Applications of LFSR in Applied Cryptography and Associated Security Risks

Mohit Sharma
Engineering Physics III (2018-22)
18122013

PHN-319
Technical Communication
Department of Physics
IIT Roorkee

02-02-2021

PHN-319

Final Report

February 2, 2021

Abstract

The following report is a comprehensive discussion of the applications of Linear Feedback Shift Register (LFSR) as a Pseudo Random Number Generator (PRNG) for generating entropy in the context of secure communication. This is followed by a review of its viability as a cryptographically secure PRNG, associated security risks, practical attacks and relevant mitigation mechanisms.

1 Introduction and Overview

Communications over public networks can be easily intercepted. End to end encryption is therefore essential for the preservation of privacy. In two-way communication, this can be achieved by employing a **cryptographic cipher** between the two parties. Most such ciphers require a source of entropy, and for the sake of efficiency and ease of implementation, often rely on Pseudo Random Number Generators (PRNGs) to achieve this.

2 Theoretical Background

Encryption algorithms can be divided into two broad categories: **Block ciphers** and **Stream ciphers**

Note that we will be restricting our discussion to only symmetric ciphers. We will also work with the assumption that the two parties involved in the communication already have a shared encryption/decryption key. This is typically achieved via handshake mechanisms built on key-exchange algorithms like Diffie-Hellman key exchange [4], but a deeper discussion of such handshake protocols is also omitted here.

2.1 Block ciphers

A block cipher is an encryption method that takes in a plaintext block of **N** bits (decided by the algorithm specifications) along with a symmetric key and outputs a ciphertext block of **M** bits. The significance of the key being symmetric here is that the same key is required for encryption and decryption.

Typical Block cipher bitlengths range between 128-256 bits. Examples include DES, AES, Triple DES e.t.c

2.2 Stream ciphers

Since Block ciphers can only operate on fixed length blocks of data, they are unsuitable for encrypting a continuous stream of data as in the case of Audio aur Video streaming (phone calls for example). Stream ciphers are capable of encrypting/decrypting data 1 bit at a time.

An ideal Stream cipher can be visualized as a one time pad

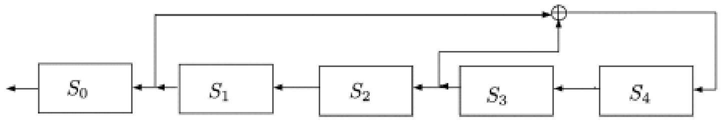
$$G : \{0,1\}^k \to \{0,1\}^k k \in N \tag{1}$$

It takes in an arbitrary length of data and outputs the same length ciphertext, which should appear as random noise to anyone listening without being decrypted with the corresponding key. They way this is achieved is by generating an encryption key of same length as the plaintext and then its taking bitwise XOR with the data. This key is a sequence of random bits and is generated through a PRNG and the security of the encryption depends entirely upon the "strength" (as we will define later) of this key. This is where LFSR comes in

2.3 Linear Feedback Shift Register

A linear feedback shift register (LFSR) is a shift register whose input bit is the output of a linear function of two or more of its previous states.

LFSR can be generated by a series of flip flops sharing a **common clock**, series meaning the output of one flip flop is used as input for the next. Thus a register of length **m** is capable of storing**m** bits of information at a time.



1. The shift register is initialized with a vector with entries $s_0, ..., s_{m-1}$, called the seed
2. s_0 forms part of the output
3. The content of stage i is shifted to stage i-1 for $1 \leq i \leq m - 1$
4. the new content (the feedback bit) of the stage m-1 would be obtained by xor-ing a subset of the content of the m stages
5. The flip-flops whose storage is used in feedback, are called **taps**

3 LFSR as a PRNG

The initial input of an LFSR is called a seed. Since any register has a finite number of possible states, it must eventually be periodic. However, an LFSR with a well-chosen feedback function and seed can produce a sequence of bits which appears random (has good statistical properties) and which has a large period.

The period of an LFSR with **m** stages is given by a divisor of $2^m - 1$

Two LFSR constructions will produce an identical sequence of bits as long as they have the same length, taps and are initialized with the same seed. This means a small seed can be used to generate key sequences of large lengths to be used in OTP: 1. The seed can therefore be treated as the encryption key while being relatively small in size, reducing the amount of data that needs to be exchanged securely before commencing communication. 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. We define the **characteristic polynomial** of an LFSR as the polynomial given by

$$s_m = \sum_{i=0}^{m-1} c_i x^i, c_i \in \{0, 1\} \forall 0 \leq i \leq m \tag{2}$$

4 Implementation

Though LFSRs are typically constructed as digital circuits, programmatic implementations are also widely used and can be easier to experiment with

```
def lfsr(seed, taps):
    sr, xor = seed, 0
    while 1:
        for t in taps:
            xor += int(sr[t-1])
        if xor%2 == 0.0:
            xor = 0
        else:
            xor = 1
        print(xor)
        sr, xor = str(xor) + sr[:-1], 0
        print(sr)
        if sr == seed:
            break
```

5 Security

The key difference between a cryptographically secure PRNG (CSPRNG) and an insecure one is that a random number generator used for cryptographic purposes has to stand up to an attacker. A CSPRNG must satisfy all the statistical randomness tests a statistical PRNG does, but it also needs to be unpredictable. A CSPRNG is designed to resist attempts by a human attacker to predict its next output; it should be hard to tell it from a truly random sequence even if the attacker knows the algorithm used to make it. For instance, **if an attacker sees the result of many invocations, it needs to still be hard to predict the result of any future invocation** [2]. Thus is we can retrieve the initial seed used in an LFSR, or find a way to leak the characteristic polynomial, we will have essentially broken the security since all future outputs can be predicted from this information.

5.1 Analytical attacks

since LFSR is essentially a collection of linear transforms, there exist a number of attacks against the mathematical model itself. The most prominent such attack is based on the **Berlekamp-Massey** (BM) algorithm [1]

Linear complexity, L, of a binary sequence is the shortest LFSR that generates it. Berlekamp-Massey algorithm can be used to obtain this minimum length LFSR, allowing the attacker to replicate its output and thereby compromising the PRNG.

5.1.1 Matrix form of LFSR

Consider an LFSR characterized by a monic feedback polynomial

$$s_n = \sum_{i=0}^{n-1} a_i x^i, a_i \in \{0,1\} and a_{n-1} = 1$$

Then the Frobenius companion matrix is defined as

$$C_{P_f} := \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_0 & a_1 & a_2 & \dots & a_{n-1} \end{pmatrix}$$

Let the seed of the LFSR be $U_0 = (u_0, u_1, u_2, ...u_{m-1})^t$ be the seed. Then the state of the LFSR after k cycles is given by

$$U_k = C_P^k * U_0$$

Thus, the Frobenius matrix completely determines the behaviour of the LFSR.

5.1.2 Berlekamp Massey Algorithm

It is clear from the above equation that if the length of the LSFR is known, then a simple matrix inversion is enough to reconstruct the feedback polynomial. The difficulty then is to find the length l of the LFSR from a given output stream.

Basically speaking, the BM algorithm starts with the assumption that the length of the LSFR is l=1, and then iteratively tries to generate the known sequence and if it succeeds, everything is well, if not, l must be increased

To solve a set of linear equations of the form

$$\sum_{k=0}^{i} \lambda_k s_k = 0$$

a potential instance of λ can be constructed step by step. Let C denote such a potential candidate or the "error locator" polynomial for L errors

$$C = \sum_{i=0}^{L} c_i X^i$$

The goal of the BM algorithm is to now determine the minimum value of degree L and to construct a suitable C such that the equations are satisfied

With each iteration, we calculate the discrepancy d between the candidate and the actual feedback polynomial:

$$d = S_k + c_1 S_{k-1} .... + c_L S_{k-L}$$

If the discrepancy is zero, C is a valid candidate and the algorithm can be terminated. Else, C must be adjusted according to the relation

$$C = C - \frac{d}{b} X^m B \tag{3}$$

, where B is a copy of the last candidate C since L was updated, b a copy of the last discrepancy since L was updated, and the multiplication by X^n is effectively just an index shift.

In summary, the algorithm can be summarized as

1. Compute the discrepancy, d
2. **If** d = 0, terminate the algorithm. C is the Companion matrix that characterizes the required LFSR
3. **Else:**
   - Update C according to 3
   - Calculate new d
   - Return to step 2

5.2 Side channel (circuital) attacks

Even when a cryptographic primitive is provably secure, there can exist attacks against practical implementations of it. These are referred to as **Side channel attacks**. There are a number of side-channel pathways through which information can be leaked inadvertently. The most prominent of them being through the measurement of the time taken or power consumed to perform a cryptographic function which involves the use of the cryptographic key, or some other data whose loss of integrity could compromise the given system.

5.2.1 Power analysis attacks

Traditional LFSR implementations are vulnerable to side-channel leaks through power consumption analysis [3]. It is possible to precisely determine the state of an n-bit LFSR by measuring the power consumed by the LFSR in each cycle over consecutive cycles linear in n

Theorem 1. Let HD_t be the Hamming Distance between the n-bit vectors ,ST_t and ST_{t+1}. Hamming distance is defined as the number of states that have different values in the two vectors. Hence, for an LFSR cycle, the hamming distance is equal to the number of flips
Let $PD_t = (HD_t \;\; HD_{t+1})$. Then, PD$_t \in \{1, 0, 1\}$. [3]

Corollary 1. Let PDt_t be defined as follows: It is equal to 0 when, $HD_t = HD_{t+1}$, else it is 1. Given $s_(n+1), s_n, s_1$ and s_0 as defined in Theorem 1, $PDt_t = s_{n+1} \bigoplus s_n \bigoplus s_1 \bigoplus s_0$

The dynamic power consumed by a digital circuit is directly proportional to the switching activity (number of components in the circuit that have a state-transition from 0 to 1 or vice-versa). Therefore for LFSRs, the power consumed during the transition in cycle t, that is, from time period t to time period t+ 1, is proportional to HD_t. While we cannot accurately infer the number of toggles from the amount of power consumed, we can compare the power consumption of two cycles and determine whether the number of toggles were same or different.

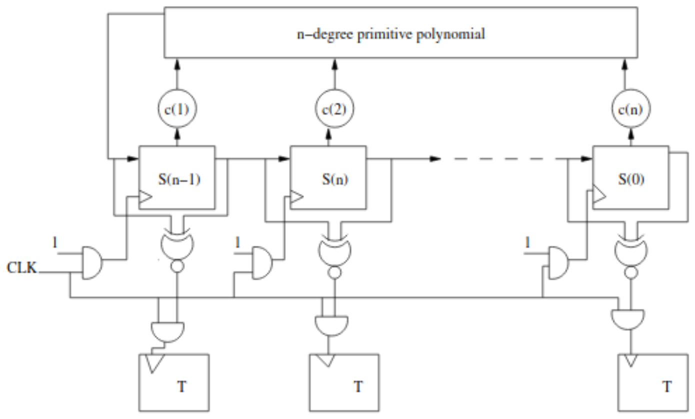
Thus power analysis can be used to leak the characteristic polynomial. The attack flow is as follows
Let POW(k) denote the dynamic power consumed by the LFSR at time instant k

1. Measure POW(0), for time instant 0
2. For each time instant k, k $\geq 1 do$
    3. Measure the dynamic power POW(k)
    4. $PDt_{k1} = 1$ if POW(k1) =POW (k), else it is 0.
    5. Input PDt_{k1} into the Berlekamp-Massey (BM) Algorithm. If BM terminates then exit this for loop; else repeat Step 2

Result: BM algorithm computes the feedback polynomial of the LFSR. The initial state can now be easily extracted

5.2.2 Countermeasures

Power analysis attacks can be countered as shown in the following circuit. For every flip flop F, there is a corresponding "shielding" flip-flop Ft. If F flips, Ft doesn't and if F doesn't flip, Ft does. Thus, the total number of toggles in the circuit remain constant for each cycle and power consumption analysis is rendered futile



References

[1] *Berlekamp Massey Algorithm.* URL: http://www-users.math.umn.edu/~garrett/students/reu/MB_algorithm.pdf.

[2] *Cryptographically secure PRNGs.* URL: https://crypto.stackexchange.com/questions/12436/what-is-the-difference-between-csprng-and-prng.

[3] *Differential power analysis attacks on LFSRs.* URL: https://sci-hub.se/10.1007/978-3-540-77026-8_30.

[4] *Diffie Hellman key exchange algorithm.* URL: https://www.math.ucla.edu/~baker/40/handouts/rev_DH/node1.html.