Extended period LFSR using variable TAP function

Ariel Molina-Rueda, Fernando Uceda-Ponga, Dr. Claudia Feregrino Uribe
Instituto Nacional de Astrofísica Óptica y Electrónica,
Luis Enrique Erro 1, Tonanzintla, Puebla. AP 72000. México
{arielm, fuceda, cferegrino}@ccc.inaoep.mx

***Abstract*—This paper presents a method to extend the period of a Linear Feedback Shift Register (LFSR) by proposing an algorithm to generate primitive polynomials, this is archived by using basic LFSR with a maximum period equal to a prime number. The period extension achieved with our proposed method is statistically robust and has a very long extension of the LFSR period, as long of $(2^{120})!(2^N-1)$ for a 127 bit length register. Also by separating the phases of setup and running in the algorithm avoid losing the characteristically speed of the LFSRs.**

***Index Terms*—**

I. Introduction

In recent years, the increasing use of the Internet and the growing exchange of digital information have lead to the necessity of reinforcing security. Currently, one of the most widely used techniques for security is cryptography. It provides the tools to build the most modern security pro tocols used for the transmission of information. Cryptography comes from the Greek words $\kappa\rho\upsilon\pi\tau\omega o\sigma$ and $\gamma\rho\alpha\epsilon\iota\nu$ which means “hidden” and “writing” respectively, its definition is: “The art of writing with secret key or in an enigmatic manner”. A cryptosystem is a quintuple (M, C, K, E, D), where: M represents the set of all messages without ciphering (that is clear text or plaintext), C represents the set of all possible ciphered messages, K represents the set of keys that the cryptosystem may use, E is the set of ciphering transformations or functions family that can be applied to M to obtain an element of C. There is a different transformation E_k for each possible value of k. D is the set of deciphering transformations. All cryptosystems must obey the following condition: $D_k(E_k(m)) = m$. There are two types of cryptosystems:

1. Symmetrical or private key, they employ the same key k to cipher and to decipher. Its main inconvenience is that the key must be known by the sender and by the receiver, so, the transmission of the key must be secure, and
2. Asymmetrical or public key, they use a double key (k_p, k_P), k_p is known as private key and k_P is known as public key. One of them is used for the ciphering E transformation and the other one for the deciphering D transformation. In many cases, they are interchangeable. k_P must not allow to compute k_p. They can be used for secure communications, since just one key travels by the insecure channel.

In practice, a combination of both schemes is used. A message is codified using a symmetric algorithm and a session key that must be different each time. The session key is ciphered using asymmetric cryptography. The only way these keys are secure is that there must not be dependence between a key and the next one, that means they are random. So, there is an interest in random numbers in cryptography. It is practically impossible to generate really random number sequences; pseudorandom number generators are used instead. All pseudorandom generators produce finite and periodic sequences of numbers employing arithmetic operations and/or logic ones. These sequences are as long as possible before they can be repeated and that they can pass the statistical test of randomness. The cryptosystems that exploit the idea of the pseudorandom generators are the stream ciphers. The cryptographically random generators follow this property: from a piece of an arbitrary long sequence, it is computationally impossible the problem of predicting the following bit of the sequence. The requirement is that the complete sequence can not be computed from a piece of it, and at the same time, it can be completely regenerated from the seed. Pseudorandom generators allow ciphering messages of arbitrary length combining the message with the sequence using the exclusive-OR operation byte to byte. Linear Feedback Shift Registers, LFSR, are the basis for many of the sequence generators for stream ciphers. This paper aims to extend the period of a LFSR in order to make them more secure.

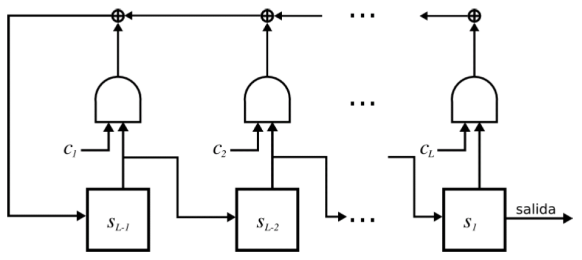

Fig. 1. LFSR of length L

The Linear Feedback Shift Register (LFSR) was first presented by Golomb [3] as pseudorandom number generator. In the present time they are used for many hardware key generators for the following reasons:

1. LFSR are easily implemented in hardware
2. LFSR archive high throughtput
3. LFSR produce sequences with long periods
4. LFSR produce have good statistical properties
5. LFSR can be easily analized with abstract algebra

II. Dissecting an LFSR

An LFSR is defined here by using Fig 1. An LFSR which has a length L will have L elements or delay elements, in Fig. 1 this are labeled $s_0, s_1, ..., s_{L-1}$; each one with the capacity to hold one bit of information. A clock modulates this LFSR, with each time unit of this clock the following occurs:

1. The content of element s_0 is an output bit of the LFSR's sequence.

2. The contents of element s_i moves to s_{i-1}, for all i in $1, \cdots, L-1$
3. The contents of element s_{L-1} is calculated by adding modulus 2, the contents of the rest of the elements. Note that the hardware implementation of this addition is the XOR function.

An LFSR as in Fig 1 is noted as $\langle L, C(x) \rangle$ being L the length and $C(x)$ the connection polynomial:

$$C(x) = 1 + c_1 x + c_2 x^2 + \cdots + c_L x^L \in \mathbf{Z}_2[x]$$

If all the the elements of the LFSR are set to zeroes, the output sequence will be all zeroes, for this reason, this set up sequence is excluded. For all other $2^L - 1$ set up sequences, the LFSR will eventually loop though some, or all of its states, and this is related with the kind of connection polynomial [4]. If an irreducible polynomial is used, the period of the LFSR will be n, such that

$$n | 2L - 1$$

so n will be a divisor of the maximum period. Moreover, if a primitive polynomial is used, a maximal period is archived for non-zero initialization. Generation of primitive polynomials is a hard task involving factorization of primes [4]. This paper is focused on the generation of irreducible polynomials instead of primitive ones, but using some mathematical properties, so that calculation of primitives is not necessary but instead all irreducibles generated are in fact, primitives. Thus reducing drastically the computing power needed.

A. Statistical properties of sequences

Let S be an sequence that is generated by a maximum-length LFSR of length L, this is a sequence generated when a primitive polynomial is used as a connection polynomial:

- Let k be an integer, $1 \leq k \leq L$, and let s_0 be any subsequence of S of length $2L + k - 2$. Then each non-zero sequence of length k appears exactly 2^{L-k} times as a subsequence of s_0. Furthermore, the zero sequence of length k appears exactly $2^{L-k} - 1$ times as a subsequence of s_0. In other words, the distribution of patterns having fixed length of at most L is almost uniform.
- S satisfies Golomb's randomness postulates.

1. In the cycle s^N of S, the number of 1's differs from the number of 0's by at most 1.
2. In the cycle s^N, at least half the runs have length 1, at least one-fourth have length 2, at least one eighth have length 3, etc., as long as the number of runs so indicated exceeds 1. Moreover, for each of these lengths, there are (almost) equally many gaps and blocks.
3. 3. The autocorrelation function $C(t)$ is two-valued. That is for some integer K,

$$N \cdot C(t) = \sum_{i=0}^{N-1} (2s_i) \cdot (2s_{i+t} - 1)$$

$$= \begin{cases} N \text{ if } t = 0 \\ K \text{ if } 1 \leq t \leq N-1 \end{cases}$$

That is, every sequence is also a pn-sequence (pseudonoise sequence).

Focusing on the irreducible polynomials. As it was said the period will be a divisor of $2^L - 1$. In the case $2^L - 1$ is a prime number, then n can only be 1 or $2^L - 1$, it cannot be 1, so it is $2^L - 1$. So a way to get maximum period of an LFSR without generating primitives was just shown, generate irreducibles such that $2^L - 1$ is prime, and this is a considerably easier task, it is possible to generate them pseudorandomly and check irreducibility within $\mathbf{Z}_2[x]$ with this algorithm:

```
is_irreducible(f(x))
 u(z)<-x
 for i=1 to floor(m/2):
  u(x) <- u(x)^2 (mod f(x))
  d(x) <-mcd(f(x), u(x)-x)
   if d(x) != 1 then return FALSE
 return TRUE
```

Which uses the Euclidean algorithm for calculating the minimum common divisor:

IN two polynomials $g(x); h(x) \in \mathbf{Z}_p[x]$.
OUT the greatest common divisor of $g(x)$ and $h(x)$.

1. While $h(x) \neq 0$ do the following:
2. $\rightarrow$ Set $r(x) = g(x) \bmod h(x)$, $g(x) = h(x)$, $h(x) = r(x)$.
3. Return(g(x)).

Also note that the lenght of the LFSR will be L, such that $2^L - 1$ is prime, this primes are known as Mersenne primes, and there is an ongoing crusade to find them [1]. The lengths are known, and the first Mersenne primes are $3, 7, 31, ...$ corresponding to $L = 2, 3, 5, ...$. There are only 44 known Mersenne primes until the writing of this paper[1]. In this paper the use of prime $2^127 - 1$ is discussed for an LFSR of length 127, but all that is said is valid for all the other primes including Mersenne primes not yet discovered and even non Mersenne primes, but Mersenne primes are used for a simple illustration. There can even exist a LFSR of length 32582657 corresponding to the largest known Mersenne prime until today $2^{32582657} - 1$ if that could be possible.

Next is the part of obtaining those irreducible to use into the LFSR as connection polynomials. This approach can be used to generate a monic polynomial of degree m:

```
generate_irreducible( m )
 Do{
  f(x) = generate random coefficients for
         a monic polynomial of deg. m.
         (seed for this PRNG is part of
         the user-provided key)
  salir = is_irreducible( f(x) )
 }while not salir
 return f(x)
```

Note that a monic is needed, otherwise it will not be of degree m, also the constant coeficient must always be nonzero or the polynomial will not be irreducible, it will be trivially divisible by x.

$$x^{127} + x^{126} + x^{125} + \cdots + x + 1$$

In our case the degree is 127, so only 125 random bits are to be generated, the bits for the initialization are received from the user as a part of the key. When using different keys, different polynomials are generated.

So, a number of polynomials is generated in an initialization stage, in this partany pseudorandom number generator can be used, it was chosen to use Blum Blum Shub[2], because is very strong and the generated number are congruent with:

$$x_i = (x_0^{2^i mod(p-1)(q-1)})modM$$

where M can be prime and also it is possible to calculate any x_i without storing, and can be calculated directly, this is a good characteristic for further hardware implementations, still a LFSR of 127 bit and a primitive tap function can be used instead of the Blum Blum Shub generator for low hardware cost implementations. A change the generator of this point have a low effect on the security, our only compromise is the use of a generator that accept user Key-Set or seed-password. This compromise will difficult the task of cryptanalyst and provide security. This is because the security relies in the output of the LFSR that is being built and not on the internal PRNG initializator. In this way, the security is also increased because the tap polynomials will be scrambled in a pseudorandom way, so even if the current tap is broken the attacker will not know which will be the next tap (that's the reason it is not recommened the use of a simple counter as a seeder for the irreducible analyzer).

After this, a 127 bit LFSR is used, with an initialization vector based on the key provided by the user. When the $2^127 - 1$ bits of the output sequence are depleted the connection polynomial is reset with another one of those generated in the initialization stage, and a different sequence is obtained, they have good statistical properties as Golomb has shown [1].

III. Extending the period to the maximum

The extension of the period is calculated by:

$$I(2^{127} - 1)$$

Where I is the total number of existing irreducible polynomials which according to Sloane's series A011260 is:

$$1339694357956450643556592942644756738$$

Which is approximately 2^{120}.

If there are 2^{120} primitives and a total of 2^{127} then the probability to get a primitive when polynomials are generated randomly is:

$$\frac{2^{120}}{2^{127}} = \frac{1}{2^7} = \frac{1}{128}$$

So on average it is expected that 128 polynomials are generated to obtain just one that is primitive and useable for our purpose.

In order to further extend the period of the LFSR the polynomials can be used in a permutated form, so each time the polynomials are depleted, there will be chosen again but in a different order, this will increase the period and is still statistical secure.

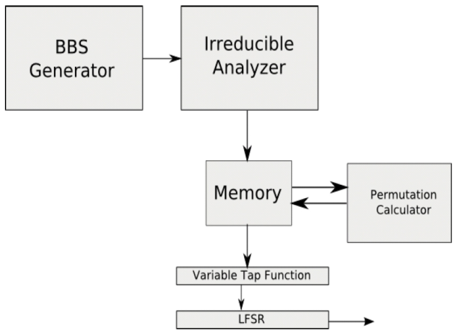

Fig. 2. Simplified Architecture Diagram

As result of the permutated use the addition of the max possible period is:

$$(2^{120})! \cdot (2^{127} - 1)$$

This period is **extremely large**, so it is nowadays practically impossible to have data input that long, even if the contents of the whole internet are used as input.

IV. Implementation constrains and tests

This LFSR was created with the objective of using it as a replacement of the ordinary LFSR in secure generators, increasing the global security of a stream cipher. Because of this porpouse, a practical implementation approach will be shown that make de extended LFSR a black box with a behavior similar to any other LFSR but with a larger period.

First of all the main architecture of our LFSR will be as the one shown in Fig. 2 .

Due to memory constrains the maximum number of irreducible polynomial will be determinated for the amount of memory available, i.e for a 4MB of memory usage a number of approximately $2,7488 \times 108$ polynomials are possible, this is aproximately 2^{28}.

This makes the LFSR period equals to $(2^{28})!(2^{127} - 1)$, this is still very large and the memory usage is low according with the actual memory availability. It shall be noted that this capacity grows as memory avaiability becomes higer in the future. Specifically for small devices

This black box LFSR was implemented in software as a test of the viability. This implementation was made in Java 1.5 using a command line setup to avoid CPU time consuming in graphics environments, all modules was created as an independent class, the LFSR and its tap function were made with integers.

The setup phase using a Blum Blum Shub Generator as the random polynomial feeder was 14 seconds.

The average speed of this LFSR after the setup phase was one bit every 0.01 seconds. That is 100bit/sec. The computer used was an AMD Turion, at 1.6Ghz, running Windows. It is important to mention that the process was restrained to use only 4 Megabytes of RAM. This is described here as a *proof-of-concept* and if implemented directly on hardware it could gain up to a $100x$ increase in troughtput.

V. Conclusions

It has been shown how to increase the period of an LFSR in an easy an viable way, if a low cost implementation is desired one LFSR could substitute the Blum Blum Shub Generator. Even with the implementation constrains, this LFSR can make stream ciphers stronger due to period extension. Our *proof-of-concept* java implementation on a PC, gave as result the generation of 100bit/sec for this extended LFSR. This throughtput can be increased if implemented on a dedicated hardware architecture, leading up to a $100x$ throughtput raise. The implementation on hardware is left as future work.

References

1. http://www.mersenne.org/.

2. L. B. M. Blum and M. Shub. A simple unpredictable pseudorandom number generator. SIAM Journal on Computing, 15:364-383, 1986.

3. S. Golomb. Shift Register Sequences. Aegean Park Press (1982) reprint, Laguna Hills, California, 1967.
4. A. Menezes. Handbook of applied Cryptography. CRC Press, 1997.