A complete analyze of using Shift Registers in Cryptosystems for Grade 4, 8 and 16 Irreducible Polynomials

MIRELLA AMELIA MIOC
Computer Science Department
"Politehnica" University of Timisoara
Bd. V. Parvan 2, RO-300223
ROMANIA
mmioc@cs.utt.ro http://www.cs.upt.ro/ro/Staff/cd.php?id=21

Abstract: A Linear Feedback Shift Register (LFSR) is always the kernel of any digital system based on pseudorandom bits sequences and is frequently used in cryptosystems, in codes for errors detecting, in wireless system communication. The Advanced Encryption System (Rijndael) is based on using a grade 8 irreducible polynomials in a Galois Field. For a better understanding this study contains aspects of functioning for Linear Feedback Shift Register and Multiple Input-Output Shift Register (MISR) using grade 4, 8 and 16 irreducible polynomials. This experiment shows that the Linear Feed-back Shift Register and Multiple Input-Output Shift Register have the same function. The conclusion of this paper is that for grade 8 and 16 irreducible polynomials the weights are calculated with a formula discovered in this work.

Key-Words: Cryptosystem, Shift registers, Calculate, Irreducible polynomials, Simulate, Rijndael, Pseudo-Random Sequence, Error Detect.

1 Introduction

Beginning with 2000 Rijndael [1] [2] cryptosystem is officially the Advanced Encryption System (AES) [7], [8]. The old DES (Data Encryption Standard) [3], [4] was broken from Electronic Frontier Foundation in 3 days. The two authors Joan Daemen and Vincent Rijman from Holland chose to use a Galois Field GF (2^8) with the following generator polynomial [7], [8].

$$P(x) = x^8+x^4+x^3+x+1 \quad (1)$$

All arithmetical operations will be developed in a Galois group.

The Shift Register Cryptosystems' variant has been developed from the evolution of the encrypting techniques [5]. Such a cryptosystem is based upon generating a sequence in a finite field and for obtaining it a Feedback Shift Register is used.

The Linear Feedback Shift Registers are used in a variety of domains [5]: sequences generators; counters; BIST (Built-In-Self-Test) [6]; encryption; PRBS (Pseudo-Random Bit Sequences).

LFSR can be realized based on XOR (exclusive OR) circuits or XNOR (exclusive denied OR).

Of course, the difference is of status, the equivalent status will be 1, where it was 0. For an n bits LFSR, all the registers will be configured as shirt registers, but only the last significant register will determine the feedback.

An n bits register will always have n + 1 signals.

A feedback shift register is composed of:

- a shift register
- a feedback function.

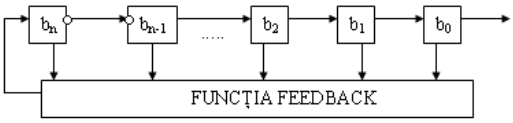

Fig. 1 Feedback Shift Register

A LFSR is a shift register, whose input bit is given from a linear function of the initial status.
The initial value of the register is called seed and the sequence produced is completely determined by the initial status. Because the register has a finite number of possible statuses, after a period the sequence will be repeated. If the feedback function is very good chosen the produced sequence will be random and the cycle will be very long. The list

with the position that influences the future status is called tap sequence. For example, for the next LFSR this list is [16, 14, 13, 11]:

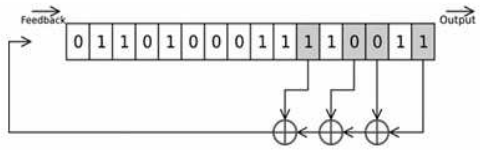

Fig. 2 A LFSR scheme

The tap sequence can be represented as a polynomial mod 2, with the coefficients 1 or 0. This is called feedback polynomial or characteristic polynomial. For the Fig.2 this polynomial is:

$$S(x) = x^{16}+x^{14}+x^{13}+x^{11}+1 \qquad (2)$$

The 1 number from the polynomial isn't correspondent in tap sequence, but the powers of the terms are correspondent to the bits from that sequence. Though the pseudorandom bits sequence produced a LFSR is the most important part of any digital systems with application in cryptography in measurement based on the error bit rates, in wireless communication systems. There are two kinds of implementation for LFSR:

- Fibonacci implementation
- Galois form.

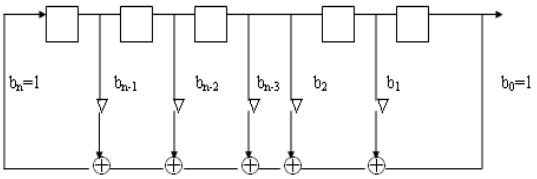

Fig. 3 Fibonacci implementation

In Fibonacci form the weight for any status is 0, when there isn't any connection and 1 for sending back. Exceptions of this are the first and the last one, both connected, so always on 1.

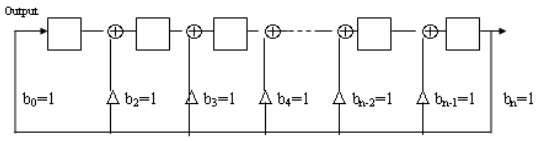

Fig. 4 Galois implementation

In Galois implementation there is a Shift Register, whose content is modified each step at a binary value sent to the output. Comparing the two type of representation it is shown that the weight order in Galois is opposite the one in Fibonacci. From the hardware point of view, Galois implementation is fastest than Fibonacci because of the reduced number of XOR gates in feedback and so it is much more used. There are some industries in which Fibonacci form is referenced as SSRG (Simple Shift Register Generator) and Galois as MRSRG(Multiple-Return Shift Register Generator). There are two types of LFSR from the utilization point of view: the well-known LFSR, that is an "in-tapping" LFSR and the "out-tapping" LFSR. The "in-tapping" LFSR is usually called a MISR (Multiple Input Shift Register). A MISR is in fact a modified LFSR, thus functioning is a little bit slower. Cycle codes belong to algebraically codes for errors detecting. This paper develops an analyze of a Linear Feedback Shift Register and a Multiple Input – output Shift Register.

2 Functioning of LFSR and MISR with grade 4 irreducible polynomials

It was necessary to use different pseudorandom sequence for input. For analyzing the functioning of a LFSR based on a grade 4 irreducible polynomial there are 3 different schemes. For example for the polynomial

$$P(x) = x^4 + x + 1 \qquad (2)$$

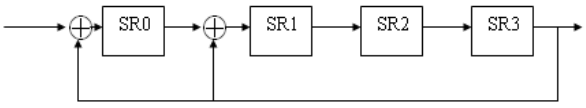

Fig. 5 Scheme A

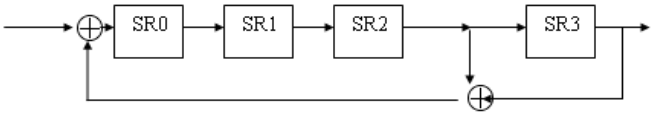

Fig. 6 Scheme B

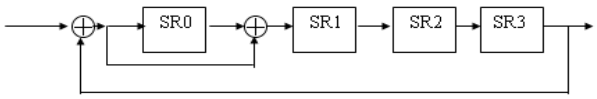

Fig. 7 Scheme C

For each type of scheme is another program for simulating the functioning of LFSR. The relations for the calculus of each weight are:

$$\begin{aligned} S_0&=1*P(x)\\ S_1&=x*P(x)\\ S_2&=x^2*P(x)\\ S_3&=x^3*P(x) \end{aligned} \quad (3)$$

For each of them these relations were verified. Also dividing the input polynomial with x^4+x+1 the rests were correct. The program contains also the time at the beginning and at the ending of the operations. Separate programs have been developed for the theoretical method presented above for calculating the output of a Linear Feedback Shift Register using the polynomial

$$x^4 + x + 1 \quad (4)$$

A program for simulating the functioning of the LFSR for scheme A will be presented in the following lines. It has to receive as input data the coefficients of the input polynomial that was already calculated in a separate part. This string of zeros and ones is afterwards used for calculating the output of the LFSR. The counter (used after calculating both the output of a Linear Feedback Shift Register and the output of a Multiple Input-Output Shift Register) is started right before the starting of the calculation. The calculation itself consists of a cycle that for every value in the input data, an intermediate result for an intermediate step is calculated as it follows.

-For the a[0] value, the program gets the result for an XOR between the value in the input data string, found on the position that has the number of the step, and the previous value(from the previous step) of a[0]; in the program, the data from where the calculation starts is contained in the string array s and this status is counted as step 0-the initial step.

-For the a[1] value, the result is calculated in the program as an XOR between the s[0](intermediate value from the previous step) ands[n-1], where n is the value of the highest power of the polynomial, actually the grade of the polynomial; both values used in this calculation are calculated during the previous step.

-For the other two values, a[2] and a[3], the calculation is as simple as this: they get the value of s[i-1], where i is either 2 for a[2], or 3 for a[3].

All operations made in this procedure for the current step are related to the results obtained during the calculation of the previous step.

The procedure that does this calculation for the intermediate cycles and also for the most important, the final values, is available below:

```
void prelucrare(int n, int knt)
{ a[0]=s[n-1]^x[knt];
 a[1]=s[0]^s[n-1];
 for (int i=2;i<4;i++)
  a[i]=s[i-1];
 k++;
}
```

As soon as the final results are obtained in the main part of the program, the counter is stopped and the elapsed time is calculated and stored.

For calculating the output of a Multiple Input-Output Shift Register there was developed a program for scheme A.

As input data, the program uses 4 columns of zeros and ones on 20 rows. These 20 sets of 4 bits are used to obtain the output of the MISR. After reading the input data in the program, the counter is starting.

This time the calculation uses for each step one row of the input data:

-For the value of a[0], the result is obtained from XOR between s[n-1] and the value from the input data on the position “first row, first column” x[0][0](for the first calculated step), using the initial step where all the elements in the array s are zero.

-For the a[1] value, the calculation is made as an XOR between s[0] and s[n-1] and x[1][0], where n is 4 in this example and this example refers to the first step, done starting from the initial step, in which the string s contains just zeros.

-For all the other values(a[2] and a[3]), the result is obtained by XOR between s[1] and x[2][0] for a[2] and s[2] and x[3][0] for a[3].

-For the nest steps, the procedure for calculation is the same, only the indexes are changing and therefore the values used in the calculation.

In this procedure, all calculations are made based on the previous step results, stored in the string s.

The final values for the calculation of MISR are obtained from this procedure:

```
void prelucrare(int n, int knt)
{ a[0]=s[n-1]^x[0][knt];
 a[1]=x[1][knt]^(s[0]^s[n-1]);
 for (int i=2;i<n;i++)
  a[i]=s[i-1]^x[i][knt];
 k++;
 }
```

After the results are obtained in the main part of the program, the counter is stopped so that the elapsed time, used for getting the MISR output, can be calculated.

In the case of the B schema for LFSR, the program developed is presented in the next section.

All the calculations that are to be done for obtaining the final result have as a basis the input data that consists of the coefficients of the same polynomial that was previously calculated also for the scheme A program for LFSR. The program contains two procedures and the main part. In the procedure *afis*, the results of each intermediate step are listed, so that it's easier to track back the results. The procedure afis is used in the main program, where it is called for each of the intermediate steps, in front of the second procedure of the program, called *prelucrare*. This second procedure is actually the most important one, as it calculates the intermediate results for the intermediate steps ending up with calculating the final results. The relations used in this procedure can be read from the scheme B for a LFSR. Below, the procedure is enclosed.

```
void prelucrare(int n, int knt)
{ for (int i=1;i<4;i++)
   a[i]=s[i-1];
  a[0]=(a[n-1]^s[n-1])^x[knt];
  k++;
 }
```

In the main section of the program, all the needed data is collected and also a counter is used for the elapsed time spent for the calculation of the final results. The program used for MISR for the scheme B is is explained next. Same as the input data for scheme A, this program gets the data like this: a table with 4 columns and 20 rows is read in the beginning of the program. The same structure of the program is applied here, like for the previous LFSR and MISR programs. This means that there are two procedures called afis and prelucrare, and there is also a main part of the program. In the procedure *afis*, the information at each intermediate step is listed, for easier understanding. This procedure is called-before the procedure prelucrare-and lists the results of the intermediary steps and also the final one. The procedure *prelucrare* is called in the main part of the program right after the procedure afis, being used for calculating both the intermediate steps and also for the final results. The main part of the program offers the same support, as a counter: how much time does it take? For exemplification, the code of the main part is:

```
void prelucrare(int n, int knt)
{ for (int i=1;i<n;i++)
   a[i]=s[i-1]^x[i][knt];
  a[0]=(s[n-1]^a[n-1])^x[0][knt];
  k++;
 }
```

Same as in the previous cases, the counter is started right after the calculation finished and ends when it is requested.

In the last case of the schema C calculating LFSR/MISR, the relations differ a bit. Overall, the program is organized in the same way as all the previous ones: two procedures and one main. The input data is the already known polynomial, made out of the same coefficients. The input data is read in the main program, afterwards the intermediate steps will be agreed. The procedure *afis* works closely especially with the procedure prelucrare. The calculation (in main) using parts of the input data is actually a cycle that for each step writes what was before and also what it is now, using the procedure afis and also prelucrare. The procedure prelucrare is enclosed:

```
void prelucrare(int n, int knt)
{ a[0]=s[n-1]^x[knt];
  a[1]=s[0]^a[0];
  for (int i=2;i<4;i++)
    a[i]=s[i-1];
  k++;
 }
```

The program used for this calculation corresponding to scheme C follows the template already established for the other programs.
Primarily, the procedures are afis and prelucrare, and they are both needed to get in the end to the final results. Same 4 columns on 20 rows are the input data for the obtaining of the next step results.
In the procedure prelucrare what is changed for this MISR scheme C is that the relations are following the scheme C for the step by step calculation and also the final result.

```
void prelucrare(int n, int knt)
{ a[0]=s[n-1]^x[0][knt];
  a[1]=x[1][knt]^(s[0]^a[0]);
  for (int i=2;i<n;i++)
    a[i]=s[i-1]^x[i][knt];
  k++;
 }
```

In the main part of the program both procedures are called to list and then calculate the intermediate step, ending with the final results. Before the calculation starts and after it had been finished, in the main part of the program is counted the elapsed time.

```
void main()
{ pf=fopen("lfsr_a1.txt","wb");
```

```
 for (int i=0;i<4;i++) s[i]=0;
 printf("\nIntroduceti coeficientii polinomului de intrare sub forma: a a a a a s.a.m.d.\n");
 scanf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",&x[0],&x[1],&x[2],&x[3],&x[4],&x[5],&x[6],&x[7],&x[8],&x[9],&x[10],&x[11],&x[12],&x[13],&x[14],&x[15],&x[16],&x[17],&x[18],&x[19],&x[20],&x[21],&x[22]);
 gettime(&timep0);
 k=1;
 for (i=0;i<24;i++)
 { afis(s,4);
   prelucrare(4,i);
   for (int j=0;j<4;j++) s[j]=a[j];
  }
 gettime(&timep1);
 long ora=timep1.ti_hour-timep0.ti_hour;
 long mint=timep1.ti_min-timep0.ti_min;
 long secn=timep1.ti_sec-timep0.ti_sec;
 float suts=timep1.ti_hund-timep0.ti_hund;
 ora*=3600;
 mint*=60;
 suts/=100;
 float suma=ora+mint+secn+suts;
 printf("\n suma de sec %f",suma);
 fclose(pf);
 getch();
}
```

The program for MISR is:

```
void main()
{ pf=fopen("misr_a1.txt","wb");
 for (int i=0;i<4;i++) s[i]=0;
 printf("\nIntroduceti cei 20 de x corespunzatori coloanei 0 in forma: a a a a  s.a.m.d.\n");
 for (i=0;i<4;i++)
   scanf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",&x[i][0],&x[i][1],&x[i][2],&x[i][3],&x[i][4],&x[i][5],&x[i][6],&x[i][7],&x[i][8],&x[i][9],&x[i][10],&x[i][11],&x[i][12],&x[i][13],&x[i][14],&x[i][15],&x[i][16],&x[i][17],&x[i][18],&x[i][19]);
 gettime(&timep0);
 k=1;
 for (i=0;i<21;i++)
 { afis(s,4);
   prelucrare(4,i);
   for (int j=0;j<4;j++) s[j]=a[j];
  }
 gettime(&timep1);
 long ora=timep1.ti_hour-timep0.ti_hour;
 long mint=timep1.ti_min-timep0.ti_min;
 long secn=timep1.ti_sec-timep0.ti_sec;
 float suts=timep1.ti_hund-timep0.ti_hund;
 ora*=3600;
 mint*=60;
 suts/=100;
 float suma=ora+mint+secn+suts;
 printf("\n suma de sec %f",suma);
 fclose(pf);
 getch();
}
```

3 Functioning of LFSR and MISR with 8 grade irreducible polynomials

First of all, the algorithm was applied using grade 4 polynomials.

The results were accurate and correct.

For each polynomial it was necessary to create three programs:

- one for simulating the use of LFSR
- one for another simulation with MISR
- another one for verifying the correctitude of the previous result.

Initially a program was specially developed to obtain all the irreducible grade 8 polynomial, thus substantially improving security [3].

In the following rows there is the description of the program used for obtaining all the Grade 8 irreducible polynomials.[21]

First of all, all the 8 grade polynomials were generated, but only that with natural coefficients. The general form for such polynomials is:

$$1\text{---}\ldots\text{-}1 \qquad (5)$$

because the first 1 is mandatory for having the established grade and the last one confirm the fact of being irreducible; if this is missing, than the polynomial can be divided by x.

For generating all the 8 grade polynomials it is necessary to work with 7 columns. The total number of polynomials will be z=128. For eliminating the reducible polynomials it is useful to use Z_2 presented in the next table:

Each of the 128 polynomials will be divided by the 12 irreducible polynomials. If one single rest is null, then the tested polynomials is dividing at least with one from the 12 irreducible polynomials. For a better verifying in the results table is maintained also the order number of the initial polynomial.

Out of the programs, it comes to the conclusion that the results should also rely on the previous link (given by the “equal with one” coefficient of the polynomial). This “specific” link that is also part of the calculation for MISR is to be taken after the XOR was made for that rank of the polynomial.

Tab. 1 The 12 irreducible polynomials for grade 1, 2, 3 and 4

Gr. 1	x	x
	1	$x + 1$
Gr. 2	2	$x^2 + 1$
	3	$x^2 + x + 1$
Gr. 3	4	$x^3 + x^2 + 1$
	5	$x^3 + x + 1$
Gr. 4	6	$x^4 + 1$
	7	$x^4 + x^3 + 1$
	8	$x^4 + x^2 + 1$
	9	$x^4 + x + 1$
	10	$x^4 + x^3 + x^2 + 1$
	11	$x^4 + x^2 + x + 1$
	12	$x^4 + x^3 + x^2 + x + 1$

Tab. 2 The 8 Grade irreducible polynomials

14.	1	1	0	1	1	0	0	0	1
15.	1	0	1	1	1	0	0	0	1
22.	1	1	0	1	0	1	0	0	1
23.	1	0	1	1	0	1	0	0	1
29.	1	0	0	1	1	1	0	0	1
32.	1	1	1	1	1	1	0	0	1
39.	1	0	1	1	0	0	1	0	1
48.	1	1	1	1	1	0	1	0	1
50.	1	1	0	0	0	1	1	0	1
51.	1	0	1	0	0	1	1	0	1
53.	1	0	0	1	0	1	1	0	1
57.	1	0	0	0	1	1	1	0	1
60.	1	1	1	0	1	1	1	0	1
62.	1	1	0	1	1	1	1	0	1
68.	1	1	1	0	0	0	0	1	1
70.	1	1	0	1	0	0	0	1	1
71.	1	0	1	1	0	0	0	1	1
80.	1	1	1	1	1	0	0	1	1
82.	1	1	0	0	0	1	0	1	1
85.	1	0	0	1	0	1	0	1	1
89.	1	0	0	0	1	1	0	1	1
95.	1	0	1	1	1	1	0	1	1
98.	1	1	0	0	0	0	1	1	1
104.	1	1	1	1	0	0	1	1	1
108.	1	1	1	0	1	0	1	1	1
111.	1	0	1	1	1	0	1	1	1
116.	1	1	1	0	0	1	1	1	1
122.	1	1	0	0	1	1	1	1	1
123.	1	0	1	0	1	1	1	1	1
125.	1	0	0	1	1	1	1	1	1

The mathematically representation of each rank of the polynomial was made accordingly to these "specific" links. Also, this new revealed thing was verified with the help of programs.

In the end, correlating the results obtained for the grade 4 polynomials with the results obtained for the 8 grade polynomials, it comes to mathematical relations for calculating each rank. These relations point out the previous existing links and act similarly to a feedback "calculated" also from the previous links.

The link acts after the XOR was calculated and in this way takes the result that was previously obtained.

In order to demonstrate that those presented above are correct and precise, was made an analysis for all the 30 irreducible grade 8 polynomials that were also found in another program specially developed for this purpose, coming to three particular cases.

Out of this analyze, there was made a generalization, that led to the writing of specific programs for each irreducible grade 8 polynomial, used for the substantially improvement of security [10].

The simulation programs were tested in both ways: with the method of making tables according to the proposed circuits and also with mathematical methods that materialize the hard operations.

As input data sets were used several different multiple combinations, randomly generated.

For building a security infrastructure the use of pseudorandom generators using packed matrices is very efficient [21]. Another possibility is to use Mersenne-Twister generator easy to find in MATLAB beginning with 7.4 version and in SCILAB.

The two programs are described in this paper, one of them simulating the functioning of a LFSR and the functioning of an analytic MISR, and the other one simulating the functioning of a synthetic MISR. The programs are based on irreducible grade 8 polynomials, allowing the user to introduce the coefficients of the chosen polynomial.

There have been chosen two polynomials for testing these two programs.

The first one is the polynomial

$$P(x)=x^8+x^6+x^5+x^3+1 \quad (6)$$

The coefficients would be introduced in the program as it follows:

$$1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \quad (7)$$

and would lead to the following scheme depicted in the next figure:

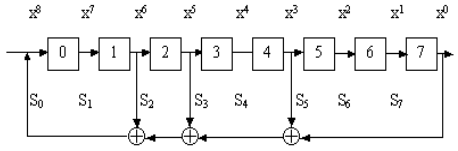

Fig. 8 Scheme for the polynomial $P(x)=x^8+x^6+x^5+x^3+1$

In the program the weights for each chosen polynomial are calculated. For this scheme, the weights are:

$$\begin{gathered} S_0=1\ P(x) \\ S_1=x\ P(x) \\ S_2=x^2\ P(x) \\ S_3=(x^3+x)\ P(x) \\ S_4=(x^4+x^2+x)\ P(x) \\ S_5=(x^5+x^3+x^2)\ P(x) \\ S_6=(x^6+x^4+x^3+x)\ P(x) \\ S_7=(x^7+x^5+x^4+x^2)\ P(x) \end{gathered} \quad (8)$$

The second one is the polynomial

$$P(x)=x^8+x^4+x^3+x+1 \quad (9)$$

used in Rijndael Cryptosystem [10]:

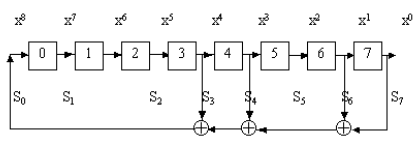

Fig. 9 Scheme for the polynomial

For the case in the "Fig. 9." scheme, the weights are:

$$\begin{gathered} S_0=1\ P(x) \\ S_1=x\ P(x) \\ S_2=x^2\ P(x) \\ S_3=x^3\ P(x) \\ S_4=x^4\ P(x) \\ S_5=(x^5+x)\ P(x) \\ S_6=(x^6+x^2+x)\ P(x) \\ S_7=(x^7+x^3+x^2)\ P(x) \end{gathered} \quad (10)$$

In order to get the final results from the programs, the user has to introduce in the program the polynomial's coefficients as described above and also the input data sets, consisting of 8 columns, each of them having a length of 2^n elements (where n is an integer). Because the new cryptographic Algorithms uses longer keys, now is important to improve the analysis of the functioning for shift registers of 16 and 32 [1],[3].

In the MISRSIN.CPP program the synthetic MISR is calculated for the input data given by the user and then the results are provided in the end. Calculating the results consists of categorizing the 8 SRi steps in which the scheme will be treated in 3 major types of ways according to the below described types:

- the first type is characterized by the existence of the coefficient of the i power corresponding to the SRi; in this case the procedure prelexi is called, having i+1 as an argument;
- the second type is characterized by the absence of the coefficient of the i power corresponding to the SRi; in this case the procedure prelabs is called, having i+1 as an argument;
- the third type is actually a particular one: it refers to the case of SR0, and the procedure prelzero is called, having no argument.

In the following rows the prelexi procedure is presented:

```
void prelexi(int ind)
{ int i,j,k,xor=0;
  for (i=0;i<8;i++)
for(j=0;j<51;j++) sr[i][j]=0;
  for (j=1;j<=n;j++)
  { xor=0;
    for (i=0;i<8;i++)
    { if ((ax[i]==1)&&
        ((ind-1)!=i))
            xor=xor^sr[i][j-1];
          if (i!=0)
```

```
          sr[i][j]=sr[i-1][j-1];
        }
   sr[ind][j]=col[ind][j-1]
^sr[ind-1][j-1];
   sr[0][j]=xor^sr[ind][j];
  }
 for (i=0;i<8;i++)
  rez[i][ind]=sr[i][n];
}
```

This time the basis is also the translation of the elements corresponding to the SRis, with i from 1 to 7 without ind (the argument of the procedure), by "connecting" the feedback in the SR0 while executing XOR with all the SRis that have a corresponding "connection" and with the current corresponding SRind, and by executing XOR with the element of the corresponding column and the element of the corresponding SRind. The code of the procedure prelabs is:

```
void prelabs(int ind)
{ int i,j,xor=0;
 for (i=0;i<8;i++)
for(j=0;j<51;j++)        sr[i][j]=0;
 for (j=1;j<=n;j++)
 { xor=0;
  for (i=0;i<8;i++)
  { if (ax[i]==1)
          xor=xor^sr[i][j-1];
        if (i!=0)
          sr[i][j]=sr[i-1][j-1];
        }
  sr[ind][j]=col[ind][j-1] ^sr[ind-1][j-1];
  sr[0][j]=xor;
  }
 for (i=0;i<8;i++)
  rez[i][ind]=sr[i][n];
}
```

Here is the same translation of the elements from SR1 to SR7 without ind (the argument of the procedure), by "connecting" the feedback in the SR0 while executing XOR with all the SRis that have a corresponding "connection" (the power i exists in the chosen polynomial) and by executing XOR with the element of the corresponding column and the element of the corresponding SRind.

The difference between prelabs and prelexi is given by the SRind: it is taken into consideration in the procedure *prelexi* in SR0, and does not appear in the procedure *prelabs* in SR0.

The final result is calculated by making XORs with all the partial results obtained in the 8 steps on each and every column of all the eight columns.

The *prelzero* procedure is reproduced below:

```
void prelzero)//SR0 calculation
{ int i,j,xor=0;
 for (j=1;j<=n;j++)
 { xor=0;
  for (i=0;i<8;i++)
  { if (ax[i]==1) //there is a //"connection"
          xor=xor^sr[i][j-1];
        if (i!=0)
          sr[i][j]=sr[i-1][j-1];
        }
  sr[0][j]=xor^col[0][j-1];
  }
 for (i=0;i<8;i++)
  rez[i][0]=sr[i][n];
}
```

In this procedure, the result is calculate by translating the elements corresponding to the SRis, with i from 1 to 7, and by "collecting" the feedback in the SR0 while executing XOR with all the SRis that have a corresponding "connection".

The analytic MISR and LFSR are calculated for the input data given by the user and the results are provided in the end of MISRCALC.

For the given polynomial, the program calculates in the procedure *genr* the corresponding 8 weights. The procedure *genr* is:

```
void genr()
{ //se calculeaza legaturile pentru polinomul
introdus
 for (int i=0;i<9;i++)
  for (int j=0;j<8;j++)
        s[j][i]=0;
//S0 e intotdeauna 1 pentru ca 8-r=8 (r e 0) si
x^8/x^8 e 1
 s[0][0]=1;
//r=8 nu are sens
 for ( i=1;i<8;i++)
  impart(i);
}
```

These weights are a base for calculation MISR and LFSR also. The whole procedure for the analytic MISR consists of 8 steps: for each step is considered the corresponding column which is multiplied by the corresponding weight, then it is divided by the chosen polynomial, again it is multiplied by x^8 and, finally, divided by the chosen polynomial. The result obtained in this way is the

result of the current step of the calculation for the analytical MISR. For each of the eight cases, the result is obtained as described above. The final result for the analytical MISR is obtained by making XORs with all the results of the 8 steps on each and every column of all the eight columns. The interpretation of this final result is that the ones and zeroes obtained are the coefficients of a polynomial. The grade of this polynomial may be any of those between seven and zero. The whole procedure for the LFSR is simpler: it is obtained by making XOR with all the weights multiplied by the corresponding column, and then, using the result obtained (whose grade gives the number of rounds that are to be done) as o column corresponding to the SR0 and no other columns, translating all the elements without 0, and "collecting" the feedback in the 0 element executing XOR with all the other elements that have a corresponding "connection" (the power with that rank exists in the chosen polynomial). In this case, the final result consisting of those eight figures (ones and zeroes) represents also the coefficients of a polynomial, just as in the case of MISR. The results obtained from the MISRS.CPP program and the other two results obtained from the MISRCALC.CPP program are the same, since they are the result of the same input data used for calculation that are equivalent. These kinds of calculation for verifying the correctness of the results have been made primarily on paper.
The next table contains all the 30 polynomials and the results of the tests. For this tests the input dates were provided random from a Pseudorandom Generator [9].

Tab. 3 The complete situation about all the 30 grade 8 irreducible polynomials

No.	Polynomial	Result
1	$x^8+x^4+x^3+x+1$	00110010
2	$x^8+x^4+x^3+x^2+1$	01101000
3	$x^8+x^5+x^3+x+1$	01011011
4	$x^8+x^5+x^3+x^2+1$	01101000
5	$x^8+x^5+x^4+x^3+1$	00111001
6	$x^8+x^5+x^4+x^3+x^2+x+1$	01100011
7	$x^8+x^6+x^3+x^2+1$	00010111
8	$x^8+x^6+x^4+x^3+x^2+x+1$	11101010
9	$x^8+x^6+x^5+x+1$	10001000
10	$x^8+x^6+x^5+x^2+1$	11010101
11	$x^8+x^6+x^5+x^3+1$	00011110
12	$x^8+x^6+x^5+x^4+1$	01011101
13	$x^8+x^6+x^5+x^4+x^2+x+1$	11100010
14	$x^8+x^6+x^5+x^4+x^3+x+1$	10010000
15	$x^8+x^7+x^2+x+1$	11101010
16	$x^8+x^7+x^3+x+1$	11001010
17	$x^8+x^7+x^3+x^2+1$	00001110
18	$x^8+x^7+x^4+x^3+x^2+x+1$	11001100
19	$x^8+x^7+x^5+x+1$	11110001
20	$x^8+x^7+x^5+x^3+1$	11011010
21	$x^8+x^7+x^5+x^4+1$	01010110
22	$x^8+x^7+x^5+x^4+x^3+x^2+1$	10111000
23	$x^8+x^7+x^6+x+1$	01010101
24	$x^8+x^7+x^6+x^3+x^2+x+1$	01110100
25	$x^8+x^7+x^6+x^4+x^2+x+1$	10101011
26	$x^8+x^7+x^6+x^4+x^3+x^2+1$	01110010
27	$x^8+x^7+x^6+x^5+x^2+x+1$	00101110
28	$x^8+x^7+x^6+x^5+x^4+x+1$	00100010
29	$x^8+x^7+x^6+x^5+x^4+x^2+1$	11010001
30	$x^8+x^7+x^6+x^5+x^4+x^3+1$	10001011

The results were obtained by running the programs for LFSR, MISR and for the mathematical method. The complete presentation for the 8 grade irreducible polynomials is presented in [22].

4 Functioning of LFSR and MISR with grade 16 irreducible polynomials

This program was developed based on the other one related to the Grade 8 Irreducible Polynomials [12]. Calling parameters for functions were extended to 2 bytes, so unsigned char became unsigned short. Another difference is at the column that is composed of 2 bytes. Power procedure returns the value of the most significant bit depending on the dimension given as argument. In the following will be useful to calculate the index for calling the function misr().

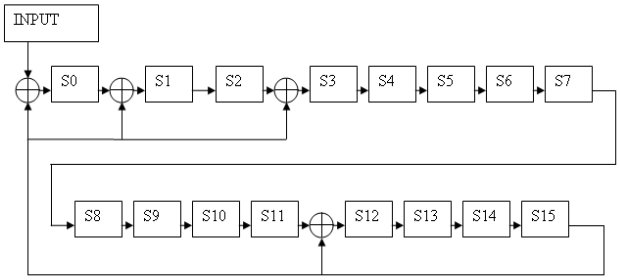

Fig. 10 Scheme A LFSR on 16 bits for the polynomial: $X^{16}+X^{12}+X^3+X+1$

The code for the program is presented below:
LFSR_16_A:

```
void prelucrare(int n,int knt)
{
a[0]=s[n-1]^x[knt];
a[1]=s[0]^s[n-1];
a[2]=s[1];
```

```
a[3]=s[n-1]^s[2];
for (int i=4;i<12;i++)
a[i]=s[i-1];
a[12]=s[n-1]^s[11];
for (int i=13;i<16;i++) a[i]=s[i-1];
k++;
}
int main(void)
{
pf=fopen("lfsr_al.txt","wb");
for (int i=0;i<16;i++) s[i]=0;
printf("\n introduceti coeficientii polinomului ex: a a a :\n");
scanf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",&x[0],&x[1],&x[2],&x[3],&x[4],&x[5],&x[6],&x[7],&x[8],&x[9],&x[10],&x[11],&x[12],&x[13],&x[14],&x[15],&x[16],&x[17],&x[18],&x[19],&x[20],&x[21],&x[22],&x[23]);
k=1;
time_t  start=clock(),end;
for (int i=0;i<24;i++)
{
afis(s,16);
prelucrare(16,i) ;
for (int j=0;j<16;j++) s[j]=a[j];
}
end=clock();
timpalocat=end;
printf("\n timpul alocat pentru lfsr_16_a este: %d microsecunde \n",timpalocat);
fclose(pf);
getch();
return 0;
}
```

For MISR Scheme A the functioning is simulated in the next program:

```
MISR_16_A:
FILE *pf;
int s[50],a[50],k ,kontor=0;
long timpalocat;
int x[16][20]={
{1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,1, 0, 1, 0},
{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,1, 0, 1, 0},
{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,1, 0, 1, 0},
{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,1, 0, 1, 0},
{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        };
void afis(int tab[],int n)
{
fprintf(pf,"\n%d.",k);
printf("\n%d.",k);
for (int i=0;i<n;i++)
{
printf("%d",tab[i]);
fprintf(pf,"%d",tab[i]);
}
}
void prelucrare(int n,int knt)
{
a[0]=s[n-1]^x[0][knt];
a[1]=x[1][knt]^(s[0]^s[n-1]);
a[2]=x[2][knt]^s[1];
a[3]=x[3][knt]^(s[n-1]^s[2]);
for (int i=4;i<12;i++) a[i]=s[i-1]^x[i][knt];
a[12]=x[12][knt]^(s[n-1]^s[11]);
for (int i=13;i<16;i++) a[i]=s[i-1]^x[i][knt];
k++;
}
int main(void)
{
pf=fopen("lfsr_al.txt","wb");
for (int i=0;i<16;i++) s[i]=0;
printf("\n introduceti coeficientii polinomului ex: a a a :\n");
/*for(int i=0;i<16;i++)
scanf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d ",&x[i][0],&x[i][1],&x[i][2],&x[i][3],&x[i][4],&x[i][5],&x[i][6],&x[i][7],&x[i][8],&x[i][9],&x[i][10],&x[i][11],&x[i][12],&x[i][13],&x[i][14],&x[i][15],&x[i][16],&x[i][17],&x[i][18],&x[i][19]);
*/
// initializare matrice de intrare
k=1;
time_t  start=clock(),end;
for (int i=0;i<20;i++)
{
afis(s,16);
prelucrare(16,i) ;
for (int j=0;j<16;j++) s[j]=a[j];
}
end=clock();
timpalocat=end;
printf("\n timpul alocat pentru misr_16_a este: %d microsecunde \n",timpalocat);
fclose(pf);
```

```
getch();
return 0;
}
```

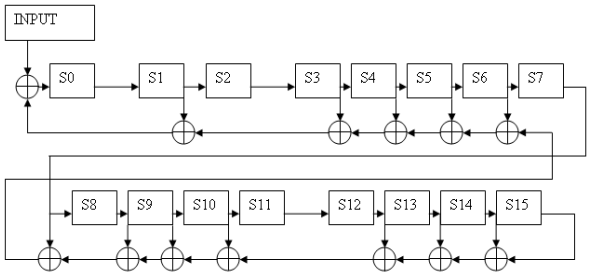

Fig. 11 Scheme B LFSR on 16 bits for the polynomial: $X^{16}+X^{12}+X^3+X+1$

Shift to the right:

```
for (int i=1;i<n;i++) a[i]=s[i-1];
a[0]=s[2]^s[4]^s[5]^s[6]^s[7]^s[8]^s[9]^s[10]^s[
11]^s[13]^s[14]^s[n-1]^x[knt];
```

The code for scheme B is:
LFSR_16_B:

In the following there are presented only the differences between the programs for scheme A, and the ones for scheme B and C.

```
void prelucrare(int n,int knt)
{
for (int i=1;i<n;i++) a[i]=s[i-1];
a[0]=s[2]^s[4]^s[5]^s[6]^s[7]^s[8]^s[9]^s[10]^s[
11]^s[13]^s[14]^s[n-1]^x[knt];
k++;
}
```

MISR_16_B:

```
void prelucrare(int n,int knt)//numerotare inversata
{
a[0]=s[0]^s[1]^s[3]^s[12]^x[0][knt];
for (int i=1;i<n;i++)
a[i]=s[i+1]^x[i][knt];
k++;
}
```

The function corresponding to Scheme C is:
LFSR_16_C:

```
void prelucrare(int n,int knt)
{
a[n-1]=s[0]^x[knt];
a[n-2]=a[n-1]^s[n-1];
a[n-3]=s[n-2];
a[n-4]=a[n-1]^s[n-3];
for (int i=n-5;i>=4;i--) a[i]=s[i+1];
a[3]=a[n-1]^s[4];
for (int i=2;i>=0;i--) a[i]=s[i+1];
k++;
}
```

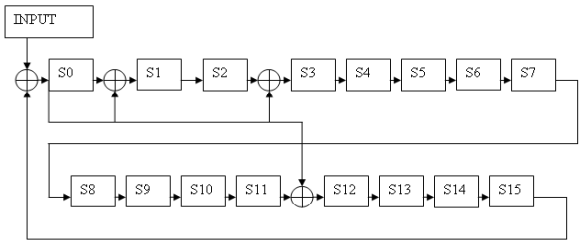

Fig. 12 Scheme C on 16 bits for the polynomial: $X^{16}+X^{12}+X^3+X+1$

MISR_16_C:

```
void prelucrare(int n,int knt)
{
 a[n-1]=s[0]^x[n-1][knt];
a[n-2]=a[n-1]^s[n-1]^x[n-2][knt];
a[n-3]=s[n-2]^x[n-3][knt];
a[n-4]=a[n-1]^s[n-3]^x[n-4][knt];
for (int i=n-5;i>=4;i--) a[i]=s[i+1]^x[i][knt];
a[3]=a[n-1]^s[4]^x[3][knt];
for (int i=2;i>=0;i--) a[i]=s[i+1]^x[i][knt];
k++;
}
```

After testing the programs with grade 16 irreducible polynomials the results for MISR and LFSR were identically.

5 Related Works

First of all it's well known that for maintaining a high level of security services is necessary to use a dependable source of random data [9].

One of the most interesting solutions is the Pseudorandom Generator using Packed Matrices. LFSR is one of the most popular elements for building cryptographic generator because it's easy to be implemented in hardware. The generator described in the mentioned paper is based on the power of a block upper triangular matrix (BUTM). The efficiency of using "packed matrices" comes from performing binary operations between processor registers. This BUTM can also be used for integrating a security kernel with many applications for low cost/ low power solutions.

For ensuring secure communications for low power devices there are some novel techniques [18]. After an interesting presentation of Data Encryption Algorithm DES AES and RSA (Rivest-

Shamir-Adleman) it was described the concept of unique key for each node. The main idea was to use the specific node properties, so that to obtain a maximizing of network security. Some experimental aspects and some results should be useful in this frame.

A very interesting point of view is to use a matricial public key cryptosystem with digital signature. [13] Again is very important to use the block upper triangular matrices (BUTM), this time based on a generalization of the discrete logarithm problem over a finite group. Confidentiality, integrity and availability are the main goals for security services. Also it's necessary to archive the authenticity of public keys and there are many ways for obtaining it. The Discrete Logarithm Problem(DLP) has a very good efficiency. After understanding the functioning of the symmetric component, the asymmetric one and the Hash one the applications proves how efficient and easy is to use them. Interesting is the fact that the implementation is almost as simple in hardware as it is in software. The conclusion is that using this discrete logarithm algorithm it will result a higher security level than on other algorithms used for the same key size.

Speaking about using hardware or software solutions for studying some cryptography methods rarely are used both of them.

One of the most efficient methods is modular multiplication [16]. Using repeatedly, this modular multiplication it obtains a modular exponentiation.
The efficiency of the implementation of modular exponentiation and multiplication determined the performance of public-key cryptosystems.
Because the plaintext is usually large as 1024 bits it was necessary to minimize the number of modular multiplications used so that this experiment take a single modular multiplication. For all the tests the efficiencies of the implementation is almost the same software and hardware. It was used and described the Karatsuba-Ofman's method for multiplying and the Barett's method for reducing.
These methods are very useful for studying the strengths and the weaknesses of a system.

Because a main problem in cryptosystems is to use LFSR with irreducible polynomials, it's interesting to find out the situation for an innovative scheme for LFSR reseeding with irreducible polynomials [20]. In fact is an investigation about the potential for effective reseeding of a built-in test pattern generation (TPG) mechanism. Application of this mechanism to pseudorandom test pattern generation demonstrated improvements over the classical approach of reseeding with a primitive characteristic polynomial.

A modern method in encoding and decoding is the use of convolutional code. A particular solution for obtaining this is Viterbi algorithm known as a maximum likelihood-decoding algorithm.[14]. The proposed model for decoder uses VDHL. Viterbi decoding motivate the use of VDHL. The produced code is more powerful when the constraint length used is larger. These were some of the most interesting news in the frame of the presented article.

6 Future Work

Studying the use of the LFSR in Galois Fields GF (2^n) it was demonstrated that for a better security is useful to operate with arithmetic's modulo n grade irreducible polynomial. This kind of situation is found in Reed Solomon and Rijndael codes. For Rijndael each coefficient in a bit, each element is a word of 4 bytes(32 bits).[15] It's possible to use convolutional codes, that are different than the codes using blocks, because they never have a constant length. There are special kinds of models for codifier. It was demonstrated that the convolutional encoding and decoding using Viterbi algorithm is a powerful method in error correction [14].

7 Conclusions

The analysis described in proves that a MISR has always the same results as a LFSR.

In the case of using LFSR, the speed increases very much, and it is well known that in using registers in cryptography an important goal is increasing the speed of calculations.

The aspect of security, as described in [3], was taken into consideration: the used polynomials are all irreducible grade 8 polynomials or 16.

Another important aspect presented in this paper is the discovery of the new formula for the calculation of the weights used for obtaining the final result of MISR.

This formula was tested for all the situations referring to grade 8 and 16 irreducible polynomials and the final conclusion is that the mathematical relations discovered are correct.

References:

[1] Landau S.: Communications Security for the Twenty-first Century : The Advanced

Encryption Standard, Notices of the American Mathematical Society, vol. 47, No. 4, 2000;

[2] Landau S.: Standing the Test of Time: The Data Encryption Standard, Notices of the American Mathematical Society, vol. 47, No. 3, 2000;

[3] Schneier B.:Applied Cryptology: Protocols, Algorithms, and Source Code in C, John Wiley and Sons, New York, 1996;

[4] Schneier B.: Bomba pentru criptare este amorsată, Byte, iunie 1998;

[5]H. Niederreiter, "A Public–Key Cryptosystem Based on Shift Register Sequences", Proceedings of EUROCRYPT'85, Linz, Austria 1985;

[6] A. Al–Yamani II, "Logic BIST: Theory, Problems, and Solutions", Stanford University, RATS/SUM02, 2002;

[7] Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002.

[8] M. Matsui, The First Experimental Cryptanalysis of the Data Encryption Standard, in Advances in Cryptology, Proc. Crypto'94, LNCS 839, Y. Desmedt, Ed., Springer-Verlag, 1994.

[9] Jose-Vicente Aguirre, Rafael Alvarez, Leandro Tortosa, Antonio Zamora: An Optimized Pseudorandom Genrator using Packed Matrices – WSEAS TRANSACTIONS on INFORMATION SCIENCE & APPLICATIONS Manuscript received Oct. 22, 2007; revised Mar. 22, 2008.

[10] Tariq Jamil, "The Rijndael Algorithm – A brief introduction to the new encryption standard," IEEE Potentials, Vol. 23, No. 2, April/May 2004;

[11] R. Sedaghat, B. O'Brien, " ASIC Implementation of a Pseudo-random Test Pattern Generator Using a 32-bit Linear Feedback Shift Register (LFSR)", Proc. International Conference for Upcoming Engineers, ICUE, 2003;

[12] Frank Tsui - LSI/VLSI Testability Design - McGraw-Hill Book Company, 1987, ISBN 0-07-065341-0;

[13] Rafael Alvarez, Francisco-Miguel Martinez, Jose-Francisco Vicent, and Antonio Zamora: A Matricial Public Key Cryptosystem with Digital Signature - WSEAS TRANSACTIONS on MATHEMATICS Manuscript received Nov. 28, 2007; revised March 17, 2008;

[14] Hema.S, Suresh Babu.V, Ramesh P: FPGA Implementation of Viterbi Decoder Proceedings of the 6th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Corfu Island, Greece, February 16-19, 2007;

[15] Ranjan Bose: An efficient method to calculate the free distance of convolutional codes - Proceedings of the 6th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Corfu Island, Greece, February 16-19, 2007;

[16] Nadia Nedjah, Luiza de Macedo Mourelle: A Review of Modular Multiplication Methods and Respective Hardware Implementations;

[17] Md. Mokammel Haque, Al-Sakib Khan Pathan, and Choong Seon Hong: Securing U-Healthcare Sensor Networks using Public Key Based Scheme;

[18] Gareth Howells, Evangelos Papoutsis, Klaus McDonald-Maier: Novel Techniques for Ensuring Secure Communications for Distributed Low Power Devices;

[19] Michael Scott: Optimal Irreducible Polynomials for GF(2^m) Arithmetic;

[20] Snehal Udar & Dimitri Kagaris: LFSR Reseeding with Irreducible Polynomials;

Wesley Peterson W.: Error-Correcting Codes, MIT Press, 1970.

[21] Mirella Amelia Mioc: Study of Using Shift Registers in Cryptosystems for Grade 8 Irreducible Polynomials; WSEAS Conference SMO 23-25 September 2008.