PSEUDO RANDOM NUMBER GENERATORS-LSFR

report submitted to

Indian institute of technology, madras

For the partial fulfilment of the course

of

Modelling in industry

by
PAWAN JIYALAL YADAV
SOWMIA.M
SHWETA SARAF
DIGANTO

Under the Supervision Of

Prof.S. Sunder

Department of Mathematics

May 14, 2020

Contents

ACKNOWLEDGEMENT

This project owes its existence to the help, support and inspiration of several people. It brings us an immense pleasure to express our deepest sense of gratitude to our project Supervisor Prof. S. Sunder, Department of Mathematics, Indian Institute of Technology, Madras for his eminent guidance and to give us the freedom for choosing *Pseudo Random Number Generators* as a project topic for modelling in industry. His wonderful social behaviour made him a visionary and inspiring teacher and an ideal supervisor for our team.

We are ineffably indebted to our Faculty Advisor-Prof. SivaramAmbikasaran, Department of Mathematics, Indian Institute of Technology, Madras for his invaluable direction, encouragement, support and for providing us all the necessary facilities for carrying out this work well before time.

Finally, our deepest gratitude goes to our family for their unflagging love and unconditional support throughout our life and studies

1. ABSTRACT

Nowadays security has become a great concern in the field of computer science and information technology. In order to protect data from unintended users and to achieve a desirable level of security, several cryptographic algorithms based on various technology have been proposed. Linear Feedback Shift Register (LFSR) may play an important role in the design of such cryptographic algorithms. LFSR based cryptographic algorithms are often lightweight in nature and are more suitable in environment where resources are constrained. In this paper, we present a detailed analysis of LFSR and design of r-sequence LFSR to implement cryptographic algorithms,namely for image encryption and decryption purposes

1.1 Key words

Galois Field $GF(p^m)$:Extension of the finite field$GF(p)$

Primitive Polynomial $Q(x)$:defined in the paper

Primitive Polynomial over $GF(p^m)$:defined in the paper

LSFR:Linear Shift Feedback register

m-sequence LSFR:An LFSR with maximal length sequence output

tap-sequence: the sequence$\{q_i\}$corresponding to the switch configuration of the LSFR

LCG:Linear congruential generator

Flip-flop:stores a bit value

XOR:mathematical equivalance is addition of two numbers mod 2

Characteristic Polynomial of a LSFR:The r^{th} degree monic polynomial with coefficients as the tap-sequence of a LSFR

2. Introduction and Motivation for PRNG

The paper is written to represent an introductory into concepts of Linear Feedback Shift Registers, Primitive Polynomial and Galois Field. Due to the simple structure and implementation of LFSR, it is widely used in network communication and industries for generating pseudo random sequences.

LFSR has the widest implementation in networking and cryptography.It is widely used in experimental design to get magnetic resonance imaging experiments, a pioneering non-invasive technology for studying the behavior of human brain and to generate Pseudo Noise (PN) or a Pseudorandom Sequences (PS).The functional magnetic resonance imaging experiment shows that the brain stimuli have correlated with the magnetic resonance scanner of the brain to collect functional MRI data for statistical analysis. An m-sequence LSFR can also be used to justify the order appearance and timing of the brain stimuli. Due to this promising work, primitive polynomial and m-sequence LFSRs have gained more acceptance in practice.

In this paper we present the linear congruential generator,the structure of LSFR, primitive polynomial over Galois Field and finally using random numbers generated by LSFR for image encryption purpose.

2.1 The Linear Congruential Generator

A linear congruential generator (LCG) is an algorithm that yields a sequence of pseudo-randomized numbers calculated with a linear recurrance relation. The method represents one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is relatively easy to understand, and they are easily implemented and fast,especially on computer hardware.The generator is defined by recurrence relation:

$$S_{i+1} = (AS_i + B) \bmod m$$

where X is the sequence of pseudorandom values, and
m, $0 < m$— the "modulus"
A, $0 < A < m$— the "multiplier"

B, $0 \leq B < m$ — the "increment"
S_0, $0 \leq S_0 < m$ — the "seed" or "start value"

A benefit of LCGs is that with appropriate choice of parameters, the period is known and long. Although not the only criterion, too short a period is a fatal flaw in a pseudorandom number generator.

2.2 Attacking LCG

We now consider a leakage where a malicious agent leaks a part of the intermediate internal secret state (say S_1, S_2, S_3) of the LCG.
The attacker does not have any knowledge of the location of leaked bits. in each algorithm execution.Now let us see if the algorithm can be attacked at a speed faster than brute-force

$$S_2 = (AS_1 + B) \bmod m$$
$$S_3 = (AS_2 + B) \bmod m$$

We have 2 linear equations with 2 unknowns,therefore the attacker can comfortable solve for these equations and get a closed form for A and B.Thus he can reproduce the entire stream cipher.

$$A = \frac{S_2 - S_3}{S_1 - S_2} \bmod m$$

$$B = S_2 - S_1 \frac{S_2 - S_3}{S_1 - S_2} \bmod m$$

2.3 Advantages And Disadvantages Of The LCG

- LCGs are fast and require minimal memory (one modulo-m number, often 32 or 64 bits) to retain state. This makes them valuable for simulating multiple independent streams.
- LCGs are not intended, and must not be used, for cryptographic applications as it is highly prone to attack; use a cryptographically secure pseudorandom number generator for such applications.

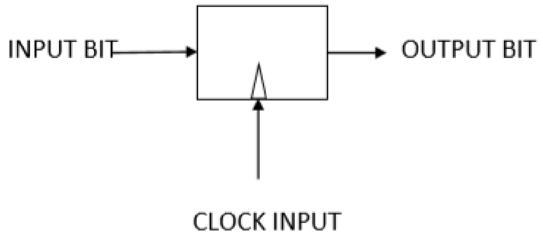
.

3. Electronic Model of an LSFR

Even though the LCG is quick ,requires negligible memory and is easy to implement on both the hardware and software,produces poor quality sequences, as the generated numbers S_i, and S_{i+1} are dependent thus the sequence is predictable, by knowing a portion of the sequence the attacker can constructs the whole sequence. Also Lower order bits of LCG have a shorter period as compared to the whole sequence.Can LSFRs achieve a extended periodic length?

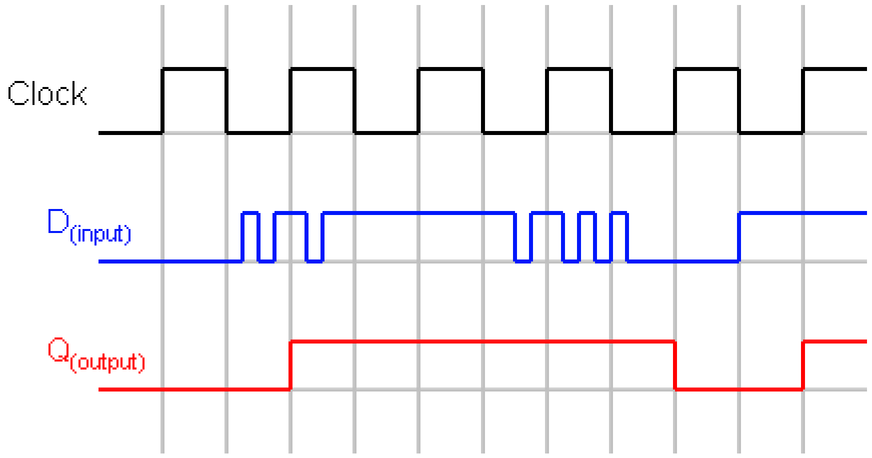
A linear feedback shift register (LFSR) is an n-bit shift register which combines some bits of it states in exclusive-OR to form a feedback, the position of bits that are selected to form the feedback is known as taps. LFSR is an appropriate choice to simulate truly random binary sequences and have the maximum length period of 2 $2^n 1$. Let us introduce the components of a LSFR one by one

3.1 Flip-Flop



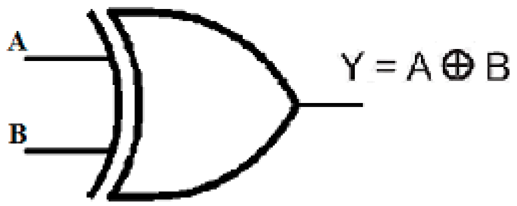
- a flip-flop or latch is a circuit that has two stable states(0 and 1) and can be used to store a bit of data
- It ignores the inputs except at the transition of a dedicated clock signal (known as clocking, pulsing, or strobing).
- Clocking causes the flip-flop either to change or to retain its output signal based upon the values of the input signals at the transition.

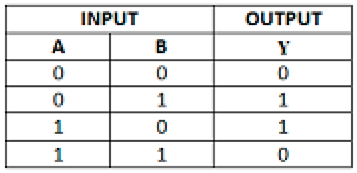
The diagram below depicts the working of a LSFR(its output) for the clock-input given



3.2 XOR-gate

A xor gate takes two input bits,calculates theire sum mod 2 and outputs the result





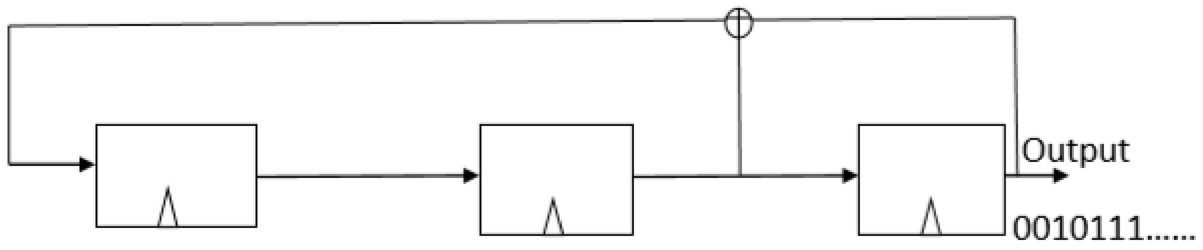
INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

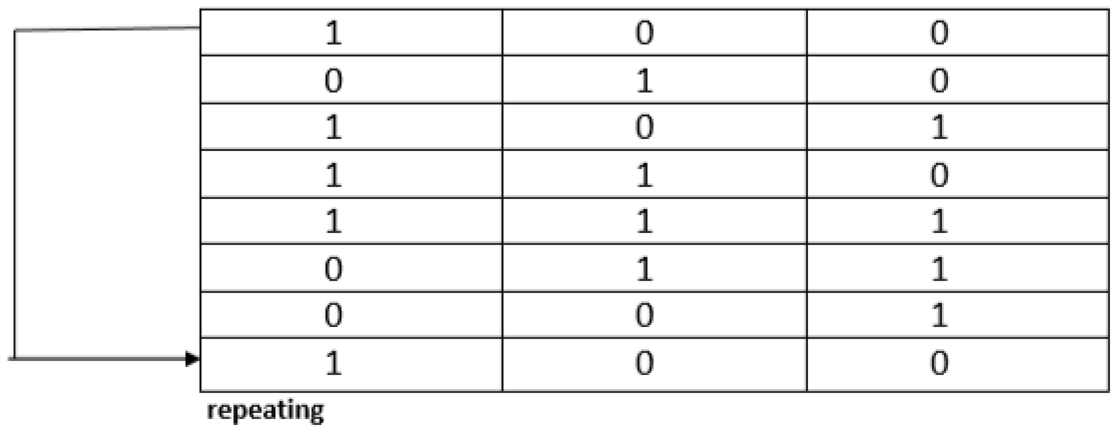
3.3 Examples

A 3 Flip-Flop LSFR

Now,we integrate both the components and investigate a LSFR with 3 flip-flops

We initialise the registers with bit values where not all are zero.Because in such a case the LSFR would just be producing a steady stream of zeroes which is of no use





1	0	0
0	1	0
1	0	1
1	1	0
1	1	1
0	1	1
0	0	1
1	0	0

repeating

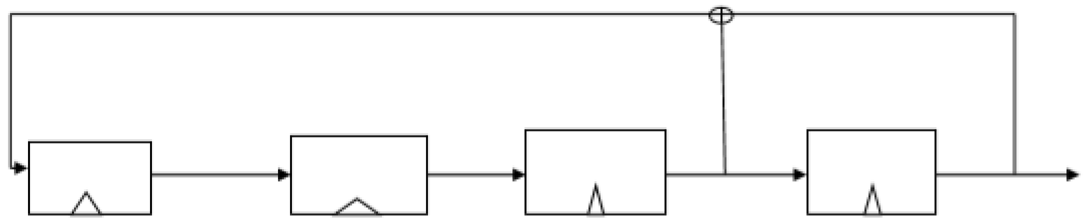
Thus the OUTPUT is going to be: 0010111 0010111 00101110010111......

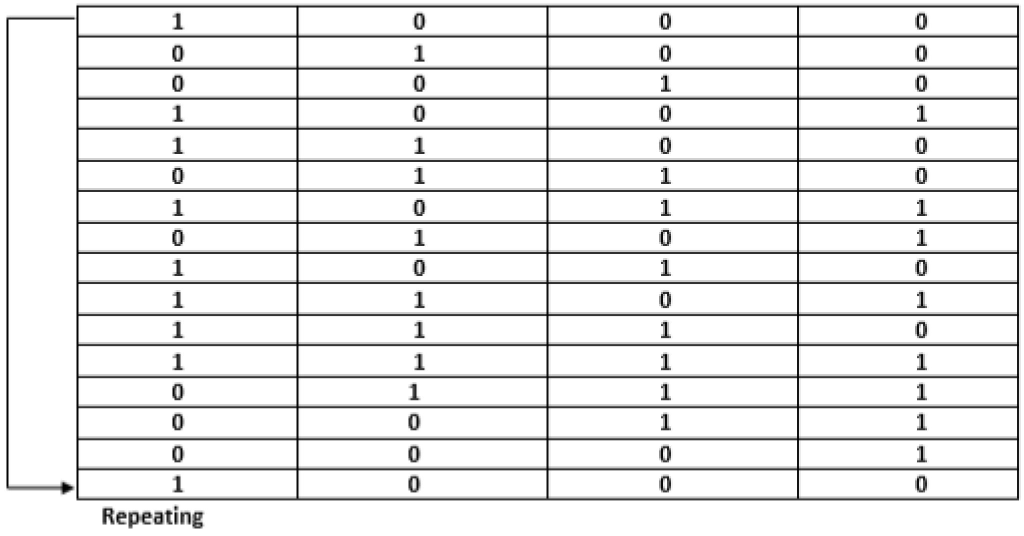
Since ,the sequence is repeating after every 7 bits produced,we say it has period:7.
Also note that this could be the maximum sequence period possible for a 3-flipflop - LSFR.The case where all registers have zeroes is impossible when the initialised values are not all zero

A 4 Flip-Flop LSFR

Having a output sequence with such a short period is not a good idea.So one obvious approach to increase the period is to increase the no of registers/flip-flops.Now let us investigate the output sequence of a 4-flipflop LSFR

EXAMPLE:1 m=4, XOR between 2nd and 3rd flipflops





1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	1
1	1	0	0
0	1	1	0
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1
1	0	0	0

Repeating

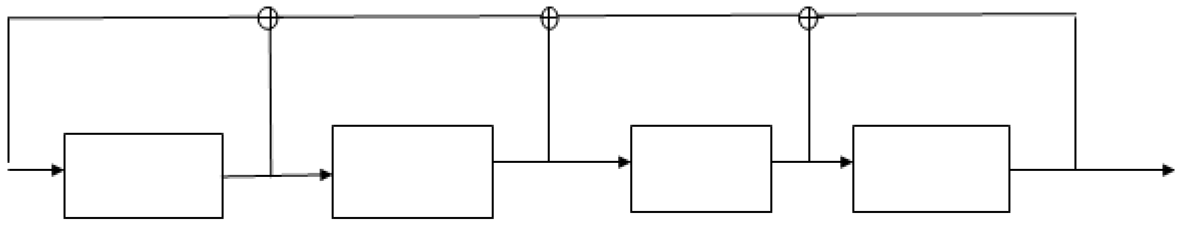
Thus the OUTPUT is going to be: $\underbrace{0001001101011110}$ $\underbrace{0001001101011110}$000100........

Since ,the sequence is repeating after every 16 bits produced, it has period:16.This is the maximum possible period for a 4-register LSFR.

The most crucial thing to note here is that t each flip –flop has a fair chance of getting fed back into the register.In the above two cases,we only tapped out values of the last 2 registers but other possiblities are still open.

Let us investigate the output sequence of a 4-register LSFR but with the feedback loop configuration altered

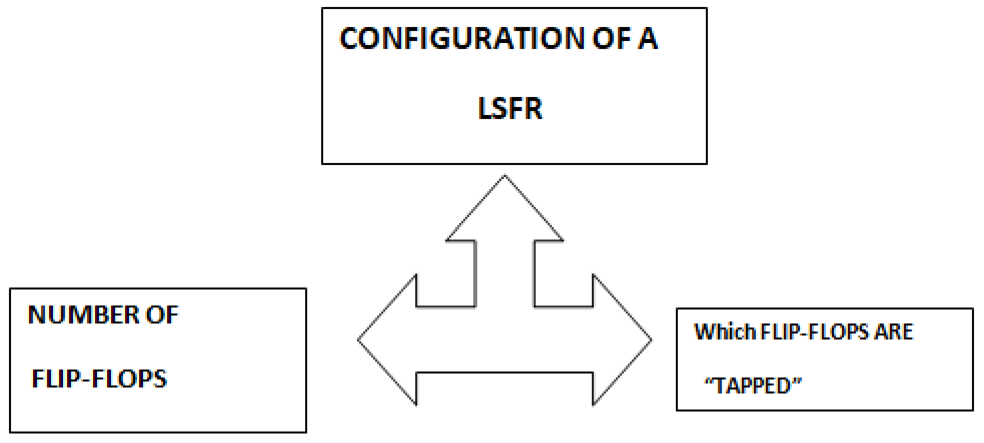
EXAMPLE:2 m=4,all flip-flops are tapped



1	0	0	0
1	1	0	0
0	1	1	0
0	0	1	1
0	0	0	1
1	0	0	0

Thus the OUTPUT is going to be: 00011|00011|00011 ………………

Since ,the sequence is repeating after every 5 bits produced, it has period:5
Thus we see,the period of the output also depends on the "taps" of the LSFR.Configuration of a LSFR is dictated by the number of flip-flops in it and the tap-sequence of the LSFR.



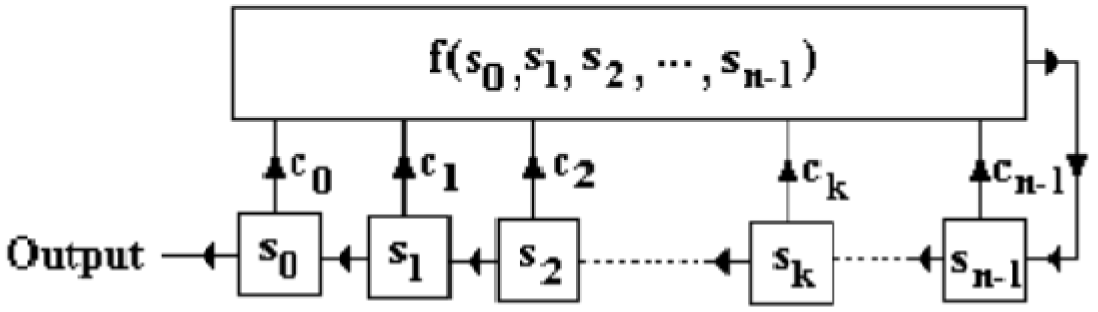
4. Mathematical Model of an LSFR

In this section ,we give the configuration of a LSFR interms of its characteristic polynomial Assume a generalised LSFR with n flip-flops.Since each flip-flop may/maynot be tapped,we put a switch for each as shown below.

Let

$$c_i = 0 \quad if \quad i^{th} \quad is \quad open$$
$$c_i = 1 \quad if \quad i^{th} \quad is \quad closed$$

Let the initial input bits be: $\{S_0, S_1, S_2 \ldots\ldots\ldots\ldots S_{n-1}\}$



Then the first output to be in the feedback loop is S_n given by:

$$S_n = f(S_0, S_1 \ldots\ldots S_{N-1}) = (S_0c_0 + S_1c_1 + S_2c_2 \ldots\ldots S_{n-1}c_{n-1}) \quad mod2$$
$$S_{n+1} = f(S_1, S_2 \ldots\ldots S_N) = (S_1c_0 + S_2c_1 + S_3c_2 \ldots\ldots S_nc_{n-1}) \quad mod2$$

Therefore in general we have,

$$S_{m+i} = \sum_{j=0}^{m-1} S_{i+j}c_j \quad mod2$$

4.1 characteristic polynomial of LSFR

The output of this LFSR is determined by the initital values $\{S_0, S_1, S_2 \ldots\ldots\ldots..S_{n-1}\}$ and the linear recursion relationship

$$S_{m+i} = \sum_{j=0}^{m-1} S_{i+j} c_j \quad mod2$$

A sequence produced by a length n LFSR which has period $2^n - 1$is called a **PN-sequence** (or a pseudo-noise sequence).We can characterize the LFSR's that produce PN-sequences and the whole of next section is dedicated it.

Definition 1. *We define the characteristic polynomial of an LFSR as the polynomial,*

$$f(x) = c_0 + c_1 x + c_2 x^2 + \ldots.. c_{n-1} x^{n-1} + x^n = \sum_{i=0}^{n} c_i x^i$$

where $c_n = 1$ by definition and $c_0 = 1$ by assumption.

The reason we have a polynomial depiction for LSFR is that it produces a PN-sequence if and only if its characteristic polynomial is a primitive polynomial.This statement will be proved in next section

5. LSFR Output Sequence Of Maximum Period

we state the following theorem that gives a method to generate a output sequence of maximum period ie to create a m-sequence LSFR

Theorem 1. *If the coefficients $q_0 \ldots \ldots, q_{r-1}$ of a linear shift register are chosen such that the polynomial*

$$Q(x) = x_r + q_{r-1}x^{r-1} + \ldots\ldots. + q_1 x + q_0$$

is primitive over F_2, then for all initial conditions in which not all a_i are zero, the sequence generated by the linear shift register will have a period of $2^r - 1$.

In order to prove the above theorem we will need to use of finite fields. In particular, we will need to make use of the field F_{2^r} containing 2^r elements.

5.1 The structure of extension field F_{2^r}

The elements of F_{2^r} are the r-tuples $(b_0, ..., b_{r-1})$, where $b_i \in \{0, 1\}$.

5.1.1 Addition Of Two Elements

The addition of two such r-tuples is simply addition modulo 2, performed entry by entry

$$(b_0, ..., b_{r1}) + (c_0, ..., c_{r-1}) = (d_0, ..., d_{r1})$$

$$where \quad d_i = b_i + c_i (mod 2)$$

.

5.1.2 Multiplication Of Two Elements

To define a multiplication operator we start by choosing an irreducible polynomial

$$P(x) = x^r + p_{r-1}x^{r-1} + ... + p_1x + p_0$$

over the field F_2.
We interpret each r-tuple $(b_0, ..., b_{r1})$ as a polynomial of degree less than or equal to r-1 ie $b_{r-1}x^{r-1} + ... + b_1x + b_0$. In order to multiply the two r-tuples we multiply the two associated polynomials. The product is a polynomial of degree less than or equal to 2(r-1), which is then reduced to a polynomial in x of degree r-1 by taking its modulo P(x).

SOME RESULTS ON FIELD F_{2^r}
1) The set F_{2^r} together with addition and multiplication as defined above is a field.
2)There exists an element α such that the nonzero elements of F_{2^r} are precisely the elements α^i for $i = 0, ..., 2^{r-2}$. In other words,

$$F_{2^r}/\{0\} = \{1, \alpha, \alpha^2, ..., \alpha^{2r-2}\}$$

An element α satisfying the above property is called a ***primitive root of the field*** F_{2^r}, and satisfies $\alpha^{2^r-1} = 1$.
what guarantees the existance of such a element??
Non-zero elements of F_{2^r} form a multiplicative subgroup G.
Every non-zero element generates a finite sub-group H . |H| must divide |G|
Since H is finite,there must be a min s, s.t $y^s = 1$ where s= |H| $\implies$ y is root of $x^s + 1 = 0$
$\implies$ any element of G is root of $x^{2^r-1} + 1 = 0$

5.2 Primitive polynomials

Definition 2. *A polynomial*

$$Q(x) = x^r + q_{r-1}x^{r-1} + ... + q_1x + q_0$$

with coefficients $q_i \in F_p$ is primitive if and only if it is irreducible and the set of nonzero elements of F_{p^r}, where

$$F_{p^r} = \{b_0 + b_1x + ... + b_{r-1}x^{r-1} | b_i \in F_p\}$$

is of the form $F_{p^r}/\{0\} = \{x^i | i = 0, ..., p^{r-2}\}$, where the powers x^i are taken modulo Q(x)

Thus a polynomial Q(x) with coefficients in F_2 is called primitive if it is irreducible and if the polynomial x is a primitive root of the field F_{2^r} constructed with respect to Q(x).

5.3 Weaker Form of The Theoerem

Theorem 2. *Given a linear shift register , there exist coefficients $q_0, ..., q_{r-1} \in \{0, 1\}$ and initial conditions $a_0, ..., a_{r-1} \in \{0, 1\}$ such that the sequence generated by the register has a period of length $2^r - 1$.*

Definition 3 (TRACE FUNCTION)**.** *The trace function is the function*

$$T : F_{2^r} \to F_2$$

given by

$$T(b_{r-1}x^{r-1} + ... + b_1x + b_0) = b_{r-1}$$

.

Initialising LSFR using the Trace function
We choose a primitive polynomial P(x) over F_2

$$P(x) = x^r + q_{r-1}x^{r-1} + ... + q_1x + q_0$$

The q_i of the linear shift register are the coefficients of the polynomial P(x). In order to initialise,we choose any nonzero polynomial

$$b = b_{r-1}x^{r-1} + ... + b_1x + b_0$$

from $F_{2^r}$and define the initial conditions as follows

$$\begin{aligned} a_0 &= T(b) = b_{r-1} \\ a_1 &= T(xb) \\ &\vdots \\ a_{r-1} &= T(x^{r-1}b) \end{aligned}$$

$$\begin{aligned} a_1 &= T(bx) \\ &= T(b_{r-1}x^r + b_{r-2}x^{r-1} + ... + b_0x) \\ &= T(b_{r-1}(q_{r-1}x^{r-1} + ... + q_1x + q_0) + b_{r-2}x^{r-1} + ... + b_0x) \\ &= T((b_{r-1}q_{r-1} + b_{r-2})x^{r-1} + ...) \\ &= b_{r-1}q^{r-1} + b_{r-2} \end{aligned}$$

Similarly we compute other $a_i s$

Lemma 1 (LEMMA:1)**.** *Let us choose the coeffcients* $q_0, ..., q_{r-1}$ *of a shift register as those of a primitive polynomial*

$$P(x) = x^r + q_{r-1}x^{r-1} + ... + q_1x + q_0$$

Let $b = b_{r-1}x^{r-1} + ... + b_1x + b_0$*.Choose the initial elements* $a_0, ..., a_{r-1}$ *as discussed above. Then the sequence* $\{a_n\}_{n\geq 0}$ *generated by the shift register is given by* $a_n = T(x^n b)$*, and it repeats with a period that divides* $2^r - 1$*.*

Proof.

$$P(x) = 0 \implies x_r = q_{r1}x^{r1} + ... + q_1x + q_0.$$

Then

$$\begin{aligned} T(x^r b) &= T((q_{r-1}x^{r-1} + ... + q_1x + q_0)b) \\ &= q_{r-1}T(x^{r-1}b) + ... + q_1T(xb) + q_0T(b) \\ &= q_{r-1}a_{r-1} + ... + q_1a_1 + q_0a_0 \qquad (5.1) \\ &= a_r \end{aligned}$$

Recall that **(5.1)** is the feedback loop value calculated at the first clock pulse by the LSFR. We proceed by induction. Suppose now that the elements of the sequence satisfy
$a_i = T(x^i b) \quad for \quad i \leq n-1$. Then

$$\begin{aligned} T(x^n b) &= T(x^r x^{n-r} b) \\ &= T((q_{r-1}x^{r-1} + .. + q_1x + q_0)x^{n-r}b) \\ &= q_{r-1}T(x^{n-1}b) + ... + q_1T(x^{n-r+1}b) + q_0T(x^{n-r}b) \\ &= q_{r-1}a_{n-1} + ... + q_1a_{n-r+1} + q_0a_{n-r} \qquad (5.2) \\ &= a_n \end{aligned}$$

Recall that **(5.2)** is the feedback loop value calculated at the n^{th} clock pulse by the LSFR.

Thus multiplication by x corresponds exactly to the calculation performed by the shift register, and therefore $a_n = T(x^n b)$ **for all n. We see immediately that the minimal period has length at most** $2^r - 1$**, since** $x^{2^r-1} = 1$ □

This proves the weaker form of the theorem as well

5.4 Stronger Form Of The Theorem

Theorem 3 (STRONGER FORM)**.** *If the coefficients* $q_0. . . . . ., q_{r-1}$ *of a linear shift register are chosen such that the polynomial*

$$Q(x) = x_r + q_{r-1}x^{r-1} + ... + q_1x + q_0$$

is primitive over F_2, then for all initial conditions in which not all a_i are zero, the sequence generated by the linear shift register will have a period of $2^r - 1$.

To prove this theorem ,we state and prove the following lemma

Lemma 2 (LEMMA:2)**.** *let Q(x)be a irreducible polynomial*

$$Q(x) = x^r + q_{r-1}x^{r-1} + ... + q_1x + q_0$$

Then for any nonzero sequence $(a_0, ..., a_{r-1})$, there exists a unique $b = b_0 + b_1x + ... + b_{r-1}x^{r-1}$ such that $a_i = T(bx^i) \quad for \quad i = 0, ..., r-1$.

Proof. consider the linear system of equations $T(bx^i) = a_i \quad , for \quad i = 0, ..., r-1$ where $\{b_i\}_{i=0}^{r-1}$ are unknowns

$$\begin{aligned} T(b) &= b_{r-1} = a_0 \\ T(bx) &= b_{r-2} + q_{r-1}b_{r-1} = a_1 \\ T(bx^2) &= b_{r-1}(q_{r-1})^2 + b_{r-2}q_{r-1} + b_{r-3} = a_2 \end{aligned}$$

Thus given a non-zero sequence $(a_0, ..., a_{r-1})$,we can uniquely determine $\{b_i\}_{i=0}^{r-1}$ by the above method.This shows that we can uniquely determine b given $(a_0, ..., a_{r-1})$ □

Let us now prove stronger form of the theorem

Proof. If the coefficients$\{q_i\}$ of LSFR are such that

$$Q(x) = x_r + q_{r-1}x^{r-1} + ... + q_1x + q_0$$

is primitive,then we already proved

- the existence of $(a_0, ..., a_{r-1})$ such that output sequence has period $2^r - 1$(weaker form)
- In LEMMA:2 , we also proved that for every nonzero sequence $(a_0, ..., a_{r-1})$ there exists a unique $b = b_0 + b_1x + ... + b_{r-1}x^{r-1}$ such that $a_i = T(bx^i)$, for $i = 0, ..., r-1$
- LEMMA:1 tells us that if $\{a_n\}$ is a sequence generated by a linear shift register with initial conditions $a_i = T(bx^i)$, then for all n it holds that $a_n = T(bx^n)$.
- Since $x^{2^r-1} = 1$,$\{a_n\}$ is periodic, and for all n it holds that $a_n = a_{n+2^r-1}$.Hence the proof

□

6. Simulation of LSFR

6.1 PART:I Setting Up The Code

First, we write a library LFSR.h that will simulate an LFSR using the following API:

public class LFSR

---

public static LSFR(string seed,bool *tap) // creates LFSR with the given initial seed and tap-sequence
public static bool step()// simulate one step and return the least significant (rightmost flip-flop's) bit as 0 or 1
public static int generate(int k) // simulate k steps and return k-bit integer
public static void display() // prints the content of LSFR's flip-flops currently

6.2 PART:II Initialising The LSFR

The constructor takes the initial seed as a String argument whose characters are a sequence of 0s and 1s. The length of the register is the length of the seed. The tap bit is the position of the bit that will be used (along with the highest order bit) in order to generate the next bit in the step() function.

6.2.1 Error Checking

The seed or tap sequence may be invalid for a number of reasons:

- The seed is not composed entirely of 0s and 1s.
- The tap sequence contains negative nos.

- The tap position is greater than or equal to the seed length.

If these cases, the LFSR's internal state will not be updated.The LSFR class starts out in a state in which step and generate should produce a steady stream of 0s, and a failed call of the constructor should not change that.

6.3 C++ Code

```
#ifndef LSFR_H
#define LSFR_H
#include<iostream>
#include<string>
using namespace std;

struct Node
{
bool data;
Node *next;
Node(bool x)
{
this->data=x;
next=NULL;
}
};

class LSFR
{
private:
string seed;
bool *tap;//pointing to the tap sequence array
Node *head; //to point to the first elemt of flipflop
Node *tail; //to point to last element of flip-flops
public:
LSFR(string,bool*);
void display();//output lsfr values in a string
bool step();//output for one clockcycle
long generate(int);//takes an integer-k and returns a k-bit random number
};
```

```
//defining constructor of LSFR
LSFR::LSFR(string seed,bool *tap)
{
int m=seed.length();
bool flag=0;
for(int i=0;i<m;i++)
{
if(seed[i]=='1')//atlest one bit is 1 ..valid seed
flag=1;
if(seed[i]!='1' && seed[i]!='0') //non binary values not allowed
flag=0;
}
//if seed is valid ,initialize lsfr with the seed bit values
if(flag==1)
{
head =new Node(seed[0]-'0');
Node*trvlptr=head;
for(int i=1;i<=m-1;i++)
{
trvlptr->next=new Node(seed[i]-'0');
trvlptr=trvlptr->next;
}tail=trvlptr;
this->seed=seed;
}
//if seed is invalid,initilise lsfr with all zeroes
else
{
string temp="";
for(int i=0;i<m;i++)temp+='0';
this->seed=temp;
head =new Node(0);
Node*trvlptr=head;
for(int i=1;i<=m-1;i++)
{
trvlptr->next=new Node(0);
trvlptr=trvlptr->next;
}tail=trvlptr;
}
//assign the tap values to respective flipflops in lsfr
this->tap=tap;
}
```

//**member function that displays the state of the flip-flops currently**

```
void LSFR::display()
{
Node *trvlptr=head;
while(trvlptr!=NULL)
{
cout«trvlptr->data;
trvlptr=trvlptr->next;
}
}
```

//**member function that outputs the LSFR's output for first clockimpulse**

```
bool LSFR::step()
{

//calculating the feedback loop value
int i=0;
Node*trvlptr=head;
int result=0;
while(trvlptr!=NULL)
{
if(tap[i]==1)
{
result+=trvlptr->data;
}
trvlptr=trvlptr->next;
i++;
}
result=result%2;
/now inserting the feedback loop value in first flip-flop
Node*temp=head;
head=new Node(result);
head->next=temp;

//storing the last flip-flop val and deleting the last element
bool res=tail->data;
trvlptr=head;
while(trvlptr->next->next!=NULL)trvlptr=trvlptr->next;
temp=trvlptr->next;
trvlptr->next=NULL;
tail=trvlptr;
delete temp;

//return output of lsfr
return res;
}
```

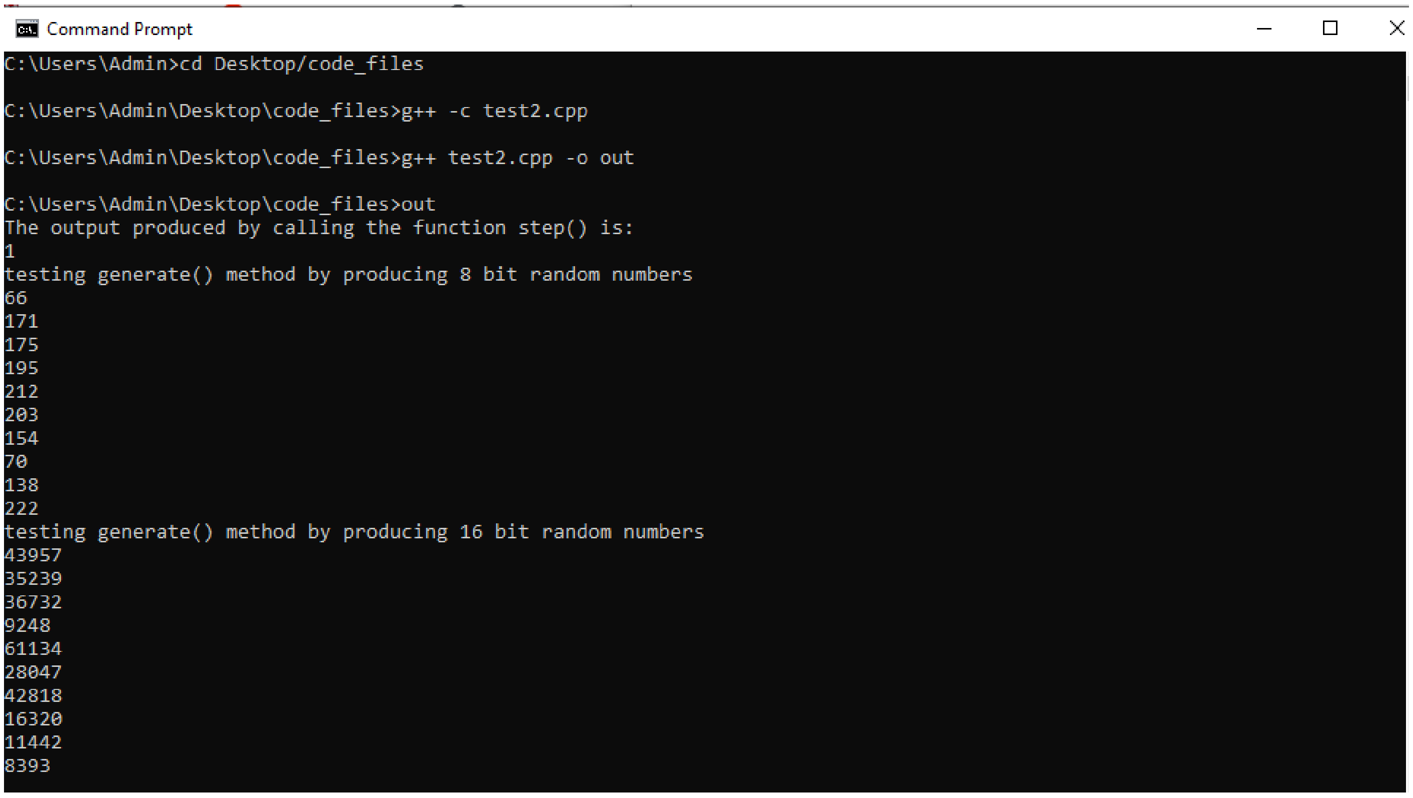
//member fucntion that outputs a k-bit random number

```
long LSFR::generate(int k)
{
long result=0;
int count=1;
for(int i=0;i<k;i++)
{
result+=(this->step())*count;
count=2*count;
}
return result;
}
#endif
```

6.4 PART:III Testing The Code

```
#include<iostream>
#include "LSFR.h"
#include<string>
using namespace std;
int main()
{
bool array[]={0,0,0,0,0,1,0,1,0,0,1,1};
string seed="011010000101";
LSFR obj(seed,array);
//single ouptput by lsfr
cout«"The output produced by calling the function step() is:"«endl;
cout«obj.step()«endl;
//producing 10 random 8 bit numbers
cout«"testing generate() method by producing 8 bit random numbers"«endl;
for(int i=0;i<10;i++)
cout«obj.generate(8)«endl;
//producing 10 random 16 bit numbers
cout«"testing generate() method by producing 16 bit random numbers"«endl;
for(int i=0;i<10;i++)
cout«obj.generate(16)«endl;
return 0;
}
```

OUTPUT PRODUCED



```
C:\Users\Admin>cd Desktop/code_files

C:\Users\Admin\Desktop\code_files>g++ -c test2.cpp

C:\Users\Admin\Desktop\code_files>g++ test2.cpp -o out

C:\Users\Admin\Desktop\code_files>out
The output produced by calling the function step() is:
1
testing generate() method by producing 8 bit random numbers
66
171
175
195
212
203
154
70
138
222
testing generate() method by producing 16 bit random numbers
43957
35239
36732
9248
61134
28047
42818
16320
11442
8393
```

7. Using PRNG produced by LSFR for image encryption and decryption

7.1 PART:I Setting up The Code

We now write a LFSR client PhotoMagic.h file that can encrypt and decrypt images.we will treat images as a 2-D array of integers, where each integer encodes a pixel color and transparency. The first dimension of the array is the row, and the second dimension is the column. To access a specific pixel, we therefore write array[row][col].If we prefer to think in terms of (x,y) coordinates, then the equivelant statement is array[y][x] (note the inversion of y and x). PhotoMagic.h provides the following API that we can use to load, encrypt and decrypt images:

public class PhotoMagic

---

public static PhotoMagic();//default constructor -creates a PhotoMagic object
Public static PhotoMagic(ifstream &);//constructor that takes ifstream object by reference to read from file and create the 2-D array of image
Public static void read(ifstream &);//reads the data file and stores in input_data
Public static void encrypt(string,string);//encrypts the input data and stores it outfile_encrypted
void decrypt(string,string);//decrypts the encrypted and writes to outfile_decrypted

7.2 C++ Code

```
#ifndef PHOTOMAGIC_H
#define PHOTOMAGIC_H
#include<fstream>
#include<sstream>
#include<vector>
#include"LSFR.h"
using namespace std;
```

```
class PhotoMagic{
private :
vector<vector<int»input_data;
vector<vector<int»encrypted;
ofstream outfile_encrypted;
ofstream outfile_decrypted;

public:
PhotoMagic();
PhotoMagic(ifstream &);
void read(ifstream &);//reads the data file and stores in input_data
void encrypt(string,string);//encrypts the input data and stores it outfile_encrypted
void decrypt(string,string);//decrypts the encrypted and writes to outfile_decrypted
};
```

//**default constructor**

```
PhotoMagic::PhotoMagic()
{}
```

constructor with ifstream object as parameter

```
PhotoMagic::PhotoMagic(ifstream &infile)
{
vector<vector<string»data;
string s,t;
while(infile)
{

if(!getline(infile,s))
break;
istringstream ss(s);
vector<string>temp;

while(ss)
{

if(!getline(ss,t,',;))
break;
temp.push_back(t);
}
data.push_back(temp);
}
if(!infile.eof())
{
cerr«"End of file";
}
int x;
for(int i=0;i<data.size();i++)
{
vector<int>temp;
for(int j=0;j<data[0].size();j++)
{
t=data[i][j];
stringstream number(t);
x=0;
number»x;
temp.push_back(x);
}
input_data.push_back(temp);
}
}
```

//**read() that explicitly initialises inputdata when not done by the constructor at object initilaisation**

```
void PhotoMagic::read(ifstream & infile)
{
vector<vector<string»data;
string s,t;
while(infile)
{

if(!getline(infile,s))
break;
istringstream ss(s);
vector<string>temp;

while(ss)
{

if(!getline(ss,t,',',))
break;
temp.push_back(t);
}
data.push_back(temp);
}
if(!infile.eof())
{
cerr«"End of file";
}
int x;
for(int i=0;i<data.size();i++)
{
vector<int>temp;
for(int j=0;j<data[0].size();j++)
{
t=data[i][j];
stringstream number(t);
x=0;
number»x;
temp.push_back(x);
}
input_data.push_back(temp);
}
}
```

encrypt() method that encrypts the array elements one by one

```
void PhotoMagic::encrypt(string filename,string seed)
{
//now we encrytp by XOR-ing the data with a different newly-generated,random 16-bit integer
returned by
LFSR.generate(16)
bool array[]={0,0,0,0,0,1,0,1,0,0,1,1}; //primitve polynomial of degree 12
LSFR lsfr_en("011010000101",array); //12 bit seed
int x;
for(int i=0;i<input_data.size();i++)
{
vector<int>temp;
for(int j=0;j<input_data[0].size();j++)
{
x=lsfr_en.generate(16);
temp.push_back(x xor input_data[i][j]); }
encrypted.push_back(temp); }
//we now write the encrypted data to a new text file
outfile_encrypted.open(filename);
for(int i=0;i<encrypted.size();i++)
{
for(int j=0;j<encrypted[0].size()-1;j++)
{
outfile_encrypted«encrypted[i][j]«’,’;
}
outfile_encrypted«encrypted[i][encrypted[0].size()-1]«endl;
} outfile_encrypted.close(); }
```

decrypt() method that decrypt the encrypted array elements one by one

```
void PhotoMagic::decrypt(string filename,string seed)
{
bool array[]={0,0,1,0,0,0,0,0,0,0,1};
LSFR lsfr_de("01101000010",array);
vector<vector<int»decrypted;
int x;
for(int i=0;i<encrypted.size();i++)
{
vector<int>temp;
for(int j=0;j<encrypted[0].size();j++)
{
x=lsfr_de.generate(16);
temp.push_back(x xor encrypted[i][j]);
}
decrypted.push_back(temp);
}
//we now write the encrypted data to a new text file


outfile_decrypted.open(filename);
for(int i=0;i<decrypted.size();i++)
{
for(int j=0;j<decrypted[0].size()-1;j++)
{
outfile_decrypted«decrypted[i][j]«’,’;
}
outfile_decrypted«decrypted[i][decrypted[0].size()-1]«endl;
}
outfile_decrypted.close();
}
```

```
#endif
```

7.3 PART -II Testing the Code

I read the following image and convert it to a grayscale image and store the pixel values in a file.

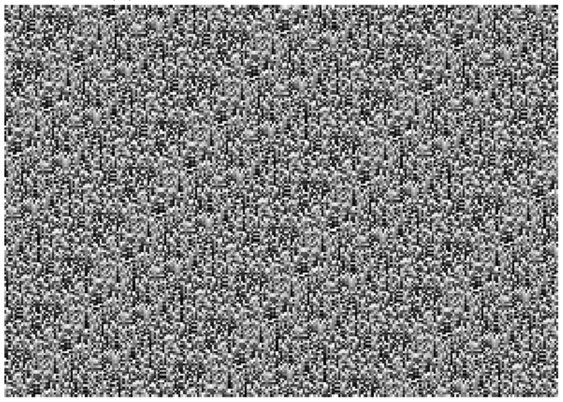
Matlab code to read the image into a 2-D array

```
I=imread(’elephant.jpg’)
imshow(I)
title(’Input Image’)
A=rgb2gray(I);
figure,imshow(A)
title(’gray converted Image’)
fid = fopen(’eleph_in.txt’,’wt’);
for ii = 1:size(A,1)
fprintf(fid,’%g
t’,A(ii,:));
fprintf(fid);
end
fclose(fid)
```

Matlab code to convert the encrypted matrix into image

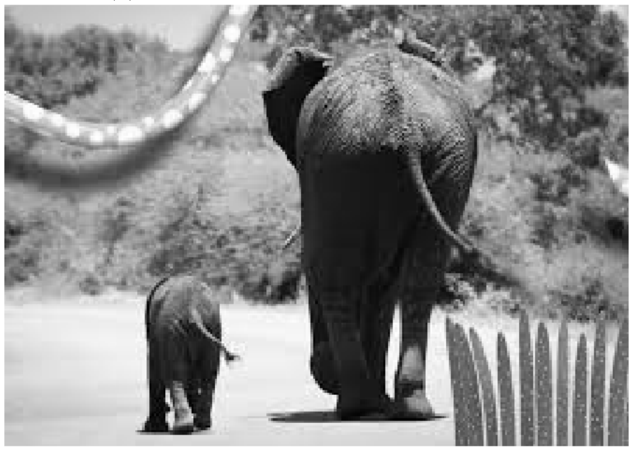
```
M=dlmread("eleph_encrypted.txt");
I = mat2gray(M);
figure
```

```
imshow(I)
```



Matlab code to convert the decrypted matrix into image

```
M=dlmread("eleph_decrypted.txt");
I = mat2gray(M);
figure
imshow(I)
```



8. Conclusion

The paper presented a brief analysis and design of LFSR based m-sequence PRNG that are very useful to implement cryptographic primitives. Some important mathematical tools such as Field, Galois Field, Primitive Polynomial and Primitive Polynomial over Galois Field and LFSR have been discussed.LSFRr is a effective approach for generating random numbers on the basis of user's seed (keyword). By using C++ program, its performance was measured on a 3GHz Pentium4 with 1GB of RAM running Windows 10 professional Version 2010, Service pack 3. Results conclude that LSFR can generate any count of random numbers against the user's key. Its implementation is quite simple and performance of proposed approach is awesome

Bibliography

[1] Wayne Tomasi “Electronic Communications System Fundamentals through Advanced . 5th edition, Pearson Education, 2008
[2]Harsh kumar verma and Ravindra kumar singh "Linear Feedback Shift Register Based Unique Random Number Generator ", Dr.B.R. Ambedkar National Institute of Technology
[3]christiane Rousseau"Mathematics And Technology",3rd edition,springer