

Appendix A. Pseudo-random Sequence (Number) Generators

In this appendix, we introduce how to design pseudo-random sequence (number) generators (PRSG) which are primarily used in stream ciphers for generating key streams, and key deviation functions in key agreement protocols. We present two types of pseudo-random sequence (number) generators. One is of LFSR based PRSGs whose randomness does not depend on any computational hard problems. The randomness of the second type of PRSG is based on some computational infeasible problems. This appendix is organized as follows. Section 1 is a general introduction to linear feedback shift register (LFSR) sequences. The concepts of one-time pad and randomness criteria are introduced in Section 2. In Sections 3, three types of frequently used LFSR based PRSGs are introduced, i.e., filtering sequence generator, combinatorial sequence generators, and clock-control and shrink generators, respectively. In Section 4, the Blum-Blum-Shub generator is presented, whose randomness is based on computational infeasible problems.

1 Linear Feedback Shift Register Sequences

Feedback shift register sequences have been widely used as synchronization codes, masking or scrambling codes, and for white noise signals in communication systems, signal sets in CDMA (code division multiple access) communications, key stream generators in stream cipher cryptosystems, random number generators in many cryptographic primitive algorithms, and for testing vectors in hardware design. S. Golomb's popular book "Shift Register Sequences", first published in 1967 and revised in 1982 is a pioneering book which discusses this type of sequences. This section introduces basic concepts of LFSR sequences.

1.1 Feedback Shift Registers

In this subsection, we give a definition and some of the basic terms for feedback shift register sequences. We denote $F = \{0, 1\}$ where the addition and multiplication in this set are defined as the usual addition and multiplication reduced by modulo 2, and it is called a finite field with two elements, or a binary field. Let

$$F^n = \{(a_0, a_1, \dots, a_{n-1}) \mid a_i \in F\}$$

which is a vector space over F of dimension n . A function with n binary inputs and one binary output is called a *boolean function of n variables*, i.e., $f : F^n \rightarrow F$, which can be represented as

⁰Copyright ©2008 L. Chen and G. Gong. All rights reserved. May be freely reproduced for educational or personal use.

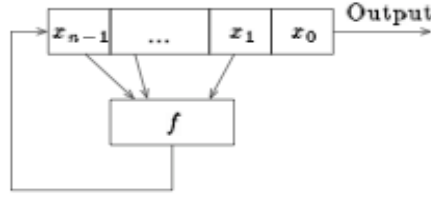


Figure 1: A Block Diagram for an FSR



follows:

$$f(x_0, x_1, \dots, x_{n-1}) = \sum c_{i_1 i_2 \dots i_t} x_{i_1} x_{i_2} \dots x_{i_t}, c_{i_1 i_2 \dots i_t} \in F \quad (1)$$

where the sum runs through all subsets $\{i_1, \dots, i_t\}$ of $\{0, 1, \dots, n-1\}$. This shows that there are 2^{2^n} different boolean functions of n variables.

A. Basic Concepts and Examples

An n -stage shift register is a circuit consisting of n consecutive 2-state storage units (flip-flops) regulated by a single clock. At each clock pulse, the state (1 or 0) of each memory stage is shifted to the next stage in line. A shift register is converted into a code generator by including a feedback loop, which computes a new term for the left-most stage, based on the n previous terms. In Figure 1, we see a diagram of a feedback shift register (FSR).

Each of the squares is a 2-state storage unit. The n binary storage elements are called the *stages* of the shift register, and their contents (regarded as either a binary number or a binary vector, n bits in length) is called a *state* of the shift register. $(a_0, a_1, \dots, a_{n-1}) \in F^n$ is called an *initial state of the shift register*. The feedback function $f(x_0, x_1, \dots, x_{n-1})$ is a boolean function of n variables, defined in (1). At every clock pulse, there is a transition from one state to the next. To obtain a new value for stage n , we compute $f(x_0, x_1, \dots, x_{n-1})$ of all the present terms in the shift register and use this in stage n . For example, the next state of the shift register in Figure 1 becomes where

$$a_n = f(a_0, a_1, \dots, a_{n-1}).$$

After the consecutive clock pulses, a feedback shift register outputs a sequence:

$$a_0, a_1, \dots, a_n, \dots \quad (2)$$

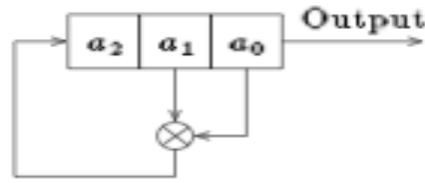


Figure 2: A 3-stage nonlinear feedback shift register

The sequence satisfies the following recursive relation

$$a_{k+n} = f(a_k, a_{k+1}, \dots, a_{k+n-1}), k = 0, 1, \dots \quad (3)$$

Any n consecutive terms of the sequence in (2)

$$a_k, a_{k+1}, \dots, a_{k+n-1}$$

represents a state of the shift register in Figure 1. A *state (or vector) diagram* is a diagram that is drawn based on the successors of each of the states. The output sequence is called a *feedback shift register sequence*. If the feedback function $f(x_0, x_1, \dots, x_{n-1})$ is a linear function, then the output sequence is called a *linear feedback shift register (LFSR) sequence*. Otherwise, it is called a *nonlinear feedback shift register (NLFSR) sequence*. Sometimes, we also say that \mathbf{a} is *generated* by an LFSR (or NLFSR). Here linear means that the feedback function computes the modulo 2 sum of a subset of the stages of the shift registers.

Example 1 In Figure 2, we see a 3-stage shift register with a (nonlinear) feedback function $f(x_0, x_1, x_2) = x_0x_1$.

From this, we can compute the next-state function, as shown in the following table, a “successor table”, for each of the eight states of the shift register.

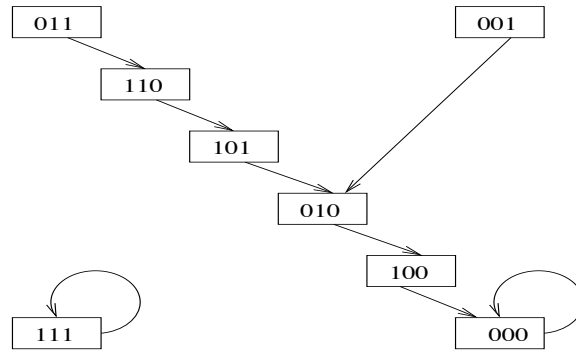


Figure 3: The state diagram of the FSR in Fig.2

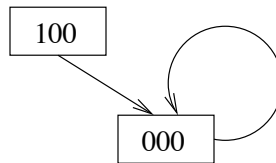


Figure 4: The state diagram with the initial state $x_0x_1x_2 = 100$

Succession of states

| $x_0x_1x_2$ | $x_0x_1x_2$ |
|---------------|-------------|
| current state | next state |
| 000 | 000 |
| 001 | 010 |
| 010 | 100 |
| 011 | 110 |
| 100 | 000 |
| 101 | 010 |
| 110 | 101 |
| 111 | 111 |

The state diagram of the eight states is shown in Figure 3. From the state diagram, we can directly observe the autonomous behavior of the device. For example, the initial state 100 leads to the output sequence 1000..., and this state diagram is shown in Figure 4.

Example 2 A 3-stage LFSR is shown in Figure 5 with the linear feedback function $f(x_0, x_1, x_2) = x_0 + x_1$.

The truth table of this feedback function is given in Table 1, and the state diagram of the corresponding LFSR is shown in Figure 6.

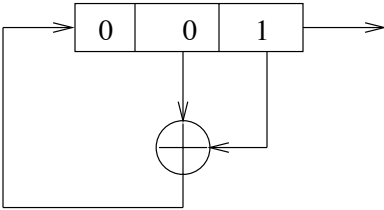


Figure 5: A 3-stage LFSR for example 2

Table 1: Truth table of $f = x_0 + x_1$

Truth table of $f = x_0 + x_1$

| $x_0x_1x_2$ | $f = x_0 + x_1$ |
|-------------|-----------------|
| 000 | 0 |
| 001 | 0 |
| 010 | 1 |
| 011 | 1 |
| 100 | 1 |
| 101 | 1 |
| 110 | 0 |
| 111 | 0 |

The output sequence with the initial state 100 is

$$10010111001011 \dots,$$

which is seen to repeat periodically with a period of 7.

Example 3 A 3-stage LFSR with a nonlinear feedback function $f(x_0, x_1, x_2) = x_0 + x_1x_2 + x_2 + 1$, as shown in Figure 7.

The truth table of this boolean function is given in Table 2. From the truth table, the state diagram of the NLFSR is easily obtained, as shown in Figure 8. Therefore, the output sequence with the initial state 100 is

$$1000101110001011 \dots.$$

Example 4 A 4-stage NLFSR with a nonlinear feedback function $f(x_0, x_1, x_2, x_3) = x_0 + x_1x_2x_3 + x_1 + 1$. The output sequence with the initial state 1111 is given by

$$1111011001010000 \dots$$

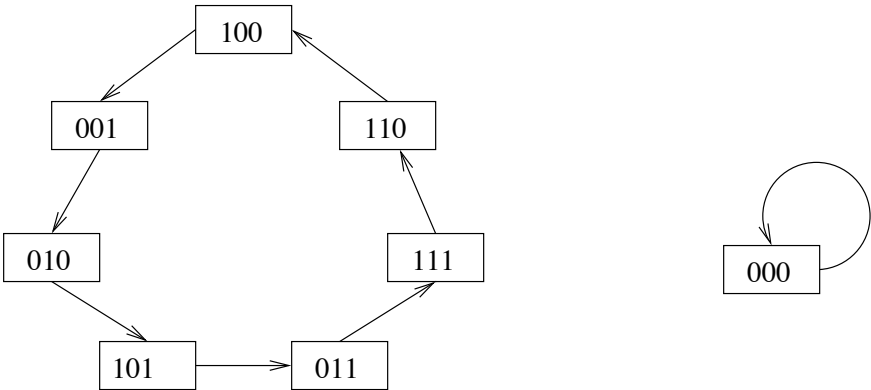


Figure 6: State diagram of the LFSR in Figure 5

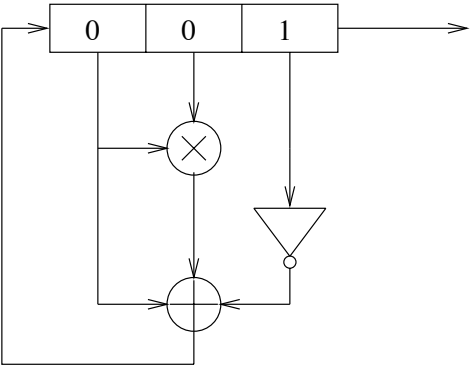


Figure 7: The 3-stage NLFSR for example 3

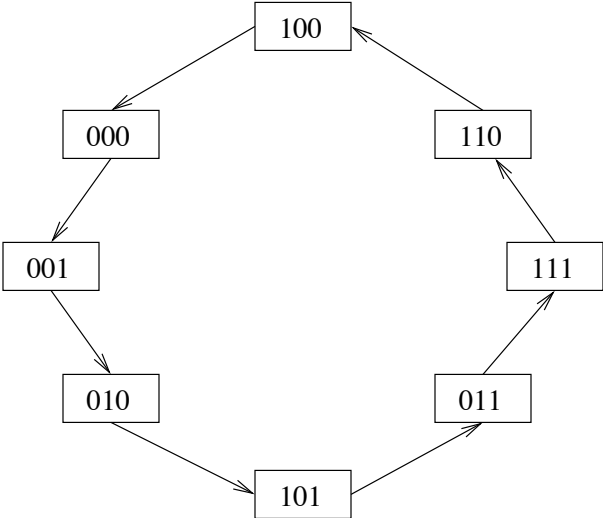


Figure 8: State diagram of Figure 7

Table 2: Truth table of $f = x_0 + x_1x_2 + x_2 + 1$

| $x_0x_1x_2$ | $x_0 + x_1x_2 + x_2 + 1$ |
|-------------|--------------------------|
| 000 | 1 |
| 001 | 0 |
| 010 | 1 |
| 011 | 1 |
| 100 | 0 |
| 101 | 1 |
| 110 | 0 |
| 111 | 0 |

which has a period of 8.

A *de Bruijn sequence* is an output sequence of an n -stage NLFSR having period 2^n and satisfying that each n -tuple occurs exactly once in each period. The sequences given by Examples 3 and 4 are de Bruijn sequences with periods 8 and 16 respectively. For a general discussion of nonlinear feedback shift register sequences including the number of de Bruijn sequences and constructions for several subsets of de Bruijn sequences, see Chapter 6 in *Shift Register Sequences* [12].

We will see that the periods of LFSRs are completely determined by their feedback functions in a mathematically predictable way. However, for NLFSRs, there are only a few results on the period problem in the literature.

Remark 1 The de Bruijn sequence in Example 3 can be obtained by inserting an extra 0 into the run of 2 consecutive zeros of the LFSR sequence in Example 2. But the feedback functions of these two sequences are completely different. A sequence generated by an n -stage LFSR with period $2^n - 1$ is called a *maximal length sequence*, or m -sequence for short (we will formally define it later). From any m -sequence of period $2^n - 1$, we can get a de Bruijn sequence by inserting an extra 0 into the run of $n - 1$ consecutive zeros of the m -sequence. However, this type of de Bruijn sequence is not secure for use in stream cipher cryptosystems.

B. Periodic Property

For a binary sequence generated by an LFSR, say $\mathbf{a} = a_0, a_1, \dots$, then it is either *periodic* or *ultimately periodic* in the sense that there exist integers $r > 0$ and $u \geq 0$ such that

$$a_{i+r} = a_i \text{ for all } i \geq u. \quad (4)$$

The sequence is said to be *ultimately periodic* if $u > 0$. Otherwise the sequence is said to be *periodic*.

For example, the output sequence 00011011011 \cdots of a 4-stage LFSR with the feedback function $f(x_0, x_1, x_2, x_3) = x_2 + x_3$ and the initial state $a_0a_1a_2a_3 = 0001$ is an ultimately periodic sequence, where $u = 2$ and the period r is 3. The output sequence of the feedback shift register sequence in Figure 5 is a periodic sequence with period 7.

C. Linear Feedback Shift Register Sequences

If the feedback function $f(x_0, x_1, \dots, x_{n-1})$ is a linear function, i.e., if it can be expressed as

$$f(x_0, x_1, \dots, x_{n-1}) = c_0x_0 + c_1x_1 + \dots + c_{n-1}x_{n-1}, c_i \in F, \quad (5)$$

then the recursive relation shown in (3) becomes the following linear recursive relation

$$a_{k+n} = \sum_{i=0}^{n-1} c_i a_{k+i}, k = 0, 1, \dots. \quad (6)$$

Thus an LFSR sequence is also called a *linear recursive sequence* (or *linear recurring sequence*) over F in the literature.

The maximal period of an LFSR sequence with n stage is $2^n - 1$, which is called a *maximal length sequence*, shortened as *m-sequence*, or pseudo noise sequence in communications. Usually, we associate a linear feedback boolean function given in (5) with a polynomial

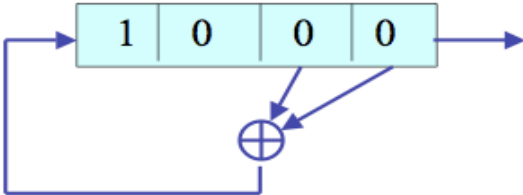
$$f(x) = x^n + c_{n-1}x^{n-1} + \dots + c_1x + c_0$$

which is called a *characteristic polynomial* of the LFSR. If \mathbf{a} is an *m-sequence*, then $f(x)$ is called a *primitive polynomial*. In the literature, primitive polynomials are computed by many researchers and listed them in tables. We attach a table with primitive polynomials of degrees from 2 to 30 by the end of this chapter.

Example 5 Examples of the *m-sequences* of period 7, 15, 31 and 63 together with their respective corresponding primitive polynomials and LFSR implementations are given as follows.

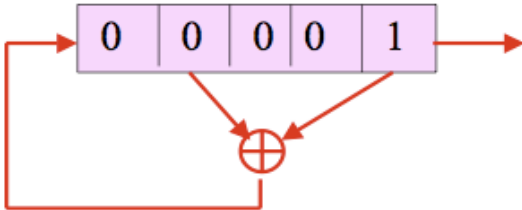
1. $n = 3$, the LFSR given in Example 2 is an *m-sequence* with characteristic polynomial $f(x) = x^3 + x + 1$.
2. $n = 4$, $f(x) = x^4 + x + 1$, and

$$\mathbf{a} = 000100110101111.$$



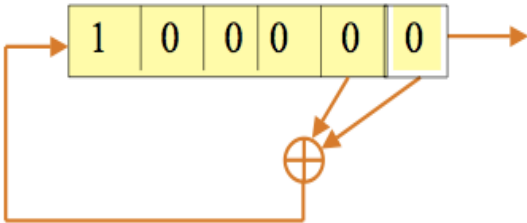
3. $n = 5$, $f(x) = x^5 + x^3 + 1$, and

$$\mathbf{a} = 1000010101110110001111100110100.$$



4. $n = 6$, $f(x) = x^6 + x + 1$, and

$$\mathbf{a} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}$$



2 Randomness Measurements

Randomness of a sequence refers to the unpredictability of the sequence. Any deterministically generated sequence used in practical applications is not truly random. The best that can be done

here is to single out certain properties as being associated with randomness, and to accept any sequence which has these properties as *a random or more properly, a pseudorandom sequence*.

2.1 Golomb's Randomness Postulates and Randomness Criteria

Let \mathbf{a} be a sequence of period N over \mathbb{F} . As usual, we also write $\mathbf{a} = (a_0, a_1, \dots, a_{N-1})$, an element in a vector space of dimension N over \mathbb{F} .

A. Golomb's Three Randomness Postulates for Binary Sequences

Run: For a binary sequence \mathbf{a} with period N , k consecutive zeroes (or ones) preceded by one (or zero) and followed by one (or zero) is called a *run of zeroes (or ones) of length k* .

Autocorrelation and crosscorrelation: Let \mathbf{a} and \mathbf{b} be two binary sequences with period N , the crosscorrelation of \mathbf{a} and \mathbf{b} , denoted as $C_{\mathbf{a},\mathbf{b}}(\tau)$, is defined as

$$C_{\mathbf{a},\mathbf{b}}(\tau) = \sum_{i=0}^{N-1} (-1)^{a_i + b_{i+\tau}}, \quad (7)$$

i.e., $C_{\mathbf{a},\mathbf{b}}(\tau)$ is the (unnormalized) dot product of two vectors:

$$((-1)^{a_0}, (-1)^{a_1}, \dots, (-1)^{a_{N-1}}) \text{ and } ((-1)^{b_\tau}, (-1)^{b_{\tau+1}}, \dots, (-1)^{b_{\tau+N-1}}).$$

Here the indices are reduced by modulo N . When $\mathbf{b} = \mathbf{a}$, it is called autocorrelation of \mathbf{a} and denoted as $C_{\mathbf{a}}(\tau)$ or $C(\tau)$ if the context is clear. The autocorrelation $C(\tau)$ measures the amount of similarity between the sequence and its shift. This is always highest for $\tau = 0$, since $C(0) = \sum_{i=0}^{N-1} (-1)^{a_i + a_i} = N$.

Example 6 Let $\mathbf{a} = 1001011$ and $\mathbf{b} = 1110100$. Then we have

$$C_{\mathbf{a}}(\tau) = \begin{cases} 7 & \text{for } \tau \equiv 0 \pmod{7} \\ -1 & \text{for } \tau \not\equiv 0 \pmod{7} \end{cases}$$

| | | | | | | | |
|-----------------------------------|----|---|---|----|---|----|----|
| τ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| $C_{\mathbf{a},\mathbf{b}}(\tau)$ | -5 | 3 | 3 | -1 | 3 | -1 | -1 |

Golomb proposed the following three randomness postulates to measure randomness of binary periodic sequences.

R-1 In every period, the number of zeroes is nearly equal to the number of ones. (More precisely, the disparity is not to exceed 1, i.e., $\left| \sum_{i=0}^{N-1} (-1)^{a_i} \right| \leq 1$.)

R-2 In every period, half the runs have length one, one-fourth have length two, one-eighth have length three, etc., as long as the number of runs so indicted exceeds 1. Moreover, for each of these lengths, there are equally many runs of 0's and of 1's.

R-3 The autocorrelation function $C(\tau)$ is two-valued, given by

$$C(\tau) = \begin{cases} N & \text{if } \tau \equiv 0 \pmod{N} \\ K & \text{if } \tau \not\equiv 0 \pmod{N} \end{cases} \quad (8)$$

where K is a constant. If $K = -1$ for N odd and $K = 0$ for N even, then we say that the sequence has the *(ideal) 2-level autocorrelation function*.

If $N = 2^n - 1$, we have the following refined postulates.

R-1 In every period, zeroes occur $2^{n-1} - 1$ (or 2^{n-1}) times and ones occur 2^{n-1} (or $2^{n-1} - 1$) times. This property is referred to as the *balance property*.

R-2 In every period, runs of 0's (or 1's) of length $k : 1 \leq k \leq n - 2$ occur 2^{n-2-k} times. A run of 0's of length $n - 1$ occurs once and a run of 1's of length n occurs once. This is referred to as the *run property*.

R-3 The auto-correlation function $C(\tau)$ is two-valued, and is given by

$$C(\tau) = \begin{cases} 2^n - 1 & \text{if } \tau \equiv 0 \pmod{2^n - 1} \\ -1 & \text{if } \tau \not\equiv 0 \pmod{2^n - 1}. \end{cases} \quad (9)$$

Example 7 We consider the following two m -sequences.

(a) Let $\mathbf{a} = (1110010)$ with the minimal polynomial $f(x) = x^3 + x + 1$. There are four 1's and three 0's, which is adequate to satisfies $R-1$. Of the four runs, half have length one, and one-fourth have length two. The autocorrelation is 7 in phase and -1 out of phase. Thus, this sequence satisfies $R-1$, $R-2$, and $R-3$.

(b) Let $\mathbf{a} = (000100110101111)$ with the minimal polynomial $f(x) = x^4 + x + 1$. It can be easy verified that this sequence also satisfies $R-1$, $R-2$ and $R-3$.

2.2 Linear Spans

If a sequence \mathbf{a} generated by an LFSR with characteristic polynomial $f(x)$ of degree n , and no other LFSR with the number of stages less than n could generate the sequence, then $f(x)$ is called the *minimal polynomial* of \mathbf{a} and the degree of the minimal polynomial of \mathbf{a} is called a *linear span* of \mathbf{a} , denoted as $LS(\mathbf{a})$. Thus the linear span of \mathbf{a} is the shortest length of LFSR that generates \mathbf{a} . Given any binary sequence of length N , the linear span of the sequence can be computed by the Berlekamp-Massey algorithm (BMA). Furthermore, the minimal polynomial which generates the sequence also can be computed by the Berlekamp-Massey algorithm.

Remark 2 The BMA is an iterative algorithm, similarly like Euclidean algorithm. Applying the BMA to a given sequence \mathbf{a} of length N , the output is a characteristic polynomial of an LFSR with the shortest length which generates \mathbf{a} if the length N is large enough. Or equivalently, the output is the minimal polynomial of the sequence.

1. If the linear span of \mathbf{a} is n and $2n < N$, then the LFSR with the shortest length which generates \mathbf{a} can be computed from $2n$ consecutive elements of \mathbf{a} .
2. If \mathbf{a} is used as a key stream in a cryptosystem, then computing an LFSR with the shortest length which generates \mathbf{a} is known as *linear span attack*.

For example, the m -sequences shown in Example 6 have linear spans 3, 4, 5 and 6, respectively. Thus, by knowing consecutive 12 bits of the m -sequence of period 63, one can recover the rest of bits in the sequence. This shows that m -sequences are not secure for use in a cryptosystem.

2.3 One-time-pad and Randomness Criteria

One-time-pad means that different messages are encrypted by different key streams. In 1948, Shannon showed that one-time-pad is unbreakable in his milestone paper. This requests that a key stream should have a large period. In 1968, Berlekamp [?] found a decoding algorithm. Shortly after that (1969) Massey [?] discovered that if a binary sequence has linear span n , then the entire sequence can be reconstructed from $2n$ consecutive known bits by the Berlekamp algorithm. This result imposes another condition on the design of PRNG, i.e., it requests a key stream should have a large linear span. When the Berlekamp algorithm is applied to synthesis of random sequences, it is referred to as *Berlekamp-Massey algorithm* since then.

In general, we have the following criteria to determine whether a given sequence is random or unpredictability.

1. Long period.

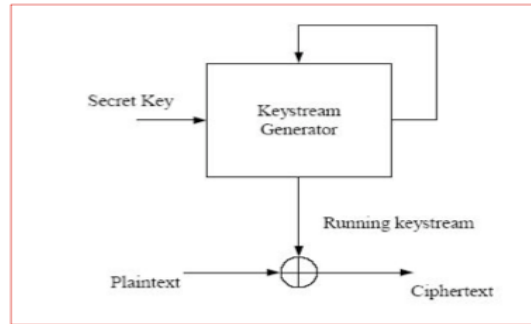


Figure 9: A General Model of Stream Cipher

2. Statistic properties:

- (a) Balanced property: each element occurs nearly equally many times.
- (b) Run property R2.
- (c) k -tuple distribution: for $k = \log N$, each binary k -tuple d_0, d_1, \dots, d_{k-1} occurs nearly equally many times in one period.

3. Correlation

- (a) 2-level autocorrelation.
- (b) Low crosscorrelation: $|C_{\mathbf{a},\mathbf{b}}(\tau)| \leq c\sqrt{N}$ where c is constant.

4. Large linear span:

$$t(N/2) \leq LS(\mathbf{a}) \leq N, \text{ where } t \text{ is constant with } 0 < t \leq 1 \quad (10)$$

Let $\rho = LS(\mathbf{a})/N$, which is referred to as a *normalized linear span* of \mathbf{a} . Then (10) becomes that $t/2 \leq \rho \leq 1$.

- 5. Indistinguishability: Indistinguishability of a pseudo-random sequence means that a sequence, schematically generated by an algorithm or a device cannot be distinguished from a truly random sequence in terms of any polynomial algorithm with negligible probability.

3 Nonlinear Generators

Linear feedback shift register (LFSR) sequences are widely used as basic functional blocks in key stream generators in stream cipher models due to their fast implementation in hardware as well as in software in some cases. In LFSR based stream ciphers, there are mainly two types of operations which operate on the LFSRs:

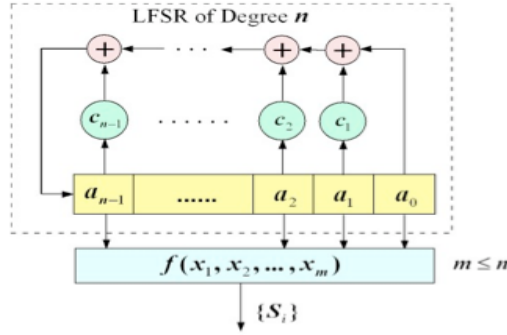


Figure 10: A Diagram of a Filtering Sequence Generator

- a) outputs of one LFSR or multiple LFSRs are transformed by nonlinear functions with or without memory states, including those by using mixed finite field operations and integer modular operations (filtering or combinatorial generators);
- b) change the clock of the LFSRs to an irregular clock or by deleting some output bits of the LFSRs (clock control or shrinking generators).

In practice, such key stream generators include E0 in Bluetooth, and most of the submissions in EStream Project. In this type of key stream generators, *initial states* of the LFSRs are served as a cryptographic key in each communication session. The goal of an attack is to recover the key, i.e., the initial states of the LFSRs from some *known bits* of the key stream, then generates the rest of bits of the key stream used in that particular session. There are many attacks on the LFSR based key stream generators recorded in the literature. In next section, we will consider attacks related to solve a system of linear equations, the so called algebraic attacks and its variants, in depth.

3.1 Filtering Sequence Generators

A. Scheme

Let $\mathbf{w} = \{w_t\}$ be an m -sequence of period $2^n - 1$, and $0 \leq d_0 < d_1 < \dots < d_{m-1} < n$. A sequence $\mathbf{s} = \{s_t\}$ is referred to as a filtering sequence if

$$s_t = f(w_{d_0+t}, w_{d_1+t}, \dots, w_{d_{m-1}+t}), t = 0, 1, \dots \quad (11)$$

where $f(x_0, x_1, \dots, x_{m-1})$ is a boolean function in m variables. The boolean function f is referred to as a *filtering function*.

B. Randomness Profile of Filtering Sequences

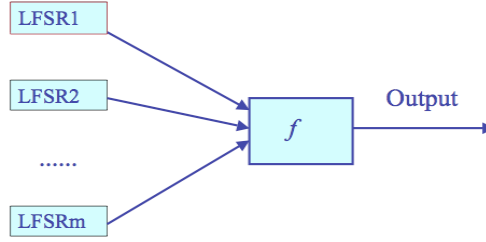


Figure 11: A Diagram of the Combinatorial Sequence Generator

1. The period of \mathbf{s} is given by $2^n - 1$.
2. The linear span of \mathbf{s} , $LS(\mathbf{s})$, is upper bounded by

$$LS(\mathbf{s}) \leq \sum_{k=1}^m \binom{n}{k}.$$

In particular, if $m = 2$, then $LS = n(n + 1)/2$.

3. If d_i 's are equally spaced, then the linear span will be lower bounded by

$$LS(\mathbf{s}) \geq \binom{n}{m}.$$

4. It is not clear for other randomness properties.

3.2 Combinatorial Sequence Generators

A. Scheme

Let

$$s_t = f(w_{0,t}, w_{1,t}, \dots, w_{m-1,t}), t = 0, 1, \dots \quad (12)$$

where $\mathbf{w}_i = \{w_{i,t}\}_{t \geq 0}$ are m -sequences with period $2^{n_i} - 1$, respectively. Then $\mathbf{s} = \{s_t\}$ is called a *combinatorial sequence*.

Remark 3 If we apply a boolean function in m variables to m LFSRs which generates m -sequences, then the output sequence of the boolean function is a combinatorial sequence. Note that when the m LFSRs have the same characteristic primitive polynomials, then a combinatorial sequence becomes a filtering sequence. Thus a filtering sequence generator is a special case of combinatorial sequence generators.

B. Randomness Profile of Combinatorial Sequences

a) Period: The period of \mathbf{s} , $per(\mathbf{s})$, is a factor of the least common multiple of $2^{n_i} - 1$, $i = 1, \dots, m$, i.e.,

$$per(\mathbf{s}) \mid \text{lcm}\{2^{n_i} - 1 \mid i = 1, \dots, m\}.$$

If $\gcd(n_i, n_j) = 1$ for any $i \neq j$, let $N = \prod_{i=1}^{m-1} (2^{n_i} - 1)$, then N is a period of \mathbf{s} . But it may not be the least period of \mathbf{s} . In the following, we only consider this special case. Furthermore, if the periods of any pair of m -sequences are coprime, then N is the least period of \mathbf{s} when all the LFSRs generates nonzero sequences.

b). Linear Span: Let the algebraic normal form of f be given as follows

$$f(x_0, \dots, x_{m-1}) = \sum_{(i_1, \dots, i_r)} x_{i_1} \cdots x_{i_r} \quad (13)$$

summed over some vectors (i_1, \dots, i_r) with

$$\{i_1, \dots, i_r\} \subset \mathbb{Z}_m = \{0, 1, \dots, m-1\}, r \leq m.$$

Let D be the set consisting of all the vectors (i_1, \dots, i_r) for which $x_{i_1} \cdots x_{i_r}$ is a monomial term of f . If $\gcd(n_i, n_j) = 1$ for any $i \neq j$, then the linear span of \mathbf{s} is equal to

$$LS(\mathbf{s}) = \sum_{(i_1, \dots, i_r) \in D} n_{i_1} \cdots n_{i_r}.$$

Example 8 Let LFSR i , $i = 1, 2, 3$ have their respective characteristic polynomials $f_1(x) = x^2 + x + 1$, $f_2(x) = x^3 + x + 1$ and $f_3(x) = x^5 + x^3 + 1$. Those LFSRs generate m -sequences of period 3, 7 and 31 respectively. Let $\{a_i\}$, $\{b_i\}$ and $\{c_i\}$ be the outputs of those three LFSRs, and $f(x_0, x_1, x_2)$. Then a combinatorial sequence $\{s_t\}$ is given by

$$s_t = a_t + b_t + c_t, t = 0, 1, \dots,$$

as shown in Figure 12. The period of an output sequence $\{s_t\}$ is given by

$$(2^2 - 1)(2^3 - 1)(2^5 - 1) = 651$$

when each LFSR is loaded nonzero initial state. The linear span of the output sequence is 10. (Can you comment that since there used 10 registers, why not directly uses an LFSR with primitive feedback which produces an m -sequence with period $1023 > 651$ and linear span 10, the same as the combinatorial sequence?)

Figure 12: A Diagram of a Combinatorial Generator with Three LFSRs

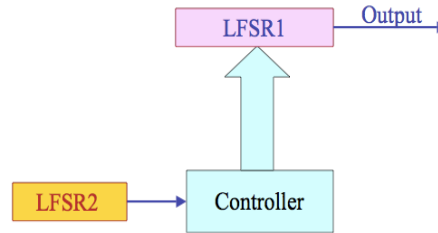


Figure 13: Model of the Clock Controlled Generator and Shrinking Generator

C. How to select f in both filtering and combinatorial generators?

In general, one should consider that the output sequences satisfy as many as the requirements listed in Section 2. Another consideration is to resistance to known attacks. We list some criteria as follows.

1. f should have arge degree, in order to have large linear span.
2. In order to resistance to correlation attack (Siegenthaler, 1984), f should be balanced, and has low correlation from all affine function (so-called nonlinearity), thus large correlation immunity. (Many tricks here for selection of f !)
3. For resistance to *linear cryptanalysis*, f cannot be approximated by any affine functions with high probability.
4. There is a trade-off between degree and correlation immunity, i.e., one cannot make them both large.
5. In order to be resistant to *differential cryptanalysis*, it requests that difference between $f(\mathbf{x}+\mathbf{a})$ and $f(\mathbf{x})$ should be large, i.e., the correlation between $f(\mathbf{x} + \mathbf{a})$ and $f(\mathbf{x})$ should be small.
6. It should also have large *algebraic immunity*, i.e., the degree of the product of f and any boolean functions should be at least the degree of f .

3.3 Clock-control Generators and Shrinking Generators

A. Clock Controlled Generator: The basic idea is to change the clock pulse in LFSRs. For example, the clock pulse of LFSR 1 is controlled by LFSR 2.

Stop-and-go generator: Let two sequences $\mathbf{a} = \{a(t)\}$ and $\mathbf{b} = \{b_i\}$ be generated by LFSR1 and LFSR 2 respectively. The output sequence is denoted as $\mathbf{u} = \{u(t)\}$. At time instance t ,

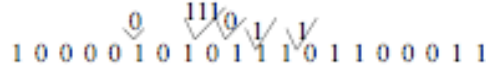


Figure 14: Inserting Operation in the Clock-control Generator

assume that the previous bit is $u(t - 1) = a(i_t - 1)$, then

$$u(t) = \begin{cases} a(i_t) & \text{if } b_t = 1 \\ a(i_t - 1) & \text{if } b_t = 0. \end{cases}$$

In other words, if $b_t = 1$, then the generator outputs the current bit $a(i_t)$ at LFSR1. Otherwise, the generator repeats the previous output bit $a(i_t - 1)$ (equivalent to an inserting operation) of LFSR1. Furthermore, in formula, we have

$$u(t) = a\left(\sum_{i=0}^t b_i\right), t = 0, 1, \dots$$

Example 9 Let two sequences **a** and **b** be given by

$$\begin{aligned} \mathbf{a} &= 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ \dots \\ \mathbf{b} &= 1\ 1\ 1\ 1\ 1\ 1\ \underline{0}\ 1\ 1\ 1\ \underline{0}\ \underline{0}\ \underline{0}\ 1\ \underline{0}\ 1\ \underline{0}\ 1\ 1\ \underline{0}\ \underline{0}\ \dots \end{aligned}$$

The inserting process is shown in Figure 9. The first 20 output bits are:

$$1\ 0\ 0\ 0\ 0\ \underline{0}\ 1\ 0\ 1\ \underline{111}\ 0\ \underline{0}\ 1\ \underline{1}\ 1\ 1\ \underline{1}\ 0$$

The underlined bits are the ones who have been inserted.

B. Shrinking Generators: Suppose that two input sequences **a** and **b** are the same as those in the stop-and-go generator. The output is $\mathbf{u} = \{u(t)\}$ whose elements are given as follows.

- (a) Let the previous output be $u(i - 1)$ where $i = \sum_{j=0}^{t-1} b_j$ with the initial state $u(0) = a(s)$ where $b_0 = b_1 = \dots = b_{s-1} = 0$ and $b_s = 1$.
- (b) At time instance t , if $b_t = 1$, then the generator outputs the current bit $a(t)$ of LFSR1, i.e., $u(i) = a(t)$, $i > 0$. Otherwise, the generator discards $a(t)$. In other words, for $i = \sum_{j=0}^t b_j$, $i > 0$, we have

$$u(i) = \begin{cases} a(t) & \text{if } b_t = 1 \\ \text{discards } a(t) & \text{if } b_t = 0. \end{cases} \tag{14}$$

Example 10 Let \mathbf{a} and \mathbf{b} be given by

$$\begin{aligned} \mathbf{a} &= 1 \ 0 \ \cancel{0} \ \cancel{0} \ \cancel{0} \ \cancel{1} \ 0 \ 1 \ 0 \ \cancel{1} \ \cancel{1} \ \cancel{1} \ 0 \ \cancel{1} \ 1 \ \cancel{0} \ 0 \ 0 \ \cancel{1} \ 1 \ \dots \\ \mathbf{b} &= 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ \dots \end{aligned}$$

Deleting Operation in the Shrinking Generator

The bits in \mathbf{a} which correspond to zeros in \mathbf{b} are deleted, which are marked. The first 10 output bits from the shrinking generator are those which are not marked, i.e.,

$$1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1.$$

4 Blum-Blum-Shub (BBS) Generators

A. $x^2 \pmod N$ Generator:

Let $N = pq$ where p and q are distinct primes $\equiv 3 \pmod 4$. Inputs are (N, x_0) where x_0 is referred to as a *seed* of the generator. The outputs of the generator is a pseudo-random sequence $\mathbf{s} = \{s_i\}$ whose elements are given by

$$\begin{aligned} \text{Computing the integer: } & x_{i+1} = x_i^2 \pmod N, i = 0, 1, \dots \\ \text{Extracting the bit: } & s_i = \text{parity}(x_i) \end{aligned}$$

where the parity function is defined by

$$\text{parity}(x) = \begin{cases} 1 & \text{if } x \text{ is odd} \\ 0 & \text{if } x \text{ is even} \end{cases}$$

B. BBS as a Filtering Model

The BBS generator can be considered as a filtering generator where the LFSR is replaced by an NLFSR of one stage with the feedback function $x^2 \pmod N$, and the filtering function is the parity check function which maps $\log N$ bits to one bit, which is shown in Figure 15.

Example 11 Let $N = 7 \times 19 = 133$ and $x_0 = 4$. Then the sequence $x_0, x_1 = x_0^2 \pmod{133}, \dots$ has period 6:

$$\begin{aligned} \{x_i\} &= 4, \ 16, \ 123, \ 100, \ 25, \ 93, \ \dots \\ \{s_i\} &= 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ \dots \end{aligned}$$

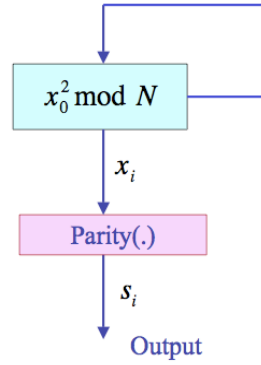


Figure 15: A Diagram of BBS as a Filtering Generator

Note that $\lambda(133) = 18$ and $\lambda(\lambda(133)) = 6$ where $\lambda(n)$ is defined below. The period of the sequence is 6.

C. Property of BBS Generators

The BBS generator is unpredictable, in the sense that given x_0 and N , but not the factors of N , a cryptanalyst cannot effectively make any prediction about the values of x_{-1}, x_{-2}, \dots . Or equivalently, given a portion of the key stream $\{s_i\}$, it is computational infeasible to recover the seed x_0 by knowing N without knowing the factors of N . For positive integers

$$n = 2^e p_1^{e_1} \cdots p_k^{e_k}, \text{ where } p_1 < p_2 < \cdots < p_k \text{ are odd primes,}$$

we define the lambda function $\lambda(n)$ by

$$\begin{aligned} \lambda(2) &= 1, \lambda(4) = 2, \lambda(2^e) = 2^{e-2}, e > 2 \\ \lambda(p^r) &= \phi(p^r) = p^{r-1}(p-1) \text{ for } p \text{ odd} \\ \lambda(n) &= \text{lcm}\{\lambda(2^e), \lambda(p_1^{e_1}), \dots, \lambda(p_k^{e_k})\}. \end{aligned}$$

If x_0 is a quadratic residue modulo N , then the period of the BBS sequence \mathbf{s} , $per(\mathbf{s})$, satisfies

$$per(\mathbf{s}) \mid \lambda(\lambda(N)).$$

The imbalanced range of \mathbf{s} is defined by

$$I(\mathbf{s}) = | \text{number of 0's in } \mathbf{s} - \text{number of 1's in } \mathbf{s} |.$$

The average of imbalance is with order no larger than $N^{1/4} \log N$.

Note that the average of imbalance for a random sequence with period $\lambda(\lambda(N))$ is with order \sqrt{N} , which is much bigger than $N^{1/4} \log N$. In this sense, the BBS generator is better than the random sequences of the same period $\lambda(\lambda(N))$.

5 Known Attacks

For pseudo-random sequence generators, especially LFSR based generators, a number of attacks have been found, and some of them are efficient enough to break systems.

1. Correlation attacks: recovering keys through computing correlation between key stream sequences, treated as functions, and linear functions.
2. Linear cryptanalysis: finding linear relation among key streams, and then approximating the cipher functions by these linear relations for recovering keys.
3. Differential cryptanalysis: look at difference between a function and its time shift of the function for recovering keys.
4. Time and memory trade-off attacks: exhaustively searching for N possible solutions is done through finding the solution in T operations (time) with M words of memory, provided the time-memory product TM equals N .
5. Algebraic attacks: recovering keys in a large system of linear equations (obtained by linearization) by transferring it into a small system of linear equations.

In the following, we introduce this type of attacks in more details.

5.1 A Glance at Algebraic Attacks

The goal is to find the seed in the pseudorandom sequence (number) generator (PRSG), i.e., the initial state in the LFSRs based generator. The evolution of series of algebraic attacks is as follows.

- Linearizations
- Algebraic Attacks
- Fast Algebraic Attacks
- Selective Discrete Fourier Transform (DFT) Attacks (including special case of linear subspace attacks)

Algebraic attacks and fast algebraic attacks have been shown as an important cryptanalysis method for symmetric-key cryptographical systems in recent work by many researchers. Especially, it significantly improved efficiency of attacks on stream cipher systems in which key streams are

generated by linear feedback shift register based systems. Those attacks usually contain three steps: (a) pre-computation, (b) substitution for establishing a system of linear equations from known key stream bits (linearization), and (c) solving the system.

5.2 Equations with Unknown Keys

In a stream cipher model, a ciphertext is a bit stream c_0, c_1, \dots , obtained by exclusive-or a message bit stream m_0, m_1, \dots , with the key stream s_0, s_1, \dots , i.e.,

$$c_i = m_i + s_i, i = 0, 1, \dots, \text{ in } \mathbb{F}_2.$$

One of strong attacks comes from known plaintext attacks, i.e., if a certain plaintext is known, then some bits of the key stream $\{s_t\}$ can be recovered. If the key can be recovered from those known bits of $\{s_t\}$, then the rest of bits of the key stream, i.e., all bits of $\{s_t\}$, can be reconstructed.

A. Known plaintext Attack: An initial state of the LFSR is a key when $\{s_t\}$ is served as a key stream generator, denoted by $K = (k_0, \dots, k_{n-1}), k_i \in F$. At time instance t , f operates on the state $(s_t, s_{t+1}, \dots, s_{t+n-1})$ which is a shift of t from the initial state of the LFSR. Thus, in terms of the shift operator L , we may write $(s_1, \dots, s_n) = L(k_0, \dots, k_{n-1}), (s_t, s_{t+1}, \dots, s_{t+n-1}) = L^t(k_0, \dots, k_{n-1}), t \geq 0$ where $(s_0, \dots, s_{n-1}) = (k_0, \dots, k_{n-1})$. Then the t term of the filtering sequence is given by

$$s_t = f(L^t(k_0, k_1, \dots, k_{n-1})) = f_t(k_0, \dots, k_{n-1}), t = 0, 1, \dots \quad (15)$$

where

$$f_t(x_0, \dots, x_{n-1}) = f(L^t(x_0, \dots, x_{n-1})) = f(L^t(\mathbf{x})), \mathbf{x} = (x_0, \dots, x_{n-1}).$$

The equation (15) shows how the output sequence of the filtering sequence is related to the key. If a certain plaintext is known, then some bits of $\{s_t\}$ can be recovered. If the key can be recovered from those known bits of $\{s_t\}$, then the rest of bits of the key stream can be reconstructed. This attack is referred to as a *known plaintext attack*.

B. Linearization

Question: How can one efficiently solve (15) with minimized known bits of the key stream?

The system of the equations (15) can be linearized when each monomial in $k_{i_1} \cdots k_{i_s}$ is treated as a variable. The number of unknowns in (15) is varied, but it is dominated by the degree of f . For filtering function generators, i.e., apply f on m tap positions of an LFSR of degree n , the

number of unknown in (15) is upper bounded by $T_{deg(f)}$ where $deg(f)$ is the degree of f and T_j is defined as

$$T_j = \sum_{i=0}^j \binom{n}{i}. \quad (16)$$

Thus, by linearization, we can establish a system of linear equations in $T_{deg(f)}$ unknowns. In order to solve the system, one needs $T_{deg(f)}$ bits from the key stream $\{s_t\}$.

5.3 Algebraic Attacks

A. Attack Method

The algebraic attack is to multiply f by a function g with a degree lower than f such that the product fg is zero. In other words, using g with $deg(g) < deg(f)$ such that $fg = 0$, we have the system of the linear equations as follows

$$s_t g_t(K) = 0, t = 0, 1, \dots \quad (17)$$

In this case, the number of the unknowns of (17) is now dominated by the degree of g instead of f , which is decreased. For example, in the filtering generators, this is equal to $T_{deg(g)}$. However

$$T_{deg(g)} < T_{deg(f)}.$$

Compared with the system of the linear equations directly from linearization, the algebraic attack can reduce the complexity for solving a system of linear equations by decreasing the number of unknowns in the system of linear equations as well as reducing the number of the required known bits of the key stream. From this result, study for algebraic immunity of boolean functions is in fashion for resistance against this attack.

B. Algebraic Immunity

Algebraic immunity is to measure how a function can be resistant to algebraic attacks. Let \mathcal{B}_n be the set consisting of all boolean functions in n variables. The *algebraic immunity* of f is defined as the smallest degree $deg(g)$ such that $fg = 0$ or $(1+f)g = 0$, denoted by $AI(f)$, i.e.,

$$AI(f) = \min_{g \in Ann(f)} deg(g), \quad (18)$$

where

$$Ann(f) = \{g \in \mathcal{B}_n \mid fg = 0 \text{ or } g(f+1) = 0\}$$

which is called the *annihilator* of f .

5.4 Fast Algebraic Attacks

The fast algebraic attack (FAA) (2003) on stream ciphers is to accelerate the algebraic attack by introducing linear relations among the key stream bits. The idea is to take some trade-off between the complexity of solving the system of linear equations and the number of the known bits from key streams.

We take a function from the annihilator of f , say w such that $w = u + g$ where $\deg(g) < \deg(f)$, possibly, $\deg(w) > \deg(f)$. Thus we have

$$fw = 0 \implies f(u + g) = 0 \implies fu = fg.$$

By letting $h = fu$, we have $h = fg$ where $\deg(g) < AI(f)$. Then $\deg(h) \geq AI(f)$. In this way, one could further reduce the number of the unknowns in the linear equations.

FAA consists of two steps:

- (a) Find a boolean function g with $d = \deg(g) < \deg(f)$ such that the product $h = fg \neq 0$ with $e = \deg(h) > 0$ where $d < e$;
- (b) Compute $q(x)$ which is a characteristic polynomial of the output sequence or a factor of it, and apply $q(x) = \sum_i c_i x^i$ to $s_t g_t(K) = h_t(K)$ which results in

$$\sum_{i=0}^r c_i s_{i+t} g_{i+t}(K) = \sum_{i=0}^r c_i h_{i+t}(K). \quad (19)$$

We denote $\Delta_i(x_0, x_1, \dots, x_{n-1})$ be the boolean function containing all the monomial terms with degree less than or equal to i , and denote $\Delta_i(K)$ the filtering sequence resulting by applying $\Delta_i(x_0, \dots, x_{n-1})$ to the LFSR with the initial state K . If $v_t = \sum_{i=0}^r c_i h_{i+t}(K)$, $t = 0, 1, \dots$ is equal to zero, then (19) is the system of linear equations in at most T_d variables which can be solved by known T_e consecutive bits. If we choose $q(x)$ as the quotient of the minimal polynomial of $\Delta_e(K)$ dividing by the minimal polynomial of $\Delta_d(K)$. Then $\{v_t\}$ is a nonzero sequence. Hence (19) can be solved by known $T_e - T_d$ consecutive bits, which is less than the case that $\{v_t\}$ is a zero sequence.

According to the above analysis, the number of unknowns in (19) is less than the number of unknowns in the algebraic attack. But in FAA, the known bits of the key stream should be consecutive. However this condition is not needed in the algebraic attack.

5.5 Selective DFT Attacks

More recently, it is proposed a new method to recover an initial state in a filtering sequence generator by reducing the number of unknowns in the system of linear equations to the minimum, which is

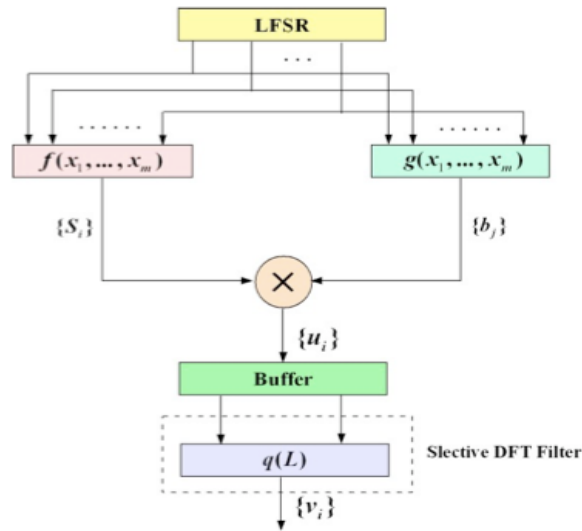


Figure 16: Model of Solving Equations Related Attacks on the Filtering Generator

the degree of the LFSR by an increased complexity in the both pre-computation and substitution, especially, in the substitution step.

In those methods, the pre-computation and substitution are done by forming a system of linear equations over $GF(2^n)$, an extension field of F , instead of linear equations over F . The number of the required consecutive bits, say j , is reduced to the linear span of the filtering sequence, denoted as $LS(\mathbf{s})$. If the number of the required consecutive bits is greater than the linear span of the sequence \mathbf{s} , i.e., $j \geq LS(\mathbf{s})$, then the number of the unknowns is reduced to n which is the degree of the LFSR. If $j < LS(\mathbf{s})$, the FAA fails. However, the system of the equations is solvable by the selective DFT method.

The core part of the selective DFT is to choose different $q(x)$, which results an attack of either more efficient than FAA or it can work for the case that the number of the known consecutive bits of the key stream is not adequate to apply FAA. However, this method needs to know the exact linear span of the key stream sequence $\{s_t\}$ and their respective discrete Fourier transform (DFT) spectra of the multiplier sequence and the product sequence, which are not needed in FAA.

All the solving equations related attacks can be considered as a certain special case of the selective DFT attacks. A general model is shown in Figure 16.

Notes

The research of shift register sequences are pioneered by Golomb, and the earlier research work in this areas are collected in his book [12], later on some appeared in [29], and more recent results

about correlation in [13]. Berlekamp discovered his algorithm in 1968 [3], and Massey used this algorithm for sequence synthesis in 1969 [20]. LFSR based PRSGs started in the early of 1970s, see [15] [26] for combinatorial generators, [19] [17] for filtering sequences, [2][16] [14] for clock-controlled sequences, and [6][21] for shrinking generators. The Blum-Blum-Shub computational hard problem based PRSGs was appeared in [5] and some randomness properties are discussed in [9]. Correlation attacks are introduced in [27] [28], linear cryptanalysis in [22], time and memory trade-off attacks in [18], and algebraic attacks and fast algebraic attacks are introduced [7] [1] [8], and special cases of selective DFT attacks are presented in [23] [24]. The proposals of practical ciphers are documented in [4] [10].

References

- [1] F. Armknecht and M. Krause, Algebraic attacks on stream combiners with memory, *Advances in Cryptology CRYPTO 2003*, Lecture Notes in Computer Science, No. 2729, pp. 162-176, Springer-Verlag, 2003.
- [2] T. Beth and F. Piper, The stop-and-go generator, *Advances in Cryptology, Eurocrypt'94*, vol. 209, Springer-Verlag, 1985, pp. 88-92.
- [3] E.R. Berlekamp, *Algebraic coding theory*, New York, McGraw-Hill, 1968.
- [4] Bluetooth CIG, Specification of the Bluetooth system, Version 1.1, February 22, 2001. Available from www.bluetooth.com.
- [5] L. Blum, M. Blum, and M. Shub, A Simple Unpredictable Pseudo-Random Number Generator, *SIAM J. Comput.*, vol.15, No. 2, 1986. pp.364-383.
- [6] D. Coppersmith, H. Krawczyk and Y. Mansour, The shrinking generator, *Advances in Cryptology-Crypto'93*, D. R. Stinson (ed.), LNCS 773, Springer-Verlag, 1994, pp. 22-39.
- [7] N. Courtois, Fast algebraic attacks on stream ciphers with linear feedback, *Advances in Cryptology-Crypto'2003*, Lecture Notes in Computer Science, vol. 2729, pp. 176-194, Springer-Verlag, 2003.
- [8] N. Courtois and W. Meier, Algebraic attacks on stream ciphers with linear feedback, *Advances in Cryptology-Eurocrypt'2003*, Lecture Notes in Computer Science, vol. 2656, pp. 345-359, Springer, 2003.
- [9] T. W. Cusick, Properties of the $x^2 \bmod N$ pseudorandom number generator, *IEEE Trans. on Inform. Theory* vol. IT-41, No. 4, July 1995, pp. 1155-1159.

- [10] eSTREAM - *The ECRYPT Stream Cipher Project*, <http://www.ecrypt.eu.org/stream/>
- [11] Philip Hawkes and Gregory G. Rose, Rewriting variables: the complexity of fast algebraic attacks on stream ciphers, *Advances in Cryptology-Crypto'2004*, Lecture Notes in Computer Science, No. 3152, pp. 390-406, Springer-Verlag, 2004.
- [12] S.W. Golomb, *Shift Register Sequences*, Holden-Day, Inc., San Francisco, 1967, revised edition, Aegean Park Press, Laguna Hills, CA, (1982).
- [13] S.W. Golomb and G. Gong, *Signal Design with Good Correlation: for Wireless Communications, Cryptography and Radar Applications*, Cambridge University Press, 2005.
- [14] D. Gollman and W. G. Chambers, Clock-controlled Shift Register: A Review, *IEEE Journal on Selected Areas in Communications*, Vol. 7, pp. 525-533, May, 1989.
- [15] E. J. Groth, Generation of Binary sequences with controllable complexity, *IEEE Transactions on Information Theory*, IT-17, pp. 288-296, May 1971.
- [16] C.G. Gunther, Alternating Step Generators Controlled by de Bruijn Sequences, *Advances in Cryptology-Eurocrypt'88*, S. Goldwasser (Ed.), LNCS 403, Springer-Verlag, 1988, pp. 88-92.
- [17] T. Herlestam, On Functions of Linear Shift Register Sequences, *Advances in Cryptology - Eurocrypt 1985 (Ed: Franz Pichler)*, Springer Lecture Notes in Computer Science, LNCS, vol. 0219, pp. 119-129, 1985.
- [18] M. E. Hellman, A Cryptanalytic Time-Memory Trade-Off, *IEEE Transactions on Information Theory*, Vol. IT-26, No. 4, July 1980, pp. 401-406.
- [19] E. L. Key, Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators, *IEEE Transactions on Information Theory*, IT-22, pp. 732-736, Nov. 1976.
- [20] J. L. Massey, Shift-register synthesis and BCH decoding, *IEEE Trans. Information Theory*, Vol. 15, No. 1, pp. 122-127, January 1969.
- [21] W. Meier and O. Staffelbach, The self-shrinking generator, *Advances in Cryptology, Eurocrypt'94*, Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 205-214.
- [22] M. Matsui, Linear Cryptanalysis Method for DES cipher, *Advances in Cryptology-Eurocrypt93*, Lecture Notes in Computer Science, Springer-Verlag, 1993.
- [23] S. Rønjom and T. Hellesest, A New Attack on the Filter Generator, *IEEE Transactions on Information Theory*, vol. 53, no. 5, pp. 1752-1758, 2007.

- [24] S. Rønjom, G. Gong and T. Helleseth, On attacks on filtering generators using linear subspace structures, *Sequences, Subsequences, and Consequences, Lecture Notes in Computer Science*, S.W. Golomb *et al.* (Eds.), vol. 4893, pp. 204-217. Springer-Verlag, 2007.
- [25] R.A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.
- [26] R.A. Rueppel and O.J. Staffelbach, Products of linear recurring sequences with maximum complexity, *IEEE Transactions on Information Theory*, vol. 33, no. 1, pp. 124-131, January, 1987.
- [27] T. Siegenthaler, Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications, *IEEE transactions on information theory*, Vol. IT-30, No.5, September 1984.
- [28] T. Siegenthaler, Decrypting a Class of Stream Cipher Using Ciphertext Only, *IEEE Transactions on Computers*, Vol. 34, No. 1, January, 1985.
- [29] M.K. Simon, J.K. Omura, R.A. Scholtz and B.K. Levitt, *Spread Spectrum Communications Handbook*, McGraw-Hill, Inc., Revised version, 1994. Chapter 5.