Abstract— In this paper we present a pseudorandom number generator using Elliptic curves and the concept of multiplicative congruential generator. We describe a method of generating a sequence of points on Elliptic curves over a finite field.

Index Terms— Elliptic curves, multiplicative congruential method, pseudorandom numbers, statistical tests.

I. INTRODUCTION

Random Number Generation is an essential part of many applications involving security. In statistics, a random number is a single observation of a specified random variable. Truly random numbers are rarely used in computation, because it is difficult to generate the same sequence again. The lack of reproducibility would make validation of programs that use these numbers extremely difficult. Pseudorandom numbers are computed from a mathematical formula or simply taken from a pre-calculated list. Pseudorandom numbers play a vital role in several fields ranging from science and technology to recreation.

A. Elliptic Curves

A cubic equation of form \( y^2 = x^3 + a x^2 + b x + c \) is called an Elliptic Curve. Using suitable transformation of the coordinates this can be expressed as \( y^2 = x^3 + a x + b \), called the standard form. Here \( a \) and \( b \) are fixed constants, \( x \) and \( y \) vary over \( R \) or \( C \) or \( Q \) or a finite field \( F_p \), where \( p \) is a prime. We add a special point, \( O \), to the curve called the point at infinity. The set of all points on \( E \) denoted by \( E (F_p) \), forms an abelian group with \( O \), the point at infinity serving as the identity element with respect to addition [1].

\[ y^2 = x^3 - x \]

\[ y^2 = x^3 - x + 1 \]

Fig 1 Elliptic curves

B. Addition Operation

Let \( P_1(x_1, y_1) \) and \( P_2(x_2, y_2) \) be two points such that \( P_1 \neq P_2 \neq O \)

1. If \( x_1 \neq x_2 \), \( x_3 = m^2 - x_1 - x_2 \), \( y_3 = m(x_1 - x_3) - y_1 \), where

   \[ m = (y_2 - y_1)/(x_2 - x_1) \]

2. If \( x_1 = x_2 \) but \( y_1 \neq y_2 \), \( P_1 + P_2 = O \).

3. If \( P_1 = P_2 \) and \( y_1 \neq 0 \), \( x_3 = m^2 - 2x_1 \) and

   \[ y_3 = m(x_1 - x_3) - y_1 \]

   \[ m = (3x_1^2 + a)/2y_1 \]

4. If \( P_1 = P_2 \) and \( y_1 = 0 \) then \( P_1 + P_2 = O \)

Karuna Kamath K, Shankar B R

Karuna Kamath K, NMAM Institute of Technology, Nitte 574 110, Udupi Dist, Karnataka, India, karunapandit@gmail.com

Shankar B R, National Institute of Technology Karnataka, Surathkal 575 025, shankarbr@gmail.com, brs@nitk.ac.in
With this definition of addition, the set of all points on elliptic curve over $F_p$ forms an abelian group, with 'O' as additive identity.

C. LINEAR CONGRUENTIAL GENERATOR (LCG)
A Linear Congruential Generator (LCG) represents one of the oldest and best-known pseudorandom number generators [2]. The theory behind them is easy to understand, and they are easily implemented and fast.

The generator is defined by a recurrence relation:

$$X_{n+1} = (a \cdot X_n + c) \mod m$$

where $X_n$ is the sequence of pseudorandom values, and $m$, $0 < m$, is the modulus $a$, $0 < a < m$, the multiplier $c$, $0 \leq c < m$, the increment $X_0$, $0 \leq X_0 < m$, the seed or initial value are integer constants that specify the generator.

The LCG will have a full period for all seed values if and only if

1. $c$ and $m$ are relatively prime,
2. $a-1$ is divisible by all prime factors of $m$,
3. $a-1$ is a multiple of 4 if $m$ is a multiple of 4.

While LCGs are capable of producing decent pseudorandom numbers, they are extremely sensitive to the choice of the parameters $c$, $m$, and $a$ [3]. Various choices of $c$, $m$, $a$ and $X_0$ are given in [4].

If $c = 0$, the generator is often called a Multiplicative Congruential Generator (MCG), or Lehmer random number generator (RNG). If $c \neq 0$, the generator is called a mixed congruential generator [3].

If $m$ is prime, $a$ is a primitive element modulo $m$, and $X_0 \neq 0$, then the generated sequence will have period of length, $m - 1$, and the generator is called a full period MCG.

The multiplicative congruential generator can be generalized to get the extended congruential generator, in which each new integer is a linear combination of previous $n$ integers [5]:

$$X_{n+1} = a_1 X_1 + a_2 X_2 + a_3 X_3 + \ldots + a_n X_n \mod p$$

$a_1, a_2, a_3, \ldots, a_n$ are constants.

This concept is used to generate pseudorandom numbers using elliptic curves over a finite field $F_p$.

II. PROPOSED GENERATOR

1. Select an Elliptic curve say, $E: y^2 = x^3 + x +1$ over a field $F_p$, where $p$ is some fixed prime.
2. Choose $n$ points say, $P_1, P_2, \ldots, P_n$ on $E$.
3. Select $n$ integers $a_1, a_2, a_3, \ldots, a_n$ such that $a_i < \text{order of } P_i$ for all $i$, $1 \leq i \leq n$.
4. Generate sequence of points

$$P_{n+1} = a_1 P_1 + a_2 P_2 + a_3 P_3 + \ldots + a_n P_n \mod E.$$}

The extended multiplicative congruential generator seems to produce sequences with fairly long period from simple multiplicative congruential generators with short periods. For example, consider the elliptic curve $E: y^2 = x^3 + x +1; p=101$, $n=1$. With seed $(27, 4)$ and multiplier 2, the random sequence generated has the period 12 and with seed as $(28, 8)$ and multiplier 3, the sequence has period 2. But the combination of these two seeds with respective multipliers, $2 \cdot (27, 4) + 3 \cdot (28, 8)$ resulted in a sequence with a relatively long period of 144. Similarly, $2 \cdot (93, 17)$ generated a sequence with period 12, and $95 \cdot (95, 48)$ also generated a sequence with period 12, but the combination, $2 \cdot (93, 17) + 95 \cdot (95, 48)$ could generate a sequence with period 144.
Table 1 seeds and period of sequences generated with n=2

<table>
<thead>
<tr>
<th>Sl.No</th>
<th>First generator &amp; period</th>
<th>Seed</th>
<th>Second generator &amp; period</th>
<th>Seed</th>
<th>Period of the combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(82,30) 2 12</td>
<td>(83,3) 3 4</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(30,8) 85 4</td>
<td>(32,28) 17 4</td>
<td>210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(93,17) 2 12</td>
<td>(95,46) 95 12</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(99,30) 98 6</td>
<td>(100,4) 59 4</td>
<td>410</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 seeds and period of sequences generated with n=3

<table>
<thead>
<tr>
<th>Sl.No</th>
<th>First generator &amp; period</th>
<th>Second generator &amp; period</th>
<th>Third generator &amp; period</th>
<th>Period of the combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15*(32,28) 12</td>
<td>5*(35,17) 6</td>
<td>10*(36,43) 4</td>
<td>195</td>
</tr>
<tr>
<td>2</td>
<td>15*(83,3) 4</td>
<td>5*(84,11) 6</td>
<td>2*(85,38) 12</td>
<td>176</td>
</tr>
<tr>
<td>3</td>
<td>7*(86,34) 6</td>
<td>11*(87,24) 2</td>
<td>65*(88,35) 2</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>19*(25,20) 4</td>
<td>23*(27,4) 12</td>
<td>41*(28,8) 6</td>
<td>195</td>
</tr>
</tbody>
</table>

Table 3 seeds and period of sequences generated with n=4

<table>
<thead>
<tr>
<th>Sl.No</th>
<th>First generator &amp; period</th>
<th>Second generator &amp; period</th>
<th>Third generator &amp; period</th>
<th>Fourth generator &amp; period</th>
<th>Period of the combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2*(23,24) 12</td>
<td>2*(25,20) 12</td>
<td>2*(27,4) 12</td>
<td>28*(28,8) 6</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>2*(74,17) 6</td>
<td>2*(76,39) 6</td>
<td>2*(79,21) 12</td>
<td>99*(82,30) 6</td>
<td>205</td>
</tr>
<tr>
<td>3</td>
<td>5*(8,4) 6</td>
<td>8*(10,1) 4</td>
<td>12*(11,38) 12</td>
<td>75*(12,23) 3</td>
<td>296</td>
</tr>
<tr>
<td>4</td>
<td>5*(60,27) 2</td>
<td>8*(61,46) 4</td>
<td>12*(62,43) 12</td>
<td>40*(64,65) 12</td>
<td>190</td>
</tr>
</tbody>
</table>

Table 4 seeds and period of sequences generated with n=5

<table>
<thead>
<tr>
<th>Sl.No</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Period of the combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2*(59,45) 12</td>
<td>2*(60,27) 4</td>
<td>2*(61,46) 12</td>
<td>2*(62,43) 12</td>
<td>27*(64,35) 2</td>
<td>275</td>
</tr>
<tr>
<td>2</td>
<td>2*(92,24) 6</td>
<td>2*(93,17) 12</td>
<td>2*(95,48) 12</td>
<td>2*(99,30) 12</td>
<td>29*(100,10) 2</td>
<td>290</td>
</tr>
<tr>
<td>3</td>
<td>12*(74,17) 6</td>
<td>17*(76,39) 2</td>
<td>19*(79,21) 2</td>
<td>26*(82,30) 2</td>
<td>87*(83,3) 3</td>
<td>320</td>
</tr>
<tr>
<td>4</td>
<td>12*(79,21) 12</td>
<td>17*(82,30) 4</td>
<td>19*(83,3) 2</td>
<td>26*(84,11) 2</td>
<td>29*(85,38) 2</td>
<td>350</td>
</tr>
</tbody>
</table>

Table 5 seeds and period of sequences generated with n=6

<table>
<thead>
<tr>
<th>Sl.No</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Generator &amp; period</th>
<th>Period of the combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2*(59,45) 12</td>
<td>2*(60,27) 4</td>
<td>2*(61,46) 12</td>
<td>2*(62,43) 12</td>
<td>27*(64,35) 2</td>
</tr>
<tr>
<td>2</td>
<td>2*(92,24) 6</td>
<td>2*(93,17) 12</td>
<td>2*(95,48) 12</td>
<td>2*(99,30) 12</td>
<td>29*(100,10) 2</td>
</tr>
<tr>
<td>3</td>
<td>12*(74,17) 6</td>
<td>17*(76,39) 2</td>
<td>19*(79,21) 2</td>
<td>26*(82,30) 2</td>
<td>87*(83,3) 3</td>
</tr>
<tr>
<td>4</td>
<td>12*(79,21) 12</td>
<td>17*(82,30) 4</td>
<td>19*(83,3) 2</td>
<td>26*(84,11) 2</td>
<td>29*(85,38) 2</td>
</tr>
</tbody>
</table>
A. RANDOMNESS
Random and pseudorandom numbers generated for cryptographic applications should be unpredictable. In the case of PRNGs, if the seed is unknown, the next output number in the sequence should be unpredictable in spite of any knowledge of previous random numbers in the sequence. This property is known as forward unpredictability. It should also not be feasible to determine the seed from knowledge of any generated values i.e., backward unpredictability. Each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2.

In order to gain confidence that such generators are secure, they should be subjected to a variety of statistical tests designed to detect the specific characteristics expected of a random sequence. The passing of these statistical tests is a necessary but not sufficient condition for a generator to be secure.

B. Generator Sequence Properties
A good pseudorandom sequence generator worthy of consideration for encryption purposes is characterized by the following properties [6]:

- Long period: The generator should have long enough periods in order to avoid the recurrence of the sequence after a short length of time.
- High linear complexity: Sequences with low linear complexity are easily predictable.
- Reproducibility: Same sequence should be produced for a given seed.
- Statistical Properties: The generator must pass a battery of statistical tests to validate its randomness attributes [7].

The sequence produced by the proposed generator was analyzed and tested for conformance to the above characteristics. This analysis is important to check the significance of the sequence produced and thus ascertain confidence in its use for key stream generation.

III. ANALYSIS

A. Long Period
The generated sequence has a long period as desired. The results are illustrated in tables 1 to 5.

B. Linear Complexity
One measure of the strength of a random sequence is its linear complexity, as studied by various authors [8], [9], [10], [11], [12]. The linear complexity of a binary sequence is defined as the length of the shortest linear feedback shift register (LFSR) that generates it. If a sequence has small linear complexity, then the production of the sequence becomes computationally feasible. The linear complexity of a finite sequence is determined using Massey-Berlekamp algorithm [11]. It is desirable that a random sequence which can be used as key sequence in stream cipher systems should have a large linear complexity.
The linear complexity is an important concept in the analysis of stream ciphers. Any sequence produced over a finite field has a finite linear complexity [13]. The linear complexity of an infinite sequence’s is
- Zero if ‘s’ is a zero sequence
- $\infty$ if no LFSR generates ‘s’
- Length of shortest LFSR that can generate ‘s’

Table 6  Linear complexities of the generated sequences

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>229</td>
<td>417</td>
<td>457</td>
<td>452</td>
<td>578</td>
<td>621</td>
</tr>
<tr>
<td></td>
<td>337</td>
<td>426</td>
<td>480</td>
<td>494</td>
<td>614</td>
<td>658</td>
</tr>
<tr>
<td></td>
<td>563</td>
<td>479</td>
<td>494</td>
<td>510</td>
<td>645</td>
<td>724</td>
</tr>
<tr>
<td></td>
<td>653</td>
<td>513</td>
<td>526</td>
<td>528</td>
<td>689</td>
<td>712</td>
</tr>
<tr>
<td></td>
<td>881</td>
<td>562</td>
<td>542</td>
<td>568</td>
<td>710</td>
<td>780</td>
</tr>
</tbody>
</table>

Table of Linear Complexity of Binary Sequences Generated using extended multiplicative congruential generator for various values of primes and different values of $n$ using the elliptic curve $y^2 = x^3 + x + 1$.

Fig 2 linear complexity for different values of prime
C. Statistical Properties

The NIST test suite was applied to pseudorandom sequence produced by the generator. This test suite is used as a benchmark by NIST in the evaluation of possible candidate generators for the Advanced Encryption Standard, AES [14]. The test suite conducts a comprehensive battery of statistical tests, in which there are 16 core test strategies. The proposed generator successfully passes the test suite, and thus strengthens the confidence in these generators. For the battery of tests the level of significance \( \alpha \) is set to 0.01. An evaluated \( p \)-value > \( \alpha \) signifies that the sequence is random with a confidence of 99% [7]. The NIST test suite also has a routine to aid the analysis of results. It generates a report file containing the proportion of sequences that pass the test. In Fig. 3 the proportion of successful sequences for each test is plotted against the test ID number.

![Fig 3 NIST Test results for \( p = 101 \).](image)

IV. CONCLUSION

The extended multiplicative congruential method is simple and easy to implement. Experimental results show that the periods of such generators are much longer than those that use the congruential method. Thus, in any Elliptic curve Cryptosystem, pseudorandom numbers can be easily generated without much overhead.

REFERENCES