p-adics in Cryptography: FCSRs and Their Use

Vedanth Chakravarthi and Dhyutidhar Sravankumar

May 2026

Abstract

This article explores two types of pseudo-random number generators used in cryptography: Linear Feedback Shift Register (LFSR) and Feedback with Carry Shift Register (FCSR). LFSR can be predicted using a rational function of polynomials, and FCSR can be predicted using the 2-adic expansion of a rational number in $\mathbb{Z}_2$.

1 CSPRNGs

When messages are sent online, they have to be encrypted (turned into a code) to prevent being intercepted. However, they must also be readable by the recipient of the message, who must be able to decrypt the code. This ensures that from the moment the message leaves the sender to the moment it reaches the recipient, it is scrambled and unreadable.

One of the most widely used methods of encryption is known as public key encryption. In this system, each user has a pair of keys, one visible to everyone, and one secret key. The public key can encrypt messages, but is very difficult to reverse, while the private key can decrypt the public key's code but is kept secret by the owner. Another type is symmetric encryption, in which the recipient and the sender have identical keys to encrypt and decrypt messages. Although this method works quickly and is quite simple, the hard part is distributing the keys to the messengers securely.

Often, there is a hybrid combination of both methods known as HPKE, where public key is used to distribute symmetric keys to a pair of users, which are then used for all communication between the two parties.

Both of these methods require unpredictable keys in order to prevent attackers from guessing or reconstructing the key, even if they get hold of a small continuous sample of the key. This is why a method to generate this randomness must be constructed.

2 LFSRs

One common way to generate pseudo-random numbers is by using an LFSR. An LFSR, or Linear Feedback Shift Register, is a filled array of r cells, $a_0, a_1, \cdots a_{r-1}$,

each with a starting value of 0 or 1. Each cell also has a 'tap' value of either 0 or 1, which is fixed to the position of the cell. We say a cell is 'tapped' if its tap value is 1. Each step, the values in the array are each shifted 1 space to the right, so a_1 goes to where a_0 used to be, and so on. The rightmost value, a_0, 'falls off the edge' of the LFSR and is taken as output. There is also an empty cell at the left-side edge. To calculate a new value for this cell (a_r), we take the mod 2 sum (XOR) of the bits inside the tapped cells. As the LFSR keeps running, it spits out a_0, a_1, and goes on to produce an infinite sequence of pseudo-random bits.

How can we predict the behavior of an LFSR? To start, note that all math is done in $\mathbb{F}_2$, the finite field with 2 elements (this is equivalent to arithmetic in base 2). Let's define the r taps on the array as $q_0, q_1, \cdots, q_{r-1}$. For each a_n where $n \geq r$, we can express it as the sum of the tapped values of the previous r terms, or

$$a_n = q_0 a_{n-1} + q_1 a_{n-2} + \cdots + q_{r-1} a_{n-r}.$$

Since addition is the same as subtraction in $\mathbb{F}_2$, we can move all the terms to one side to get

$$a_n + \sum_{i=0}^{r-1} q_i a_{n-i-1} = 0,$$

for all $n \geq r$.

We can express the infinite output sequence as a formal power series with coefficients in $\mathbb{F}_2$:

$$A(X) = \sum_{n=0}^{\infty} a_n X^n.$$

The goal now is to multiply this infinite power series by some finite polynomial in X to get a finite number of terms. This might seem like a counter-intuitive step, but it will hopefully be made clear when this method is put to use in an example.

We define

$$C(X) = 1 + q_0 X + q_1 X^2 + \cdots + q_{r-1} X^r.$$

When expand the product $A(X) \cdot C(X)$, we get:

$$A(X)C(X) = (a_0 + a_1 X + a_2 X^2 \cdots)(1 + q_0 X + q_1 X^2 + \cdots + q_{r-1} X^r).$$

Let's look closely at the coefficient on X^n, for $n \geq r$. The coefficient is:

$$a_n + q_0 a_{n-1} + q_1 a_{n-2} + \cdots q_{r-1} a_{n-r} = a_n + \sum_{i=0}^{r-1} q_i a_{n-i-1},$$

which, as previously established, is always 0 given $n \geq r$. So the only remaining nonzero terms of the product make a (finite) polynomial with degree less than r. Let this polynomial be $P(X)$. We have

$$A(X)C(X) = P(X),$$

or

$$A(X) = \frac{P(X)}{C(X)},$$

where $A(X)$ represents the output sequence of 'random' bits while $P(X)$ and $C(X)$ are polynomials with degree r or lower. This means that the sequence that an LFSR generates is just a ratio of polynomials, each with a relatively low degree.

Example 1. We will now walk through the steps of a mini-LFSR. It will have a 3-bit shifting register, with taps at the second and third positions and starting load $(1, 0, 1)$. So $a_0 = 1, a_1 = 0, a_2 = 1$, and $q_0 = 0, q_1 = 1, q_2 = 1$.

For any $n \geq 3$, we can write a recurrence relation:

$$a_n = a_{n-2} + a_{n-3},$$

or

$$a_n + a_{n-2} + a_{n-3} = 0.$$

We define the formal power series $A(X) = \sum_{n=0}^{\infty} a_n X^n$, and $C(X) = 1 + q_0 X + q_1 X^2 + q_2 X^3 = 1 + X^2 + X^3$. Note that to shift the LFSR by i steps forward, we use the power series $X^i \cdot A(X)$. When we multiply $A(X) \cdot C(X)$, we get the sum of the all $a_n X^n$ terms, plus all of the $a_{n-2} X^n$ terms, plus all of the $a_{n-3} X^n$ terms. When all of the X^n terms are grouped together, for each n there is a form $X^n(a_n + a_{n-2} + a_{n-3})$, which is 0 by our rule. The only terms left over are $a_0, a_1 X, a_2 X^2, a_0 X^2$. Since we have all of the values already, we can calculate the product

$$A(X) \cdot (1 + X^2 + X^3) = 1 + 2X^2,$$

or

$$A(X) = \frac{1 + 2X^2}{1 + X^2 + X^3}.$$

The entire output sequence can be defined by this rational function.

This highlights the main issue of LFSRs: although they do produce good sequences statistically, the highly structured rational function output makes it easy for certain algorithms to predict future outcomes given just a small sample from the LFSR..

Theorem 1. *The Berlekamp-Massey algorithm is a procedure that finds the shortest LFSR that produces the sequence of elements $s_0, s_1, \ldots, s_{N-1}$ given the sequence. It does so by computing the pair $(L, C(x))$ where L is the smallest nonnegative integer and $C(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_L x^L \in \mathbb{F}[x]$ is a polynomial which satisfies $s_j + c_1 + s_{j-1} + c_2 s_{j-2} + \ldots + c_L s_{j-L} = 0$ for $j \geq L$.*

In order to predict the shortest LFSR given a sequence, the algorithm merely checks whether it's current guess correctly predicts the next bit in the sequence. If it does, it leaves it unchanged. If not, this leads to a *discrepancy* leading to the algorithm adjusting the polynomial by adding a shifted copy of a polynomial, stored in memory earlier, to cancel the discrepancy. This operation may

or may not grow the size of the LFSR, however, if it is forced to grow, it makes sure it is the shortest possible length after growth.

This algorithm allows potential attackers to XOR known text with cipher text to recover a piece of that key stream. For instance, if an LFSR has n stages and the attacker obtains $2n$ key stream bits, they can obtain the polynomial and the state of the register using the algorithm. This leads to the attacker only needing 32 bytes to recover LFSR of length 128 as $\frac{2 \cdot 128}{8} = 32$.

This issue in security is what leads to us finding more reliable and secure registers such as FCSR that replace the linear registers.

3 FCSRs

A Feedback Carry Shift Register (FCSR) is very similar to a LFSR, but there is one added element. Every step is done the same way, however a small memory, or 'carry' value is added to the tapped cells to get the sum. The carry starts at 0 at the first step, and the next carry is calculated by taking the floor of half the sum. So to recap, we have a register of r bits, each with a tap of 0 or a 1. It shifts to the left by 1 cell every step. The left-most term 'falls off' the register and is taken as output, while the empty cell at the right is calculated using the sum of the values in the tapped cells (tap = 1) plus the carry from the previous step. The sum is divided by 2, and the integer quotient becomes the new carry while the remainder fills in the rightmost empty cell. When the step is repeated, we get a long sequence of random bits.

If an r-bit FCSR has an infinite output sequence $a_0, a_1, \cdots$, we can express it as a $2-$adic number by embedding the sequence into a power series on powers of 2:

$$\alpha = \sum_{n=0}^{\infty} a_n 2^n \in \mathbb{Z}_2.$$

When the FCSR is on the kth step, the sum s is composed of the current carry m_k plus the tapped bits. The values currently in the register are a_{k-r} through a_{k-1}, and we multiply each value by the tap of its cell q_i, so that the cells with tap = 0 are ignored while the cells with tap = 1 are added to the sum. The sum is then split into the next output bit a_k, plus twice the next carry. We have

$$s = m_k + \sum_{i=1}^{r} q_i a_{k-i} = a_k + 2m_{k+1}.$$

This is a recursive relation which works for any step where $k \geq r$. Rearranging some terms, we have

$$-a_k + \sum_{i=1}^{r} q_i a_{k-i} = 2m_{k+1} - m_k.$$

Now, we use a similar trick to LFSR. We multiply our power series α by a 2-adic integer q, defined as follows:

$$q = -1 + q_1 2^1 + q_2 2^2 + \cdots + q_r 2^r,$$

where q_i is the tap on the ith cell of the register. Let's see what happens when we expand the product.

$$q \cdot \alpha = -\alpha + \sum_{i=1}^{r} q_i 2^i \alpha = -\sum_{n=0}^{\infty} a_n 2^n + \sum_{i=1}^{r} (q_i \sum_{n=0}^{\infty} a_n 2^{n+i}).$$

Let's analyze the coefficient of 2^k in this sum, for a general value of k. We have the coefficient of 2^k is

$$-a_k + \sum_{i=1}^{r} q_i a_{k-i}.$$

We can see that this is the exact same as the left-hand side of the identity we proved a little while ago! Thus, the coefficient of 2^k is equal to

$$2m_{k+1} - m_k$$

for $k \geq r$. Adding up over all k-values $\geq r$, we have

$$\sum_{k=r}^{\infty} (2m_{k+1} - m_k) 2^k = \sum_{k=r}^{\infty} m_{k+1} 2^{k+1} - \sum_{s=r}^{\infty} m_k 2^k.$$

This is a telescoping series. Except for the term $-m_r 2^r$, everything is added to its negative and cancels out. We are still missing the terms where $k < r$, but that is a finite integer, so after adding this to $-m_r 2^r$, we still have some integer, say p. Thus, we can express $q \cdot \alpha$ as an integer p, so

$$\alpha = \frac{q}{p},$$

a rational number in $\mathbb{Z}_2$. This also guarantees that, after a certain point, the output values a_i will repeat themselves eventually. The time complexity of FCSR is $O(n)$ to generate n random bits because it takes the same, constant amount of time to run each step.

Since the output streams of FCSRs don't match rational functions of polynomials, they are resistant to the Berlekamp-Massey attack. However, they aren't completely random. The 2-adic structure that is generated by the FCSR is vulnerable to a different type of attack. If the attacker knows that an FCSR is being used, they can guess pairs of integers instead of guessing rational functions.

In conclusion, an LFSR is not cryptographically secure because it is directly linked to a rational function, while an FCSR is also not cryptographically secure because it is linked to a rational number in $\mathbb{Z}_2$. However, the use of 2-adic numbers can modify an FCSR to become cryptographically secure.

4 References

Andrew Klapper and Mark Goresky - Feedback Shift Registers, 2-adic Span, and Combiners With Memory, Published in Journal of Cryptography Vol. 10 Issue 2, 1997.