithm 1. If $k>0$ then we are allowed to make k (or fewer) bit changes in s^N in order to reduce the complexity c as much as possible. As with Algorithm 1, c only increases when $(A_1, A_2) = (A_3, A_4) = \cdots = (A_{2p-1}, A_{2p})$ doesn't hold. Therefore, if we can force $(A_1,A_2)=(A_3,A_4)=\cdots=(A_{2p-1},A_{2p})$ at the mth step of Algorithm 2 we should do so, since this prevent $2(p-1)p^{n-m}$ from being added to c, and the total of all remaining possible additions is only $2p^{n-m}$.

Now, suppose that at some step m, the value of cost[i] correctly gives the cost of changing a_i. If $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ doesn't hold, changing all 1 or all 0 in $\{a_{i+2(j-1)l}, j=1, 2, \cdots, p\}$ will force $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$, and hence the cost of forcing $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ is the minimum $T_i=\min\{T_{i1}, T_{i0}\}$ of the cost

$$T_{i1}=a_i\text{cost}[i]+a_{i+2l}\text{cost}[i+2l]+\cdots+a_{i+2(p-1)l}\text{cost}[i+2(p-1)l]$$

of changing all 1 and the cost

$$T_{i0}=\text{cost}[i]+ \text{cost}[i+2l]+\cdots+\text{cost}[i+2(p-1)l] -T_{i1}$$

of changing all 0. Therefore, the variable

$$T=T_1+T_2+\cdots+T_{l-1}$$

in Algorithm 2 correctly gives the total cost of forcing

$$(A_1, A_2)= (A_3, A_4)= \cdots=(A_{2p-1}, A_{2p}).$$

Suppose that $T\le K$. (1) If $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ doesn't hold and $T_{i1}\ge T_{i0}$, then we change all 0 in $\{a_{i+2(j-1)l}, j=1,2,\cdots,p\}$ since it has the lower cost, thus forcing $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$. Notice that at the end of this step we will let $a\leftarrow(A_1, A_2)$. If we change a_i in step $m+1$ we have effectively restored 1 (in step m) to its previous value. In order to maintain $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ in step m, we must change all 1 in $\{a_{i+2(j-1)l}, j=1, 2, \cdots, p\}$ in step m, which has a net cost of $T_{i1}-T_{i0}=\max\{T_{i1},T_{i0}\}-T_i$, and hence cost[$i$] is computed correctly in this case. (2) If $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ doesn't hold and $T_{i1}<T_{i0}$, the discussion similar to (1) will show that cost[i] is computed correctly in this case. (3) If $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ holds, then $T_i=0$. Notice that at the end of this step we will let $a\leftarrow(A_1, A_2)$. If we change a_i in step $m+1$, in order to maintain $a_i=a_{i+2l}=\cdots=a_{i+2(p-1)l}$ in step m, we must change every $a_{i+2(j-1)l}$, $(j=1,2,\cdots,p)$ in step m, which has a net cost of,

$$\text{cost}[i]+\text{cost}[i+l]+\cdots+\text{cost}[i+(p-1)l]$$
$$=T_{i1}+T_{i0}=\max\{T_{i1}, T_{i0}\}-T_i,$$

and hence cost[i] is computed correctly in this case.

It remains to consider the case $T>K$. In this case we cannot force $(A_1, A_2)= (A_3, A_4)= \cdots=(A_{2p-1}, A_{2p})$. But if we can force $A_1+A_{p+1}=A_2+A_{p+2}=\cdots=A_p+A_{2p}$ at the mth step of Algorithm 2 we should do so, since this prevent $(p-1)p^{n-m}$ from being added to c, and the total of all remaining possible additions is only $2p^{n-m}$. Denote $B_j=A_j+A_{p+j}$, $j=1, 2, \cdots, p$. (1) If $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$ doesn't hold, changing all 1 or all 0 in $\{b_{i+(j-1)l}, j=1, 2, \cdots, p\}$ will force $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, and hence the cost of forcing $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$ is the minimum $T_i'=\min\{T_{i1}', T_{i0}'\}$ of the cost

$$T_{i1}'=b_i\text{cost}'[i]+b_{i+l}\text{cost}'[i+l]+\cdots+b_{i+(p-1)l}\text{cost}'[i+(p-1)l]$$

of changing all 1 and the cost

$$T_{i0}'=\text{cost}'[i]+ \text{cost}'[i+l]+\cdots+\text{cost}'[i+(p-1)l] -T_{i1}'$$

of changing all 0, where $b_i=a_i+a_{i+pl}$, hence the cost of changing b_i is as follows

$$\text{cost}'[i]=\min\{\text{cost}[i], \text{cost}[i+pl]\}, i=0, 1, 2, \cdots, pl-1.$$

Therefore, the variable $T'=T_1'+ T_2'+\cdots+T_{l-1}'$ in Algorithm 2 correctly gives the total cost of forcing

$$A_1+A_{p+1}=A_2+ A_{p+2}=\cdots=A_p+A_{2p}.$$

Define $\delta(i+jl)=1$ if b_{i+jl} is changed in forcing $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, otherwise $\delta(i+jl)=0$, for $j=0, 1, 2, \cdots, p-1$, $i=0, 1, 2, \cdots, pl-1$.

Now, suppose that $T'\le K$. If $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$ doesn't hold, then we change all 0 in $\{b_{i+(j-1)l}, j=1, 2, \cdots, p\}$ if $T_{i1}'\ge T_{i0}'$, or all 1 in $\{b_{i+(j-1)l}, j=1, 2, \cdots, p\}$ if $T_{i1}'<T_{i0}'$, since it has the lower cost, thus forcing $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$. Notice that at the end of this step we will let

$$a\leftarrow(A_1+A_2+\cdots+ A_p, A_2+A_3+\cdots+A_{p+1}).$$

If we change a_i in step $m+1$ we must change one of $a_i, a_{i+l}, \cdots, a_{i+(p-1)l}$ in step m. In order to maintain $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, i.e. $a_i+a_{i+pl}=a_{i+l}+a_{i+l+pl}=\cdots=a_{i+(p-1)l}+a_{i+(2p-1)l}$ in step m, changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$ must happen at the same time, $j=1, 2, \cdots, p$.

Suppose that $T_{i1}'\ge T_{i0}'$, we change all 0 in $\{b_{i+(j-1)l}= a_{i+(j-1)l}+a_{i+(j-1)l+pl}, j=1, 2, \cdots, p\}$ in step m. If $b_{i+(j-1)l}= a_{i+(j-1)l}+a_{i+(j-1)l+pl}=1$, then $b_{i+(j-1)l}$ isn't change in forcing $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, hence $\delta(i+jl)=0$. The cost of changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$ at the same time is

$$\text{cost}[i+jl]+\text{cost}[i+jl+pl]$$
$$=\left|\text{cost}[i+jl]+(-1)^{\delta(i+jl)}\text{cost}[i+jl+pl]\right|.$$

If $b_{i+(j-1)l}= a_{i+(j-1)l}+a_{i+(j-1)l+pl}=0$, then $b_{i+(j-1)l}$ is changed in forcing $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, $\delta(i+jl)=1$, and the cost of changing b_i is as follows $\text{cost}'[i]=\min\{\text{cost}[i], \text{cost}[i+pl]\}$. The net cost of changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$ at the same time is

$$\max\{\text{cost}[i+jl], \text{cost}[i+jl+pl]\} -\min\{\text{cost}[i+jl],$$
$$\text{cost}[i+jl+pl]\}=\left|\text{cost}[i+jl]+(-1)^{\delta(i+jl)}\text{cost}[i+jl+pl]\right|.$$

Therefore, the cost of changing a_i in step $m+1$ is the minimum

$$\min_{0\le j\le p-1}\{\left|\cos t[i+jl]+(-1)^{\delta(i+jl)}\cos t[i+jl+pl]\right|\}$$

of the costs of changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$, $j=1, 2, \cdots, p$, at the same time in step m. Therefore, cost[i] is computed correctly in this case.

The discussion similar to above will show that cost[i] is computed correctly when $T_{i1}<T_{i0}$.

If $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$ holds, then $T_i'=0$. Notice that at the end of this step we will let $a\leftarrow(A_1+A_2+\cdots+A_p, A_2+A_3+\cdots+A_{p+1})$. If we change a_i in step $m+1$ we must change one of a_i, a_{i+l}, $\cdots$, $a_{i+(p-1)l}$ in step m. In order to maintain $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, i.e. $a_i+a_{i+pl}=a_{i+l}+a_{i+l+pl}=\cdots=a_{i+(p-1)l}+a_{i+(2p-1)l}$ in step m, changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$ must happen at the same time, $j=1, 2, \cdots, p$. Since $b_{i+(j-1)l}$ isn't change in forcing $b_i=b_{i+l}=\cdots=b_{i+(p-1)l}$, we have that $\delta(i+jl)=0$. The cost of changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$ at the same time is

$$\text{cost}[i+jl]+\text{cost}[i+jl+pl] = |\text{cost}[i+jl]+(-1)^{\delta(i+jl)}\text{cost}[i+jl+pl]|.$$

Therefore, the cost of changing a_i in step $m+1$ is the minimum

$$\min_{0\le j\le p-1}\{|\cos t[i+jl]+(-1)^{\delta(i+jl)}\cos t[i+jl+pl]|\}$$

of the costs of changing $a_{i+(j-1)l}$ and $a_{i+(j-1)l+pl}$, $j=1, 2, \cdots, p$, at the same time in step m. Therefore, cost[i] is computed correctly in this case.

Finally, we consider the case $T'>k$. In this case we cannot force $A_1+A_{p+1}=A_2+A_{p+2}=\cdots=A_p+A_{2p}$. Notice that at the end of this step we will let $a\leftarrow(A_1+A_3+\cdots+A_{2p-1}, A_2+A_4+\cdots+A_{2p})$. If we change a_i in step $m+1$ we have effectively restored $a_i=\sum_{j=1}^{p}a_{i+2(j-1)l}$ (in step m). We need only change one of $\{a_{i+2(j-1)l}, j=1, 2, \cdots, p\}$ in step m, hence the cost of changing a_i is the minimum $\min_{0\le j\le p-1}\{\cos t[i+2jl]\}$ of the costs of changing $a_{i+2(j-1)l}$, $j=1, 2, \cdots, p$. Therefore, cost[i] is computed correctly in this case.

4. Conclusion

In this paper, an efficient algorithm for determining the k-error linear complexity of a binary sequence s with period $2p^n$ is presented, where 2 is primitive root modulo p^2. The algorithm computes the k-error linear complexity of s in n steps. The algorithm solves partially the open problem by Stamp and Martin [6].

Acknowledgments

This work was supported in part by the Natural Science Foundation of China under Grant 60573026 and 60773121, the Key Project of Chinese Ministry of Education under Grant 205074, and the Academic and Technical leading scholars Research Project of the Education Department of Anhui Province in China under Grant 2005hbz24, and the Natural Science Foundation of Anhui Province in China under Grant 070412052.

References

[1] S. R. Blackburn, "A generalisation of the discrete Fourier transform: determining the minimal polynomial of a periodic sequence", *IEEE Trans. Inform. Theory,* Vol.40, pp.1702-1704, 1994

[2] Ding, C., Xiao, G., Shan, W.: The Stability Theory of Stream Ciphers. Lecture Notes in Computer Science, Vol. 561. Springer-Verlag, Berlin Heidelberg New York (1991)

[3] Games, R. A., Chan, A. H.: A fast algorithm for determining the complexity of a binary sequence with period 2^n. IEEE Trans on Inform. Theory. 29(1): (1983)144-146

[4] T. Kaida, S. Uehara and K. Imamura, "An algorithm for the k-error linear complexity of sequences over GF(p^m) with period p^n, p a prime", *Information and Computation*, Vol. 151, pp.134-147, 1999

[5] Massey, J. L.: Shift register synthesis and BCH decoding. IEEE Trans. on Inform. Theory. 15(1): (1969)122-127

[6] M. Stamp, C. F. Martin, "An algorithm for k-error linear complexity of binary sequences with period 2^n, *IEEE Trans on Inform. Theory*, vol. 39, pp. 1398-1401, 1993

[7] Wei S, Chen Z, Xiao G. A fast algorithm for k-error linear complexity of a binary sequence. 2001 International Conferences on Info-tech and Info-net Proceedings, Conference E, IEEE Press, 2001, 152-157

[8] Wei, S., Xiao, G., Chen, Z.: A fast algorithm for determining the linear complexity of a binary sequence with period 2^np^m. Science in China(Series F). 44(6): (2001)453-460

Shimin Wei received the B. S. degree in Mathematics from the Huaibei Coal Normal College, Huaibei, Anhui, China, in 1986, the M. S. Degree in Mathematics from the Northwest University, Xi'an, Shaanxi, China, in 1993, and the Ph. Degree in Cryptography from the Xidian University, Xi'an, Shaanxi, China, in 2001. From April 2001 to July 2003, he was a postdoctoral with the Department of Department of Computer Science and Technique, Peking University, Beijing, China.

He was a Lecturer from June 1993 to November 1994, was a associate professor from December 1994 to November 1996, has been a professor since December 1996, with the Department of Mathematics and the Department of Computer Science & Technique, Huaibei Coal Normal College, Huaibei, Anhui, China. Since October 2003, he has been the header with the Department of Computer Science & Technique, Huaibei Coal Normal College. His research interests include Applied Mathematics, Cryptography and Coding, Information and Network Security.