Chosen Ciphertext Attack on a New Class of Self-Synchronizing Stream Ciphers*

Bin Zhang[1,2], Hongjun Wu[1], Dengguo Feng[2], and Feng Bao[1]

[1] Institute for Infocomm Research, Singapore
[2] State Key Laboratory of Information Security,
Graduate School of the Chinese Academy of Sciences,
Beijing 100039, P.R. China
zhangbin@mails.gscas.ac.cn
{hongjun, baofeng}@i2r.a-star.edu.sg

Abstract. At Indocrypt'2002, Arnault et al. proposed a new class of self-synchronizing stream ciphers combining LFSR and FCSR architectures. It was claimed to be resistant to known attacks. In this paper, we show that such a self-synchronizing stream cipher is extremely vulnerable to chosen ciphertext attack. We can restore the secret keys easily from one chosen ciphertext with little computation. For the parameters given in the original design, it takes less than one second to restore the secret keys on a Pentium 4 processor.

Keywords: Stream cipher, Self-synchronizing, 2-adic expansion, Feedback shift register.

1 Introduction

Stream ciphers are an important class of encryption algorithms in practice. In general, they are classified into two kinds: synchronous stream ciphers and self-synchronous stream ciphers [5]. In a self-synchronizing stream cipher, each plaintext bit affects the entire following ciphertext through some mechanism, which makes it more likely to be resistant against attacks based on plaintext statistical properties. Since several ciphertext bits may be incorrectly decrypted when a bit modification occurs in the ciphertext, such a mechanism provides additional security against active attacks.

In [1], a new class of self-synchronous stream ciphers was proposed which exploits the concatenation of LFSR and FCSR. The main idea behind such a design is to confuse the $GF(2)$ linearity with the 2-adic linearity so that neither of the synthesis algorithms (Berlekamp-Massey type algorithms) can work in this case. However, as we will show in this paper, such a simple design is extremely weak under chosen ciphertext attack. By choosing one ciphertext, we can recover

* Supported by National Natural Science Foundation of China (Grant No. 60273027), National Key Foundation Research 973 project (Grant No. G1999035802) and National Science Fund for Distinguished Young Scholars (Grant No. 60025205).

A. Canteaut and K. Viswanathan (Eds.): INDOCRYPT 2004, LNCS 3348, pp. 73–83, 2004.

the secret keys with little computation. Assume both LFSR and FCSR are of length 89, as suggested by the authors [1], we can recover both the structures in 1 second on a Pentium 4 processor.

This paper is organized as follows. In Section 2, we will give an introduction to the self-synchronizing stream cipher together with some backgrounds. Our attack on this cipher is given in Section 3 and detailed experimental results are also included in this section. Finally, some conclusions are given in Section 4.

2 The Self-Synchronizing Stream Cipher

In this section, we will first review some backgrounds including 2-adic arithmetic and the Galois representations of LFSR and FCSR. Then a detailed description of the self-synchronizing stream cipher is presented.

2.1 2-Adic Arithmetic, Galois Representations of LFSR and FCSR

A 2-adic integer is a formal power series $s = \sum_{i=0}^{\infty} s_i 2^i$ with $s_i \in \{0, 1\}$. We denote the set of 2-adic integers by $\mathbb{Z}_2$. The addition and multiplication in $\mathbb{Z}_2$ is done according to $2^i + 2^i = 2^{i+1}$ for all $i \geq 0$, i.e. taking the carry to the higher order term. Thus the addition inverse of 1 is $\sum_{i=0}^{\infty} 2^i = -1$ and if a 2-adic integer $s = 2^r + \sum_{i=r+1}^{\infty} s_i 2^i$, its addition inverse is $-s = 2^r + \sum_{i=r+1}^{\infty} (1 - s_i) 2^i$.

A feedback with carry shift register (FCSR) is a device for the fast generation of pseudorandom sequence with good statistical properties and large period. Like LFSR, FCSR also has two architectures: Fibonacci structure and Galois structure [3]. The Galois architecture is more efficient due to the parallel computation of feedbacks. As in [1], we only consider the Galois structure in this paper.

Lemma 1 characterizes the eventually periodic binary sequences in terms of 2-adic integers.

Lemma 1. *[3] Let $S_2 = \sum_{i=0}^{\infty} s_i 2^i$ be the 2-adic integer corresponding to a binary sequence $S = \{s_i\}_{i \geq 0}$. S is eventually periodic if and only if there exists two integers p and q in $\mathbb{Z}$ such that $S_2 = p/q$ with q odd, see Figure 1. Further, S is strictly periodic if and only if $pq \leq 0$ and $|p| \leq |q|$.*

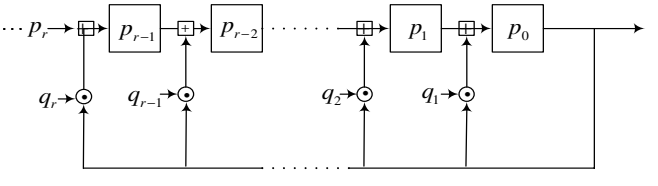

Fig. 1. Galois representation of a FCSR

The 2-adic division of p/q is fulfilled by a FCSR using Galois architecture. Without loss of generality, we always assume $p = \sum_{i=0}^{r} p_i 2^i + p_{r+1} 2^{r+1} + \cdots \geq 0$ and $q = 1 - \sum_{i=1}^{r} q_i 2^i < 0$ with p_i and $q_i \in \{0, 1\}$. In Figure 1, $\boxplus$ denotes the

addition with carry, i.e. the output of $a \boxplus b$ is $a \oplus b \oplus c_{n-1}$ and the carry is $c_n = ab \oplus ac_{n-1} \oplus bc_{n-1}$. $\odot$ is the binary multiplication. The period of S is the smallest integer t such that $2^t \equiv 1 \pmod{q}$.

Similarly, the Galois representation of a LFSR is shown in Figure 2, where the input of the circuit is $S(x) = \sum_{i=0}^{\infty} s_i x^i$ and the output is $S'(x) = S(x)/Q(x)$, with $Q(x) = 1 + \sum_{i=1}^{r} q_i x^i$ and $\oplus$ being Xor.

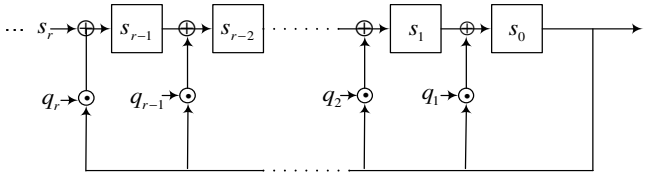

Fig. 2. Galois representation of a LFSR

2.2 Description of the Self-Synchronizing Stream Cipher

The structure of this cipher is a concatenation of one LFSR and one FCSR, as shown in Figure 3. An irreducible primitive polynomial $Q(x)$ of prime degree k is

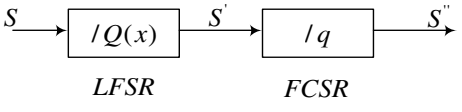

Fig. 3. The self-synchronizing stream cipher

used as the feedback polynomial of the LFSR. A negative prime q for the FCSR divisor box is of size k satisfying $|q| = 2u+1$ with u a prime congruent to 1 modulo 4 and $\gcd(|q|-1, 2^k-1) = 1$, where $|\cdot|$ denotes the absolute value. For practical applications, the authors of [1] suggest $k = 89$. Initialize LFSR and FCSR randomly and denote the message to be encrypted by S, the encryption scheme is:

1. Compute $S'(x) = S(x)/Q(x)$ by the LFSR divisor-box.
2. Convert $S'(x)$ into the 2-adic integer $S'(2)$.
3. Compute the ciphertext $S'' = S'(2)/q$ by the FCSR divisor-box.

Upon decrypting, without loss of generality, initialize all the LFSR and FCSR cells (including the carries) to be zero. The corresponding decryption scheme is:

1. Compute $S'(2) = qS''(2)$.
2. Convert $S'(2)$ into the formal power series $S'(x)$.
3. Compute the plaintext S by $S(x) = Q(x)S'(x)$.

The decryption circuits are only generally discussed in [1], no concrete circuit is given in that paper. It is claimed in [1] that this cipher is fast and secure against known attacks. However, as we will show below, this self-synchronizing stream cipher is far away from security.

3 Our Attack

In this section, we will show that this self-synchronizing stream cipher is very vulnerable to chosen ciphertext attack. Subsection 3.1 gives the circuits fulfilling multiplication by q and $Q(x)$, respectively. Our attack is given in subsection 3.2. The experimental results are given in subsection 3.3 in detail.

3.1 The Multiplication Circuits

Without loss of generality, we propose the following two circuits to fulfill the multiplication by q and $Q(x)$, respectively. The only purpose of this section is to show that the proposed stream cipher can actually decrypt the ciphertext.

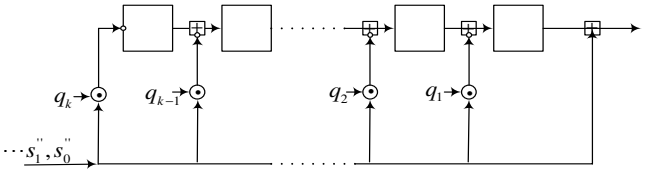

Fig. 4. Multiplication circuit with q being the multiplier

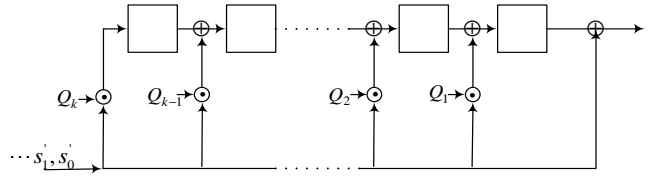

Fig. 5. Multiplication circuit with $Q(x)$ being the multiplier

In Figures 4 and 5, $q = 1 - \sum_{i=1}^{k} q_i 2^i$ and $Q(x) = 1 + \sum_{i=1}^{k} Q_i x^i$ are the secret parameters used in the cipher. Upon decrypting, let all the cells in Figures 4 and 5 (including the carries) be zero. From (1), the inputs corresponding to the k cells in Figure 4 are the addition inverses of the ciphertext, while the input corresponding to rightmost $\boxplus$ is the ciphertext itself. We denote the inverse operation by a hollow circle in Figure 4.

$$\begin{aligned}
&(s_0'' + s_1'' \cdot 2^1 + s_2'' \cdot 2^2 + \cdots)(1 - q_1 \cdot 2^1 - q_2 \cdot 2^2 - \cdots - q_k \cdot 2^k) \\
&= -(s_0'' + s_1'' \cdot 2^1 + s_2'' \cdot 2^2 + \cdots)(q_1 \cdot 2^1 + q_2 \cdot 2^2 + \cdots + q_k \cdot 2^k - 1) \\
&= -(s_0'' + s_1'' \cdot 2^1 + s_2'' \cdot 2^2 + \cdots)(q_1 \cdot 2^1 + q_2 \cdot 2^2 + \cdots + q_k \cdot 2^k) + (s_0'' + s_1'' \cdot 2^1 \\
&\quad + s_2'' \cdot 2^2 + \cdots).
\end{aligned} \tag{1}$$

3.2 A Chosen Ciphertext Attack

The basic idea of our attack is that if we choose a special ciphertext fed into the decryption circuits such that the corresponding decrypted message (including a preamble) is of finite length, then we can retrieve the secret keys q and $Q(x)$ by simple factoring the polynomial corresponding to the decrypted message over $GF(2)$ [2, 6].

Since the secret q is a negative prime, the decrypted message is of finite length if the ciphertext fed into the decryption circuits is a binary string with the following form:

$$(\underbrace{*, *, \cdots, *}_{A}, \underbrace{1, 1, \cdots, 1, 1}_{B}), \tag{2}$$

where A is a randomly-chosen binary prefix of certain length and B is a all-1 string of certain length. We use a randomly-chosen string A to disguise the subsequent all-1 string. (3) confirms the validity of the above chosen ciphertext.

$$\begin{aligned} & (s_0'' + s_1'' \cdot 2^1 + \cdots + s_l'' \cdot 2^l + 1 \cdot 2^{l+1} + 1 \cdot 2^{l+2} + \cdots)q \\ = & (s_0'' + s_1'' \cdot 2^1 + \cdots + s_l'' \cdot 2^l)q \\ & + (1 \cdot 2^{l+1} + 1 \cdot 2^{l+2} + \cdots)q \\ = & (s_0'' + s_1'' \cdot 2^1 + \cdots + s_l'' \cdot 2^l)q + \frac{2^{l+1}}{1-2}q \\ = & (s_0'' + s_1'' \cdot 2^1 + \cdots + s_l'' \cdot 2^l)q + 2^{l+1} \cdot (-q) \\ & \rightarrow D(x), \end{aligned} \tag{3}$$

where $A = (s_0'', s_1'', \cdots, s_l'')$ is of length l. From (3), $(s_0'' + s_1'' \cdot 2^1 + \cdots + s_l'' \cdot 2^l)q + 2^{l+1} \cdot (-q)$ corresponds to a polynomial $D(x)$ of finite degree. Therefore, the polynomial corresponding to the decrypted message is $D(x)Q(x) \in GF(2)[x]$. According to [6], the following lemma holds.

Lemma 2. *[6] A univariate polynomial of degree n over the finite field $GF(p^k)$, where p is a small, fixed prime, can be factored with a deterministic algorithm whose running time is $O((nk)^2)$.*

From our experimental results, the degree of $D(x)Q(x)$ is less than 300. Hence, the complexity of factoring $D(x)Q(x)$ over $GF(2)$ is only $O(2^{16})$.

Upon factoring the decrypted message, we get both $Q(x)$ and $D(x)$. Let $D(x) = d_0 + d_1x^1 + d_2x^2 + \cdots + d_hx^h$. Keeping in mind that $D(x)$ is the polynomial representation of $(s_0'' + s_1'' \cdot 2^1 + \cdots + s_l'' \cdot 2^l)q + 2^{l+1} \cdot (-q)$, $d_0 + d_1 \cdot 2^1 + d_2 \cdot 2^2 + \cdots + d_h \cdot 2^h$ has an integer factor $(-q)$. Therefore, factoring the integer $d_0 + d_1 \cdot 2^1 + d_2 \cdot 2^2 + \cdots + d_h \cdot 2^h$ retrieves the secret q. Since $2^{89} \leq (-q) \leq 2^{90}$, $(-q)$ is an integer with at most 28 decimal digits which can be recovered easily by factoring $d_0 + d_1 \cdot 2^1 + d_2 \cdot 2^2 + \cdots + d_h \cdot 2^h$ using the number field sieve algorithm [4].

A full description of our attack is as follows.

1. Choose a string as shown in (2) and feed it into the decryption circuits.

2. Convert the decrypted message into polynomial form and factor it to get $Q(x)$ and $D(x)$.
3. Transform $D(x)$ into the integer form and factor the integer to recover q.

The complexity of our attack is very low for the parameter $k = 89$. See Subsection 3.3.

3.3 Experimental Results

We have implemented the above attack on a Pentium 4 processor, see Appendix A for the C source codes. The parameters of this self-synchronizing stream cipher are chosen as follows.

$$Q(x) = x^{89} + x^6 + x^5 + x^3 + 1, \;\; q = -618970052618499608724417827, \quad (4)$$

where $Q(x)$ is a primitive polynomial of degree 89 and q is a negative prime satisfying the following three conditions:

1. $2^{89} \leq (-q) \leq 2^{90}$.
2. $\gcd(618970052618499608724417826, 2^{89} - 1) = 1$.
3. $|q| = 2u + 1$ with $u = 309485026309249804362208913$ is a prime congruent to 1 modulo 4.

These three conditions are used to verify the candidate keys obtained from the attack. We choose a 600-bit ciphertext as follow.

$$A = (10101110101010001101001100110010011100 01) \,||\, B = (11 \cdots 11), \quad (5)$$

where A is a randomly-chosen string of length 40 and B is a all-1 string of length 560. Feed the above chosen ciphertext into the decryption circuits and get the result. The polynomial corresponding to the decrypted message is $x^{216} + x^{215} + x^{214} + x^{210} + x^{209} + x^{207} + x^{206} + x^{203} + x^{202} + x^{199} + x^{198} + x^{196} + x^{194} + x^{192} + x^{190} + x^{183} + x^{181} + x^{179} + x^{178} + x^{176} + x^{174} + x^{173} + x^{167} + x^{166} + x^{165} + x^{163} + x^{159} + x^{155} + x^{153} + x^{149} + x^{148} + x^{147} + x^{145} + x^{144} + x^{143} + x^{140} + x^{139} + x^{136} + x^{133} + x^{132} + x^{131} + x^{128} + x^{126} + x^{123} + x^{122} + x^{120} + x^{118} + x^{113} + x^{111} + x^{110} + x^{105} + x^{104} + x^{103} + x^{101} + x^{100} + x^{99} + x^{98} + x^{96} + x^{95} + x^{92} + x^{91} + x^{89} + x^{88} + x^{85} + x^{78} + x^{75} + x^{74} + x^{73} + x^{72} + x^{71} + x^{67} + x^{64} + x^{61} + x^{60} + x^{59} + x^{56} + x^{52} + x^{51} + x^{49} + x^{42} + x^{40} + x^{35} + x^{34} + x^{32} + x^{29} + x^6 + x^5 + x^3 + 1$. It takes 0.203 seconds to factor the above polynomial using Mathematica. The result is $(1 + x + x^4 + x^5 + x^6)(1 + x^7 + x^9 + x^{11} + x^{13} + x^{14} + x^{15})(1 + x + x^2 + x^5 + x^8 + x^9 + x^{11} + x^{16} + x^{17} + x^{19} + x^{20} + x^{21} + x^{24} + x^{25} + x^{26})(1 + x^3 + x^8 + x^9 + x^{10} + x^{11} + x^{15} + x^{17} + x^{18} + x^{22} + x^{24} + x^{25} + x^{29} + x^{30} + x^{31} + x^{32} + x^{33})(1 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{13} + x^{15} + x^{19} + x^{20} + x^{22} + x^{23} + x^{24} + x^{26} + x^{27} + x^{30} + x^{31} + x^{33} + x^{34} + x^{35} + x^{36} + x^{37} + x^{40} + x^{42} + x^{43} + x^{44} + x^{45} + x^{46} + x^{47})(1 + x^3 + x^5 + x^6 + x^{89})$. Expand the factors other than $x^{89} + x^6 + x^5 + x^3 + 1$ and let $x = 2$. The result is 302266531961499475785005717448795619329. By Mathematica, it takes 0.219 seconds to factor this integer. The result is $3^2 \cdot 23 \cdot 8839 \cdot 266899 \cdot 618970052618499608724417827$. Therefore, the total time required to restore $Q(x)$ and q is about 0.422 seconds.

Some Remarks. A pseudorandom generator with a similar structure was also proposed in [1]. Since our attack only work in the chosen ciphertext scenario, the security of that generator is not influenced by our attack.

4 Conclusion

In this paper, we showed that the proposed self-synchronizing stream cipher is extremely weak against chosen ciphertext attack. We can restore the secret keys easily from a 600-bit chosen ciphertext in 1 second on a Pentium 4 processor. We suggest that this cipher should not be used in practice.

References

1. F. Arnault, T. P. Berger, A. Necer, "A New Class of Stream Ciphers Combining LFSR and FCSR Architectures", *Progress in Cryptology-INDOCRYPT'2002*, LNCS vol. 2551, Springer-Verlag,(2002), pp. 22-33.
2. Berlekamp, E. R., "Factoring polynomials over finite fields", *Bell Systems Tech. J.*, 46, 1967, pp. 1853-1859.
3. M. Goresky, A. M. Klapper, "Fibonacci and Galois Representations of Feedback-With-Carry Shift Registers", *IEEE Transactions on Information Theory*, vol. 48, No. 11, 2002, pp. 2826-2836.
4. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, "The Number Field Sieve", *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, Baltimore, Maryland, May 1990, pp. 14-16.
5. A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press,1997.
6. V. Shoup, "A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic", *Proc. 1991 International Symposium on Symbolic and Algebraic Computation*, 1991, pp. 14-21.

A A Non-optimized C Implementation of Our Attack

```
#include "stdio.h"
#include "math.h"
#define SIZE 8000

unsigned char LFSR[89],FCSR[89],carry[89],cipher[SIZE],
re[SIZE],fe[SIZE],inter[SIZE],plain[SIZE],deinter[SIZE];

void main()
{
 unsigned char S[256],z[256],G[5]={179,43,228,11,194};
 unsigned char RF[90],RL[90],i1,j1,L[90],F[90];
 unsigned int i,j,n,k1;
```

```
//use RC4 as the random source
for(k1=0;k1<256;k1++)S[k1]=k1;
 for(j1=0,k1=0;k1<=255;k1++)
 {
     j1=j1+S[k1]+G[k1%5];
     i1=S[k1];S[k1]=S[j1];S[j1]=i1;
 }

 for(i1=0,j1=0,k1=0;k1<256;k1++)
 {
     i1++;j1=j1+S[i1];n=S[i1];S[i1]=S[j1];S[j1]=n;
     z[k1]=S[(unsigned char)(S[i1]+S[j1])];
 }

 //initialization
 for(i=0;i<SIZE;i++)
     inter[i]=cipher[i]=plain[i]=deinter[i]=re[i]=fe[i]=0;
 for(i=0;i<89;i++)
     LFSR[i]=FCSR[i]=carry[i]=0;
 for(i=0;i<300;i++)
     for(j=0;j<8;j++)
     {
         deinter[8*i+(j%8)]=(z[i]&(1<<j))>>j;
     }

 //printf("The message is: \n");
 //for(i=0;i<300;i++)printf("%x ",deinter[i]);printf("\n");

 // for simplicity, we just initial the LFSR
 // and FCSR as follows.
 LFSR[0]=1;
 LFSR[1]=1;
 LFSR[45]=1;
 //LFSR[3]=1;
 LFSR[88]=1;
 //FCSR[1]=1;
 FCSR[0]=1;
 FCSR[29]=1;
 //FCSR[3]=1;
 FCSR[88]=1;

 for(i=0;i<90;i++)
     L[i]=F[i]=0;
 for(i=0;i<90;i++)
     RL[i]=RF[i]=0;
```

```
L[0]=L[3]=L[5]=L[6]=L[89]=1;
F[0]=F[2]=F[5]=F[8]=F[10]=1;
F[11]=F[13]=F[15]=F[18]=F[19]=F[22]=1;
F[26]=F[28]=F[29]=F[42]=F[43]=F[44]=1;
F[47]=F[48]=F[53]=F[55]=F[56]=F[59]=1;
F[62]=F[63]=F[64]=F[89]=1;
for(i=0;i<600;i++)
{
    inter[i]=(LFSR[0] & 1);
    j1=deinter[i];
    for(j=0;j<90;j++)
        RL[j]=(L[j] & LFSR[0]);
    for(j=0;j<88;j++)
        LFSR[j]=RL[j+1]^LFSR[j+1];
    LFSR[88]=RL[89]^j1;
}


for(i=0;i<600;i++)
{
    j1=inter[i];
    cipher[i]=FCSR[0];
    for(j=0;j<90;j++)
        RF[j]=(F[j] & FCSR[0]);
    for(j=0;j<88;j++)
    {
      FCSR[j]=(FCSR[j+1]+RF[j+1]+carry[j])&1;
      carry[j]=
      (unsigned char)((FCSR[j+1]+RF[j+1]+carry[j])&(1<<1))>>1;
    }
    FCSR[88]=(RF[89]+j1+carry[88])&1;
    carry[88]=
    (unsigned char)((RF[89]+j1+carry[88])&(1<<1))>>1;
}

//printf("The ciphertext is: \n");
//for(i=0;i<300;i++)printf("%x ",cipher[i]);printf("\n");

// our attack
for(i=40;i<600;i++)
    cipher[i]=1;
printf("***\n");
for(i=0;i<40;i++)
    printf("%x ", cipher[i]);
printf("***\n");
```

```
for(i=0;i<600;i++)
    re[i]=cipher[i];

i=0;
while (cipher[i]==0) i=i+1;
for(n=i+1;n<600;n++)
    cipher[n]=cipher[n]^1;

for(i=0;i<90;i++)
    L[i]=F[i]=0;
for(i=0;i<89;i++)
    FCSR[i]=carry[i]=0;
for(i=0;i<90;i++)
    RL[i]=RF[i]=0;
L[0]=L[3]=L[5]=L[6]=L[89]=1;
F[0]=F[2]=F[5]=F[8]=F[10]=1;
F[11]=F[13]=F[15]=F[18]=F[19]=F[22]=1;
F[26]=F[28]=F[29]=F[42]=F[43]=F[44]=1;
F[47]=F[48]=F[53]=F[55]=F[56]=F[59]=1;
F[62]=F[63]=F[64]=F[89]=1;
for(i=0;i<600;i++)
{
    j1=cipher[i];
    i1=re[i];
    fe[i]=((FCSR[0]+i1+carry[0])&1);
    carry[0]=
    (unsigned char)((FCSR[0]+i1+carry[0])&(1<<1))>>1;
    for(j=1;j<90;j++)
        RF[j]=(j1 & F[j]);
    for(j=0;j<88;j++)
    {
      FCSR[j]=(RF[j+1]+FCSR[j+1]+carry[j+1])&1;
      carry[j+1]=
      (unsigned char)((RF[j+1]+FCSR[j+1]+carry[j+1])&(1<<1))>>1;
    }
    FCSR[88]=j1;
}

for(i=0;i<89;i++)
    LFSR[i]=0;
for(i=0;i<600;i++)
{
    j1=fe[i];
    plain[i]=((LFSR[0]^j1)&1);
    for(j=1;j<90;j++)
```

```
            RL[j]=(L[j] & j1);
        for(j=0;j<88;j++)
            LFSR[j]=(LFSR[j+1]^RL[j+1]);
        LFSR[88]=j1;
    }


    printf("The decrypted message is: \n");
    for(i=0;i<300;i++)
        printf("%x ",plain[i]);
    printf("\n");

}
```

After getting the result, we can use Mathematica to restore the secret keys.