Chapter 1
Introduction to Stream Ciphers

The word cryptology comes from two Greek roots meaning "hidden" and "word", and is the generic name used to describe the entire field of secret communications. Cryptology clearly splits into two opposite but complementary disciplines: cryptography and cryptanalysis. Cryptography seeks methods to ensure the secrecy of a confidential message while cryptanalysis seeks to break such methods in order to recover the confidential message. In fact, the original message upon which the cryptographer applies the cryptographic transformation is called the plaintext message, or simply the *plaintext*. The result of this transformation is called the ciphertext message, or simply the *ciphertext*, or most often the *cryptogram*. In order to control the enciphering process, the cryptographer always makes use of an exclusive information, the *key*. The general assumption in cryptology is that the cryptanalyst has full access to the cryptogram. Moreover, at present the Kerckhoff's assumption [64] is almost universally adopted by the cryptological community. According to this precept, the security of the cipher must reside entirely in the key or, equivalently, the entire cryptosystem except for the value of the key is known to the cryptanalyst.

Cryptographic systems provide secrecy by means of transformations. Depending on the type of transformation and on the type of key, the cryptosystems are commonly classified into symmetric and asymmetric cryptographic systems.

In symmetric cryptography (also called secret key cryptography), there is only a single piece of private and necessarily secret information the so-called key. Such a secret key is known to and used by the sender to encrypt the original message into a ciphertext as well as such a secret key is also known to and used by the legitimate receiver to decrypt the ciphertext into the original message. It is assumed that this double operation of encryption/decryption is impossible to be carried out without the knowledge of the secret key. Thus, in symmetric cryptography the key is shared by both legitimate communicating parties. As a result, any two users who want to communicate secretly must have previously exchanged the key in a safe way, e.g., using a trusted courier. All cryptography from ancient times until 1976

S. Díaz Cardell, A. Fúster-Sabater, *Cryptography with Shrinking Generators*,
SpringerBriefs in Mathematics, https://doi.org/10.1007/978-3-030-12850-0_1

was exclusively based on symmetric methods. Nowadays symmetric cryptography is still in widespread use, particularly for data encryption and integrity check of messages.

In asymmetric cryptography (also called public key cryptography), there are two pieces of information where at least one of which is computationally infeasible to recover from the knowledge of the other. One of the pieces is the *encryption key* (public piece of information) used by the sender to encrypt the information to be secured. The other one is the *decryption key* (secret piece of information) used by the receiver to decrypt the received ciphertext. Thus, in asymmetric cryptography each legitimate communicating party has a double key: a secret key non-shared with anyone and a public key that is known to everyone simply looking up in a public directory. In 1976, public key cryptography arose as an entirely different concept in the field of cryptography. It was first introduced by W. Diffie and M. Hellman in their mythic paper "New directions in cryptography" [21]. Asymmetric ciphers are currently used in digital signatures and key establishment as well as for classical data encryption.

Conceptually speaking, asymmetric methods seem to be more adequate for cryptographic purposes as they avoid the crucial problem of key distribution. Nevertheless, due to the nature of its operations public key algorithms are much slower than secret key algorithms. In practice, an hybrid solution is required: the key exchange is performed by public key methods and then the encryption/decryption procedure is performed by secret key methods.

Traditionally, symmetric cryptography has been split into stream ciphers and block ciphers, which can be easily distinguished.

Stream ciphers encrypt bits individually. This operation is performed by adding a bit from a pseudorandom sequence (keystream sequence) to a plaintext bit. Thus, the generation of the ciphertext is reduced to an addition of bits. Stream ciphers are synchronous when the keystream sequence depends only on the secret key and are asynchronous when the keystream sequence also depends on the ciphertext. Most practical stream ciphers are synchronous as the totality of stream ciphers considered in this book are. As example of asynchronous cipher, the cipher feedback (CFB) mode can be referenced [78, Chapter 5].

Block ciphers encrypt an entire block of plaintext bits at a time by using the same secret key. Thus, the encryption of any plaintext bit inside a given block depends on every other plaintext bit in the same block. In practice, the majority of block ciphers have a block length of 128 bits such as the Advanced Encryption Standard (AES) [19]. Nevertheless, important block ciphers with a block length of 64 bits, e.g., the Data Encryption Standard (DES) [77] or the triple DES (3DES) [78, Chapter 3], can also be referenced, although they are not recommended any more for practical applications.

In addition, different designs of sponge-based constructions [2] complete the previous categorization. Indeed, a sponge function is a generalization of both hash functions, which have a fixed output length, and stream ciphers, which have a fixed input length.

Nowadays stream ciphers are the fastest and simplest among the encryption procedures so they are implemented in many technical applications, e.g., cell phones, Internet traffic or embedded devices with little computational resources. In the following sections of this chapter, main characteristics and generalities of stream ciphers will be revised. In addition, a brief description of the most important families of stream ciphers that can be found in the literature will also be provided.

1.1 Stream Cipher

The basic problem in stream cipher design is to generate from a short and truly random key a long pseudorandom bit sequence called the keystream sequence. For encryption, the sender performs the bitwise XOR (exclusive-OR) operation among the bits of the original message or plaintext and the keystream sequence. The result is the ciphertext to be sent to the receiver. For decryption, the receiver generates the same keystream sequence, performs the same bitwise XOR operation between the received ciphertext and the keystream sequence and recovers the original message. Notice that both encryption and decryption procedures use the same operation what simplifies considerably the software/hardware implementation of this type of cipher. Moreover, such an operation is nothing but the mod 2 addition or XOR logic operation, an extremely simple and balanced operation. At any rate, the security of a stream cipher depends on the nature of the keystream sequence employed.

The precursor of the modern stream cipher is the one-time pad (OTP) or Vernam cipher invented by Gilbert Vernam in 1917. According to [52] and [78, Chapter 2], Vernam built an electromechanical machine for teletypewriter communications. The plaintext was fed into the machine as one punched paper tape and the keystream sequence as the second tape of the same characteristics. This was the first time in which encryption and transmission was automated in one machine. The main features of the OTP are:

1. The keystream sequence is only known to the legitimate communicating parties.
2. The keystream sequence is generated by a true random number generator.
3. The keystream sequence needs to be as long as the plaintext.
4. Every keystream sequence is used only once.

Under the previous conditions, the OTP is unconditionally secure or, equivalently, exhibits a mathematically proven security. Condition 1 is an habitual requirement for symmetric cryptography. Concerning conditions 2, 3 and 4, the implications are much more severe. In fact, conditions 2 and 3 mean that the keystream sequence must be generated from a physical process with length at least equal to the length of the original message, then duplicated and sent to sender and receiver through a secure channel. Moreover, we need one bit of key for each bit of plaintext. Condition 4 means that the process of generation and delivery of the sequence must be repeated every time that a secure communication is required. Clearly, the OTP is an impractical cryptographic procedure for a massive use in e-mail encryption,

mobile phones, smart cards, web browsers or similar daily applications even though it is unconditionally secure.

In practice, stream cipher substitutes the truly random keystream sequence for a pseudorandom keystream sequence generated from a short random key, e.g., no more than 128 bits, and a deterministic algorithm (the keystream generator) publicly known. Once sender and receiver have exchanged the random key in a safe way and generated the same keystream sequence, then the encryption/decryption procedure is performed as described in the Vernam cipher. Due to the substitution of a truly random keystream sequence (Vernam cipher) for a pseudorandom keystream sequence (stream cipher), the latter cipher procedure does not exhibit unconditional security. In practice, the best we can do is to design keystream generators assumed to be computationally secure. In terms of symmetric cryptography, it means that there is no cryptanalytic attack with a better complexity than an exhaustive search. In brief, stream cipher is just an approximation to OTP; the more the keystream sequence looks like a truly random sequence, the more secure the stream cipher will be.

Due to its conceptual simplicity, stream cipher is the fastest among the present cryptosystems so it is easy to find many of its technological applications everywhere, e.g., the algorithms A5 in GSM communications (see Sect. 1.2.6), the encryption system E0 in Bluetooth network specifications [24], the algorithm RC4 used in Microsoft Word processor and Microsoft Excel spreadsheet [80] or the SNOW 3G Generator [49] for wireless communication of high-speed data with 4G/LTE (long-term evolution) technology.

Finally, it must be stressed that stream cipher is mainly the cipher system for military and diplomatic purposes, for which this type of symmetric cryptography is well suited. This is the reason why many important designs and practical applications of stream ciphers are and will be condemned to the most absolute obscurantism.

1.1.1 A Basic Structure in Stream Cipher: The Linear Feedback Shift Register (LFSR)

In this subsection, we provide some basic notation and concepts that will be used throughout the book.

Let p be a prime, m a positive integer and $q = p^m$. Let $\mathbb{F}_q$ denote a finite field with q elements. The order of an element $\alpha \in \mathbb{F}_q$, denoted by $\mathrm{ord}(\alpha)$, is the smallest positive integer k such that $\alpha^k = 1$. An element α with order $q - 1$ is called a primitive element in $\mathbb{F}_q$. The primitive elements are exactly the generators of $\mathbb{F}_q^*$, the multiplicative group consisting of the nonzero elements of $\mathbb{F}_q$. Thus, a finite field $\mathbb{F}_q$ consists of 0 and appropriate powers of a primitive element.

Let $\{a_i\}$, $i = 0, 1, 2, \ldots$, be a sequence over $\mathbb{F}_p$ if $a_i \in \mathbb{F}_p$, for all $i \geq 0$. The sequence $\{a_i\}$ is periodic if and only if there exists an integer $T > 0$ such that $a_{i+T} = a_i$ holds for all $i \geq 0$.

Let L be a positive integer, and let $c_0, c_1, \ldots, c_{L-1}$ be given elements of the finite field $\mathbb{F}_p$. A sequence $\{a_i\}$ of elements of $\mathbb{F}_p$ satisfying the relation

$$a_{i+L} = c_1 a_{i+L-1} + c_2 a_{i+L-2} + \ldots + c_{L-1} a_{i+1} + c_L a_i, \qquad i \geq 0, \tag{1.1}$$

is called an Lth order linear recurring sequence in $\mathbb{F}_p$. The terms $a_0, a_1, \ldots, a_{L-1}$, which determine uniquely the rest of the sequence, are referred to as the initial values. A relation of the form given in (1.1) is called an Lth order homogeneous linear recurrence relationship. The monic polynomial of degree L

$$p(x) = x^L + c_1 x^{L-1} + c_2 x^{L-2} + \ldots + c_{L-1} x + c_L \in \mathbb{F}_p[x] \tag{1.2}$$

is called the characteristic polynomial of the linear recurring sequence and the sequence $\{a_i\}$ is said to be generated by $p(x)$. The polynomial of the lowest degree in the set of characteristic polynomials of $\{a_i\}$ over $\mathbb{F}_p$ is called the minimal polynomial of $\{a_i\}$ over $\mathbb{F}_p$. For a survey of linear recurring sequences over finite fields, the interested reader is referred to [59].

In this book, we will consider sequences defined exclusively over the binary field $\mathbb{F}_2$, i.e., $p = 2$ and $q = 2^m$, while the extension field will be denoted by $\mathbb{F}_{2^m}$. It should be noticed that the analysis provided here can be extended to sequences over any prime extension $\mathbb{F}_{p^m}$.

The generation of linear recurring sequences can be implemented on linear feedback shift registers (LFSRs). These devices handle information in the form of elements of $\mathbb{F}_2$ and they are based on shifts and linear feedback. A conventional or Fibonacci LFSR consists of L *interconnected stages* numbered $0, 1, \cdots, L-1$ (from left to right) capable of storing one bit, the *feedback* or *connection polynomial*[1] and the *initial state* (stage contents at the initial instant). In addition, a clock controls the movement (shifts) of data. During each unit of time, the following operations are performed (see Fig. 1.1):

1. The content of stage 0 is output and forms part of the output sequence.
2. The content of stage n is moved to stage $n-1$ for each n $(1 \leq n \leq L-1)$.
3. The new content of stage numbered $L-1$ is the feedback bit calculated by adding mod 2 the previous contents of a fixed subset of stages (taps) determined by the feedback polynomial.

In terms of practical implementation, the Galois LFSRs appear as alternative structures that generate exactly the same linear recurring sequences as those of Fibonacci LFSRs. More precisely, in Galois LFSRs the taps are not concatenated so they can be updated in parallel, increasing the speed of execution.

For a minimal polynomial $p(x)$ as this one defined in Eq. (1.2), the output of the LFSR with nonzero initial state is the string of elements $\{a_0, a_1, a_2, \ a_3, \ldots\}$

[1]The feedback polynomial of the LFSR and the minimal polynomial of its linear recurrence relationship are reciprocal polynomials.

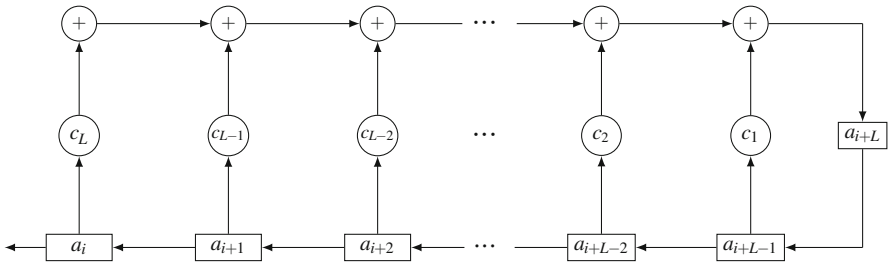

Fig. 1.1 LFSR of length L

generated in intervals of one-time unit (see Fig. 1.1). If the minimal polynomial of the linear recurring sequence is primitive [7], then the LFSR is called maximal-length LFSR and its output sequence has period $2^L - 1$, see [41]. This output sequence is called **PN-sequence** (pseudonoise sequence) or ***m*-sequence** (maximal sequence). In the sequel, all LFSRs considered will be maximal-length LFSRs. In the cryptographic literature, the LFSR minimal polynomial is simply termed as characteristic polynomial.

Linear Feedback Shift Registers are used in many of the keystream generators that have been proposed in the literature. The main reasons for such a continuous use can be enumerated as follows:

1. LFSRs provide high performance when used as sequence generators.
2. They are particularly well-suited to hardware implementations.
3. They generate output sequences with large period and good statistical properties. In fact, such sequences satisfy Golomb's pseudorandomness postulates [41].
4. Due to their simple structure, LFSRs can be readily analysed by means of algebraic techniques.

According to Golomb's pseudorandomness postulates [41], the PN-sequences are balanced (the difference between the number of ones and zeros in one period of the sequence does not exceed one), the number of binary runs (consecutive ones or consecutive zeros) occurs with the right probability (half of runs have length one, one-fourth length two, one-eighth length three, etc., as long as for each of these lengths the number of one-runs equals the number of zero-runs) and their autocorrelation function is two-valued.

At first glance, sequences obtained from maximal-length LFSRs might look like good candidates to keystream sequences. Nevertheless, as explained later, they do not satisfy a fundamental condition required to all cryptographic sequence and related with the linear character of these registers.

The linear complexity (LC) of a sequence $\{a_i\}$ is defined as the length of the shortest LFSR that can generate such a sequence or, equivalently, the order of the shortest linear recurrence relationship satisfied by such a sequence. In a general sense, linear complexity is related with the amount of sequence that

is needed to determine the whole sequence. The Berlekamp–Massey algorithm efficiently computes the length and characteristic polynomial of the shortest LFSR given at least $2 \cdot LC$ sequence bits, see [63]. Indeed, the running time of the Berlekamp–Massey algorithm is $O(N^2)$, where N is the length of the sequence under consideration.

Linear complexity is a much used metric of the security of a keystream sequence. In cryptographic terms, linear complexity must be as large as possible. The recommended value is approximately half the sequence period, $LC \simeq T/2$. According to the own definition of linear complexity, sequences generated from maximal-length LFSRs of length L will have a LC of value equal to L, what is too far from the recommended value of $T/2 \simeq 2^{L-1}$. Consequently, LFSRs should never be used alone as keystream generators. Indeed, the linear complexity of their output sequences has to be increased before such sequences are employed for cryptographic purposes.

1.2 LFSR-Based Sequence Generators

In order to overcome the low LC inherent to the sequences generated by LFSRs, in the literature several approaches are proposed. In the sequel, different methods of designing keystream sequence generators will be briefly described. All of them pursue the same goals:

- To preserve the good statistical properties of the PN-sequences.
- To increase the LC of the sequences generated by LFSRs.

Besides LC, other properties must be taken into account when keystream sequences are considered.

In fact, balancedness is one of the good statistical properties that every keystream sequence must satisfy. Roughly speaking, a binary sequence is balanced if it has approximately the same number of ones as zeros. Due to the long period of a keystream sequence ($T \simeq 10^{38}$ bits in current cryptographic applications), it is not feasible to produce an entire cycle of such a sequence and then count the number of ones and zeros. Therefore, in practice, portions of the keystream sequence are chosen randomly and the frequency test (monobit test) [70, Chapter 5] is applied to all these subsequences. If all of them pass the statistical test, then the sequence is accepted as being balanced. Nevertheless, passing the frequency test merely provides probabilistic evidence that the generator produces a balanced sequence. In the literature, balancedness of keystream sequences has been treated in a deterministic way [31, 32, 45]. Indeed, there are simple binary models based on the sequence generator parameters that allow one to compute the exact number of ones in the keystream sequence without producing the whole sequence. The same can be applied to the computation of the number of runs of any length in a keystream sequence [25].

In brief, long period, balancedness, good run distribution and large linear complexity are some necessary (never sufficient) conditions for a keystream sequence to be considered secure [32]. In addition, such sequences have to pass a battery of tests (NIST tests [76], DIEHARD tests [61] and Tuftests [62]) to be accepted as cryptographic sequences. Traditionally, the key of these stream cipher cryptosystems is the initial contents of the LFSRs included in the design. Next, a quick overview of the main families of LFSR-based sequence generators is introduced.

1.2.1 Non-linear Combination Generators

A classical technique for destroying the linearity inherent to LFSRs is to use N LFSRs working synchronously. The keystream sequence $\{s_j\}$ is produced as the image of a non-linear Boolean function f whose N variables at time t are the corresponding output bits of the N registers [59]. The function f is expressed in algebraic normal form (ANF) as the mod 2 addition (XOR logic operation) of distinct nth order products in its N variables with $0 \leq n \leq N$. The non-linear order of f is the maximum order of the terms appearing in its ANF. This construction is illustrated in Fig. 1.2, where $s(t) = s_t$ is the tth term of the keystream sequence. These keystream sequence generators are called non-linear combination generators (or non-linear combiners) and f is the combining function.

The security of those generators is conditioned by the properties of such a function. In general, the non-linear combination generators provide sequences with large period, good statistical properties and moderate linear complexity. Depending on the combining function choice, these generators can be vulnerable to certain cryptanalytic attacks (e.g., correlation attacks).

As a representative example of this type of generator, we can analyse the well-known Geffe generator [70, Chapter 6], see Fig. 1.3.

This generator is made up of three maximal-length LFSRs of lengths L_1, L_2, L_3, which are pairwise relatively prime. The combining function is

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3,$$

where the symbol $\oplus$ means the XOR logic operation. $LFSR_2$ acts as selector switching the output between $LFSR_1$ and $LFSR_3$. The keystream sequence $\{s_j\}$ obtained from the Geffe generator has period $T = (2^{L_1} - 1)(2^{L_2} - 1)(2^{L_3} - 1)$ and

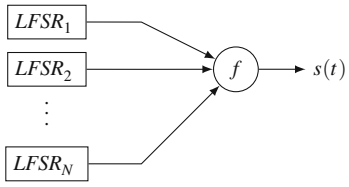

Fig. 1.2 Non-linear combiner

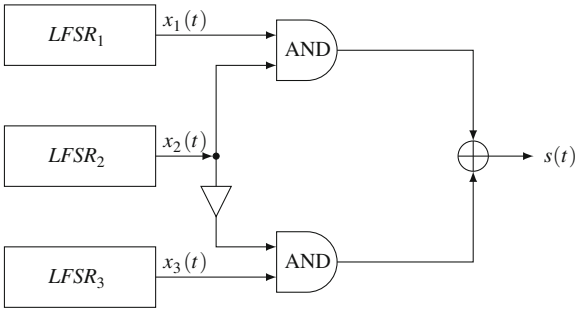

Fig. 1.3 Geffe generator

Table 1.1 Truth table for the Geffe generator

$x_1(t)$	$x_2(t)$	$x_3(t)$	$s(t)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

linear complexity $LC = L_1L_2 + L_2L_3 + L_3$. Recall that in this type of combination generators and under a variety of conditions [89] the linear complexity of the output sequence satisfies $LC = f(L_1, L_2, L_3)$. Thus, the LC of the output sequence is closely related to the order of the combining function.

Concerning balancedness, we can see in Table 1.1 that the combining function f is balanced as well as the three PN-sequences generated by the LFSRs are. In [31], a general expression in terms of L_i $(i = 1, 2, 3)$ provides the exact number of ones in the output sequence of a Geffe generator. Such an expression is

$$No(1's) = 2^{L_1-1}2^{L_2-1}(2^{L_3} - 1) + (2^{L_1} - 1)(2^{L_2} - 1)2^{L_3-1}.$$

For lengths of the LFSRs in a cryptographic range $L_i \simeq 60$, the number of ones in the output sequence is $No(1's) \simeq T/2$. Consequently, the generated sequence can be considered as a quasi-balanced sequence.

The Geffe generator is cryptographically weak because information about the successive bits from $LFSR_1$ and $LFSR_3$ leaks into the output sequence. In fact, let $x_1(t), x_2(t), x_3(t)$ be the tth output bit of $LFSR_i$ $(i = 1, 2, 3)$ and $s(t) = s_t$ the tth output bit of the keystream sequence, respectively. According to Table 1.1, the correlation probability between $x_1(t)$ and $s(t)$ and between $x_3(t)$ and $s(t)$ is

$$P(s(t) = x_1(t)) = P(s(t) = x_3(t)) = \frac{3}{4},$$

although the correlation probability between $x_2(t)$ and $s(t)$ is given by $P(s(t) = x_2(t)) = \frac{1}{2}$. Consequently, the Geffe generator is vulnerable to a simple correlation attack as it is shown in [70, Chapter 6].

In general, the combining function $f(x_1, x_2, \cdots, x_N)$ must be carefully selected in order to avoid a statistical dependence between any subset of the N PN-sequences and the keystream sequence. This condition can be guaranteed if f is chosen to be mth order correlation immune [70, Chapter 6], m being an integer $m < N$. The non-linear order of a Boolean function and the correlation immunity are properties closely related in the sense that if $f(x_1, x_2, \cdots, x_N)$ is chosen to be mth order correlation immune, then its non-linear order is at most $N - m$.

Different principles of design for good non-linear combination generators based on binary LFSR structures can be recommended:

1. Use maximal-length LFSRs to get long period and good short-term statistics in the output sequence.
2. Choose the LFSR lengths $L_1, L_2, \cdots, L_N$ to be relatively prime, i.e., $\gcd(L_i, L_j) = 1$ for $i \neq j$, to get long period.
3. Apply the practical design of balanced sequence combination generators given in [31] to get a balanced or quasi-balanced output sequence.
4. Choose the non-linear order of f to obtain a good compromise between linear complexity and correlation immunity.
5. Choose the non-linear function f to have terms of each order to get good confusion.
6. Let the key determine some terms of the function f.

The Geffe generator is an example of memoryless combination generator. Nevertheless, with the use of memory the combining function f becomes a non-linear finite state machine (FSM) which greatly increases the number of options available for these structures [88, Chapter 9]. In this case, the memoryless combining function is responsible for the level of correlation immunity and the balanced distribution of the output, whereas the next-state function is responsible for the level of non-linearity. The summation generator is a good example of memory combination generator where memory is included in the carry bit [88, Chapter 9]. Moreover, integer addition is a cryptographically useful function as it is extremely non-linear when viewed over the binary field $\mathbb{F}_2$.

1.2.2 Non-linear Filters

Another general technique for destroying the linearity inherent to LFSRs is to use a non-linear filter. In this case, the keystream sequence $\{s_j\}$ is generated as the image of a non-linear Boolean function f in the L stages of a unique LFSR, that is, the L variables of the Boolean function are the binary contents of the LFSR stages at each time instant t. This construction is illustrated in Fig. 1.4. These keystream sequence generators are called non-linear filters and f is the filtering function. Period and

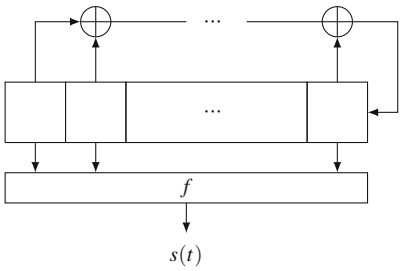

Fig. 1.4 Non-linear filter

statistical properties of the filtered sequences are characteristics deeply studied in the literature, see references [70, 78, 88].

Concerning the linear complexity, it can be stated that if the non-linear order of the Boolean function is k, then the linear complexity of the filtered sequence is at most

$$LC_{max} = \sum_{i=1}^{k} \binom{L}{i}.$$

Nevertheless, the problem of determining the exact value of the linear complexity attained by filtering functions is still an open problem [27, 55, 60]. At any rate, several contributions to the linear complexity of non-linearly filtered sequences can be quoted:

1. In [88, Chapter 5], Rueppel proves that the output sequence from non-linear filters including a unique term of equidistant stages has a linear complexity lower bounded by $LC \geq \binom{L}{k}$, where L is the LFSR length and $k \approx L/2$ the order of the filtering function. For (L, k) in a cryptographic range, e.g., $(128, 64)$, the lower bound is quite large.
2. Later, in [79] the equivalence between the root presence test [88, Chapter 5] and the discrete Fourier transform approach is established, which allows the author to give lower bounds on the linear complexity for new classes of filtering functions.
3. In [56], the authors provide an improved lower bound $LC \geq \binom{L}{k} + \binom{L}{k-1}$ on the linear complexity of filtered sequences. In any case, this lower bound is only applicable to non-linear filters of order $k \in [2, 3, L-1, L]$, which is outside the standard cryptographic range.
4. Finally, in [26] a method of computing all the non-linear filters applied to an LFSR with $LC \geq \binom{L}{k}$ is developed. The procedure is based on the concept of equivalence classes of non-linear filters and is performed by means of additions and shiftings of filtering functions coming out from different classes. The method formally completes the family of non-linear filters found in the literature with a large guaranteed linear complexity.

Concerning balancedness, it can be proved that if the filtering function f is a balanced function, then the filtered sequence will have the same period as that of the underlying LFSR [91, Theorem 1]. In addition, a binary model to compute the exact number of ones in the output sequence of a non-linear filter can be found in [32, subsection 3.2]. The computational method analyses the form of the Boolean function f and is based exclusively on the handling of binary strings by means of logic operations. The proposed model serves as a deterministic alternative to existing probabilistic methods for checking balancedness in this type of sequence generators.

Different principles of design for a good non-linear filter based on a binary LFSR structure can be recommended:

1. Use a maximal-length LFSR to get long period and good short-term statistics in the output sequence.
2. Choose a non-linear order k in the filtering function f to get large linear complexity, e.g., $k \approx L/2$, where L is the LFSR length.
3. Include a linear term and several terms of each small order in f to get good short-term statistics.
4. Apply the computational method given in [32] to check balancedness in the output sequence.
5. Include some terms of every order up to k in f to get good confusion.
6. Let the key determine some terms of the function f.

As a representative example of this type of generator, we can describe the Hitag2 generator. Hitag2 is an encryption algorithm designed by NXP Semiconductors that is used in electronic vehicle immobilizers and anti-theft devices [98]. Hitag2 uses a proprietary stream cipher with a key of 48 bits. Such a generator is a non-linear filter made up of a 48-stage LFSR and a filtering function. The feedback polynomial includes the binary contents of 16 stages in the feedback loop. The filtering function consists of three different functions f_a, f_b and f_c, see Fig. 1.5. In fact, f_a and f_b take as their four input variables the contents of different LFSR stages, while f_c takes as its five input variables the output bits of the functions f_a and f_b. Next,

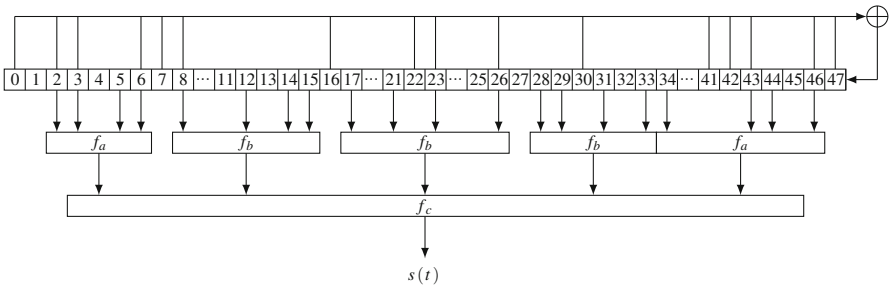

Fig. 1.5 Hitag 2

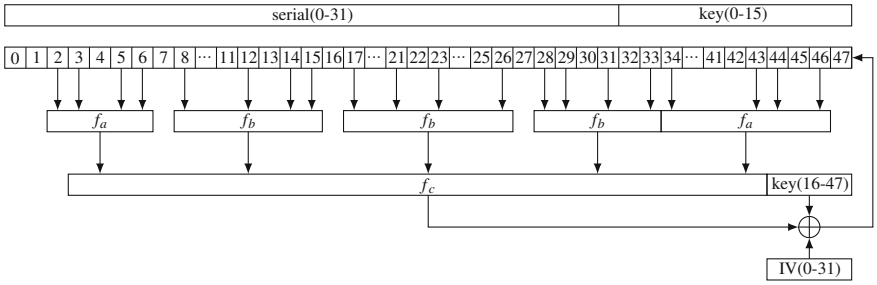

Fig. 1.6 Hitag 2 initialization

the f_c output variable is the corresponding bit $s(t)$ of the keystream generator. The previous functions are defined as follows:

$$f_a(i) = (0x2C79)_i,$$
$$f_b(i) = (0x6671)_i,$$
$$f_c(i) = (0x7907287B)_i,$$

where the output of these functions for the input i is the ith bit of the above hexadecimal values.

Previously to the keystream generation, Hitag2 needs an initialization phase to fill the 48 stages of the LFSR. The initialization procedure is described as follows. In addition to the 48-bit key, this sequence generator uses a 32-bit serial number and a 32-bit initialization vector (IV). In fact, the LFSR is filled with the 32 bits of the serial number and the first 16 bits of the key, see Fig. 1.6. Next, the cipher works in an autonomous mode for 32 cycles where the LFSR feedback bit is the result of the mod 2 addition among the corresponding key bit (16–47), the corresponding IV bit (0–31) and the Hitag2 output bit. Once the 32 cycles have been performed, the LFSR stage contents are the LFSR initial state for the register to start generating the keystream sequence.

Due to its short key, Hitag2 is considered an insecure stream cipher. Different algebraic attacks have been proposed in the literature, e.g., algebraic attacks [18, 93], attacks with a specific hardware [92, 99] or an exhaustive search attack with low cost technology [34]. Due to cost reasons, the automotive industry is surprisingly reluctant to migrate to other more secure products with a longer key.

1.2.3 Clock-Controlled Generators

In Sect. 1.2.1, the N LFSRs of a non-linear combination generator were clocked regularly, that is, the shift of data in all the registers was controlled by the same clock. Nevertheless, the main idea behind a clock-controlled generator is that the

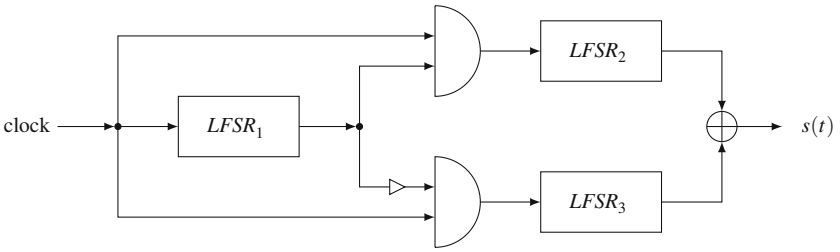

Fig. 1.7 Alternating-step generator

output sequence of one LFSR controls the clock of at least other LFSR. This irregular clocking of any LFSR is a simple strategy that introduces non-linearity into the output sequence.

As the most representative example of this type of generator, the alternating-step generator [43] is described in Fig. 1.7. The alternating-step generator uses three maximal-length LFSRs, notated $LFSR_i$ $(i = 1, 2, 3)$, of lengths L_1, L_2, L_3, which are pairwise relatively prime. In order to generate the output sequence $\{s_j\}$ the following steps are repeated:

1. The register $LFSR_1$ is clocked.
2. If the output bit of $LFSR_1$ equals 1, then $LFSR_2$ is clocked while $LFSR_3$ is not clocked but repeats its previous output bit.
3. If the output bit of $LFSR_1$ equals 0, then $LFSR_3$ is clocked while $LFSR_2$ is not clocked but repeats its previous output bit.
4. The tth bit of the keystream sequence $s(t)$ is the mod 2 addition between the output bits of $LFSR_2$ and $LFSR_3$ at the time instant t.

For the first clock cycle, the previous output bit of registers $LFSR_2$ and $LFSR_3$ is taken to be 0. The alternating-step generator is based on the stop-and-go generator of Beth and Piper [3] where only one of the LFSR was irregularly clocked.

The keystream sequence $\{s_j\}$ obtained from the alternating-step generator has period $T = 2^{L_1}(2^{L_2} - 1)(2^{L_3} - 1)$ and its linear complexity LC satisfies the inequality

$$(L_1 + L_3)\, 2^{L_1 - 1} < LC \leq (L_2 + L_3)\, 2^{L_1}.$$

The distribution of patterns in the output sequence is almost uniform. In fact, if $\mathbf{S}_s$ denotes a pattern of any s consecutive bits, then the probability P that $\mathbf{S}_s$ appears in the output sequence is given by $P(\mathbf{S}_s) \simeq (\frac{1}{2})^s$.

Recall that in this type of clock-controlled generators the linear complexity of the output sequence is lower bounded by $2^{L_1 - 1}$, that is, LC is exponential in the length of one of the LFSRs. It means that the fact of introducing irregular clocking makes increase dramatically the value of the linear complexity.

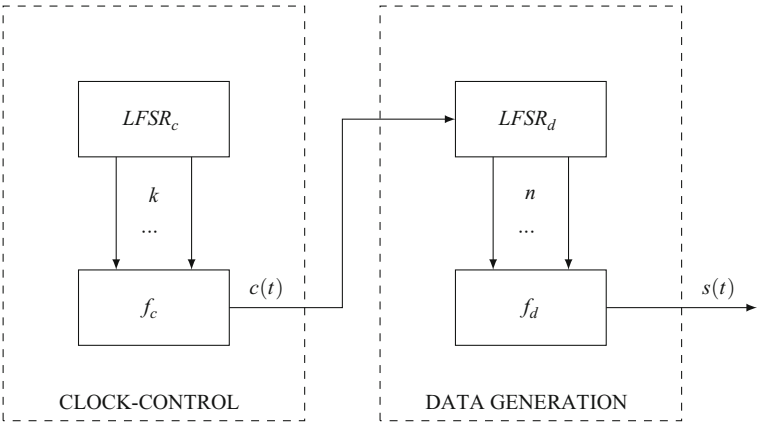

Fig. 1.8 LILI keystream generators

The security of the alternating-step generator is based on a right choice of the lengths L_i that should be about the same, that is, $L_1 \simeq L$, $L_2 \simeq L$ and $L_3 \simeq L$. In that case, the best known attack on this generator is a divide and conquer attack on the control register $LFSR_1$ [43] that takes approximately 2^L steps. Thus, if $L = 128$, then the generator is secure against this type of attack. Certain correlation attacks against clock-controlled shift registers can also be found in [35, 39] with approximately the same computational complexity as that one of the attack previously mentioned.

Among other interesting clock-controlled keystream generators, we can refer:

1. The Gollmann cascade generator [70, Chapter 6] made up of a succession of m maximal-length LFSRs of the same length L. The clock of the $LFSR_i$ is controlled by all the previous $LFSR_j$ with $j < i$. The output sequence exhibits large period $T = (2^L - 1)^m$ and excellent $LC \geq L\ (2^L - 1)^{m-1}$.
2. The LILI family of keystream generators [91] that can be viewed as a clock-controlled non-linear filter, see Fig. 1.8. The clock-control block ($LFSR_c$ + non-linear filter f_c) determines the shift of the $LFSR_d$ to whom stages a non-linear filter f_d is applied. This type of design offers large period and LC. However, some algebraic attacks can be found in the literature [16, 17]. At any rate, an attack against LILI-128 [17] can take 2^{57} CPU clocks but the requirements of intercepted bits are far from being practical.

1.2.4 Decimation-Based Generators

The underlying idea of this type of generators is the irregular decimation of a PN-sequence according to the bits of another one. The result of this decimation is

an output sequence that will be used as keystream sequence in the cryptographic procedure of encryption/decryption.

Irregularly decimated generators produce good cryptographic sequences characterized by long periods, good correlation, excellent run distribution, balancedness, simplicity of implementation, etc. Inside the family of irregularly decimated generators, we can enumerate: (a) the shrinking generator proposed by Coppersmith, Krawczyk and Mansour [15] that includes two LFSRs, (b) the self-shrinking generator designed by Meier and Staffelbach [67] involving only one LFSR, (c) the generalized self-shrinking generator or family of generators proposed by Hu and Xiao [46] that includes the self-shrinking generator and (d) the modified self-shrinking generator introduced by Kanso [53] that is related with the family of generalized self-shrinking generators. Indeed, the generalized self-shrinking generator can be seen as a specialization of the shrinking generator as well as a generalization of the self-shrinking generator. In fact, the output sequence of the self-shrinking generator is just an element of the family of generalized self-shrinking sequences.

This book focuses on decimation-based sequence generators with application in stream ciphers. Next chapters address systematically diverse features of these generators and their corresponding keystream sequences.

1.2.5 Dynamic LFSR Generators

In [74], Mita et al. proposed a new keystream sequence generator for cryptographic application based on LFSRs that they called "topology with dynamic linear feedback shift register" (DLFSR). In fact, such a topology consists in changing dynamically the feedback polynomial of the main LFSR included in the design. In this way, the output sequence of this type of generator $\{s_j\}$ is nothing but the concatenation of different portions of distinct PN-sequences. This new topology was first introduced in a generic way by means of one LFSR whose feedback polynomial was updated according to the stage contents of a secondary LFSR. In this proposal, the authors provided only series of experimental data from this particular implementation. Later in [82], Peinado et al. analysed and modelled different cryptographic parameters of the generated sequences, e.g., period, linear complexity, autocorrelation, run distribution, etc.

Basically, a DLFSR consists of:

1. A main $LFSR$ with n stages and N_p primitive feedback polynomials that will be successively applied according to a particular order determined by the feedback module.
2. A feedback control module including, among other structures, a secondary $LFSR$ with m stages and a unique primitive feedback polynomial. This module is going to control the feedback polynomial of the main register.

Although the method of generating output sequence is common to all DLFSRs, such generators can be classified into different categories depending on the operation mode:

1. DLFSR generators that apply the different N_p primitive feedback polynomials in the same order to generate the same number of output bits with each applied polynomial [68, 69, 83].
2. DLFSR generators that apply the N_p feedback polynomials in the same order to produce a different number of output bits with each one of the applied polynomials [84].
3. The most general case in which the DLFSR generators apply the N_p feedback polynomials in a pseudorandom order to produce with each polynomial a different number of output bits [1, 14, 54].

As illustrative example of DLFSR generators, Fig. 1.9 depicts a generic DLFSR generator belonging to the third category above mentioned. In fact, it is a generalization of the DLFSR module designed in [84]. The proposal represented in Fig. 1.9 is made up of two LFSRs (main and secondary registers) with n and m stages, respectively, and a counter that counts backwards from a particular value determined by the state of the secondary LFSR. At the same time, the counter controls CLK2 the clock of the secondary LFSR. The choice of the feedback polynomial applied to the main LFSR is determined by k_1 bits of the secondary LFSR among the N_p primitive feedback polynomials previously selected. Both LFSRs are initialized with their

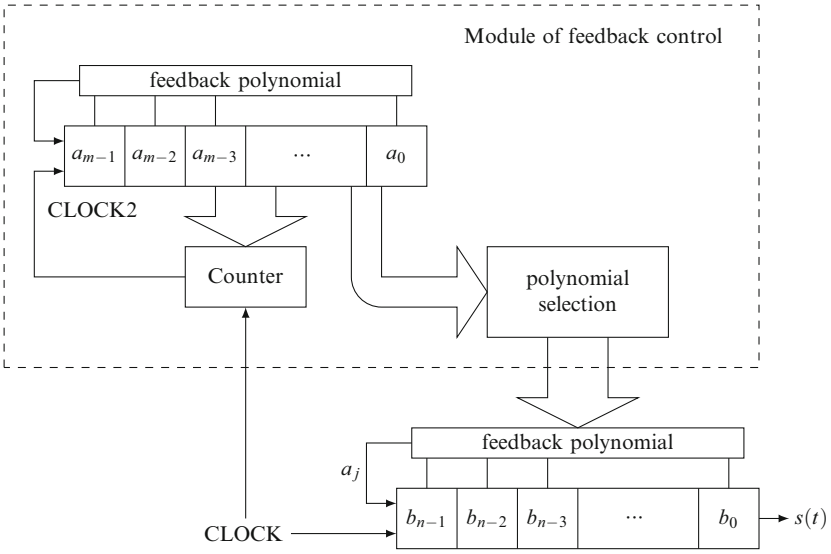

Fig. 1.9 DLFSR

corresponding initial states (key of the keystream generator). After the initialization process, the generation of the output sequence is detailed as follows:

1. The counter is initialized with k_2 bits of the secondary LFSR with $k_2 \leq \log_2 m$.
2. The main LFSR starts generating bits of the output sequence.
3. Simultaneously, the counter starts counting backwards until the value 0 is obtained. At that moment, the clock CLK2 is activated and the secondary LFSR generates a bit.
4. The new secondary LFSR state determines by means of k_1 bits the new feedback polynomial as well as by means of k_2 bits the new value of the counter.

The design here presented improves the period and linear complexity of the output sequence when compared with the same parameters obtained in DLFSR proposals [14, 74].

1.2.6 Other Types of Keystream Sequence Generators

Other types of keystream generators not included in the previous subsections can be also described. In this subsection, we consider the multiple speed Massey–Rueppel generator and the algorithms A5/1 and A5/2 used in GSM (global system for mobile communications) technology.

The Massey–Rueppel generator [65] is a keystream sequence generator employing multiple speed LFSRs. Therefore, the speed factor is treated as an additional variable in the sequence generation. The underlying idea in multiple speed generators is that, when a speed factor is introduced, a single LFSR with a fixed feedback polynomial can generate the PN-sequence corresponding to other LFSR with different feedback polynomial. Thus, multiple speed gives a new dimension to the design of secure generators.

In Fig. 1.10, an example of the simplest Massey–Rueppel generator is depicted. It consists of only two maximal-length LFSRs, notated $LFSR_i$ $(i = 1, 2)$, of lengths L_1 and L_2, respectively, which are relatively prime. The lower register $LFSR_2$ is clocked at a clock rate greater than that of the upper register $LFSR_1$. The $LFSR_2$ clock rate, notated d, is the speed factor of the generator and can be kept secret as a part of the key. The output bit $s(t)$ is the mod 2 addition of the logic products (AND operation) among the contents of the corresponding stages in both registers. The output sequence exhibits an excellent short-term statistics, no leakage and a long period of value $T = (2^{L_1} - 1)(2^{L_2} - 1)$. The weakness of this generator is its moderate linear complexity as LC is proportional to the lengths of both LFSRs. In fact, $LC = L_1 L_2$ although the scheme can be iterated to N LFSRs to get greater linear complexity of value $LC = L_1 L_2, \ldots, L_N$. The fact of changing the speed factors allows the user to generate distinct output sequences keeping unchanged the LFSRs included in the design.

Next, a different family of keystream generators is also described. The A5 stream cipher was designed to protect the over-the-air privacy of GSM telephone

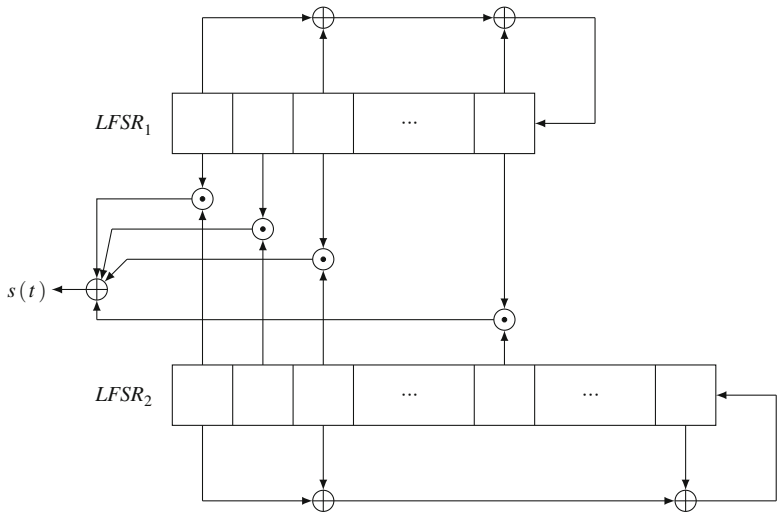

Fig. 1.10 Massey–Rueppel generator

conversations. This algorithm has two main variants: the stronger A5/1 version used by millions of customers in Europe and the weaker A5/2 version used by another millions of customers in other markets. The functional schemes of both versions will never be published. At any rate, they were reverse engineered by M. Briceno and later confirmed against official test vectors [6].

A GSM conversation is sent as a sequence of frames where every frame contains 228 bits. Each GSM conversation is encrypted by a session key K derived from algorithm A8 included in the more general algorithm COMP128, see [4]. For each frame, the key K is mixed with the corresponding frame counter (a known number of 22 bits) and the result serves as initial state of the LFSRs. From this initial state, the keystream generator first produces 100 bits that will be rejected and then the corresponding 228 keystream bits. Such bits are mod 2 added with the 228 bits of the conversation frame in order to produce the 228 bits of the ciphered conversation frame. The same process is repeated systematically for each one of the successive frames. Recall that for each conversation frame the key K is always the same only the frame counter is different.

In the following, the description of both generators A5/1 and A5/2 and their cryptanalysis are detailed.

The A5/1 generator is made up of three maximal-length LFSRs, notated $LFSR_i$ $(i = 1, 2, 3)$, of lengths $L_1 = 19$, $L_2 = 22$ and $L_3 = 23$. According to Fig. 1.11, the taps of $LFSR_1$ are at bit positions 14, 17, 18 and 19 (numbered from right to left); the taps of $LFSR_2$ are at bit positions 21 and 22; and the taps of $LFSR_3$ are at bit positions 8, 21, 22 and 23. The internal state of A5/1 at time t

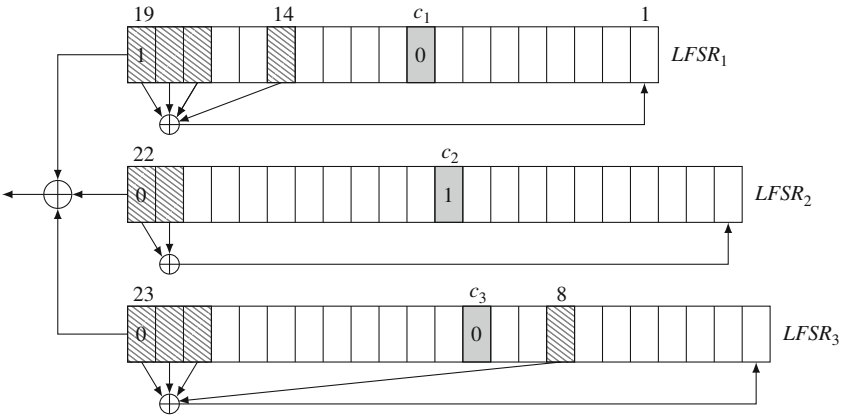

Fig. 1.11 Algoritmo A5/1

is the binary contents of the LFSRs at this particular moment. The three LFSRs are clocked in a stop/go fashion using the following majority rule:

1. Each LFSR has a single clocking stage, notated c_1, c_2 and c_3, corresponding to bit 9 in $LFSR_1$, bit 11 in $LFSR_2$ and bit 11 in $LFSR_3$.
2. At each clock cycle, the majority function F defined as

$$F(c_1, c_2, c_3) = c_1c_2 \oplus c_1c_3 \oplus c_2c_3$$

   is computed.
3. Only those LFSRs whose clocking stages agree with the majority function are actually clocked.
4. At each clock cycle, one output bit is produced as the mod 2 addition of the most significant bits in the three LFSRs.

Recall that at each clock cycle either two or three LFSRs are clocked. Moreover each LFSR moves with probability $3/4$ and stops with probability $1/4$.

Different cryptanalyses of the A5/1 generator have appeared in the literature. Particularly important is the work developed in [37], where the author describes a general time-memory trade-off attack concluding that it is possible to find the A5/1 key. This attack is based on the knowledge of a certain amount of intercepted keystream sequence and a precomputed table storing internal states and their corresponding output sequence portions. Comparing the intercepted sequence with these output prefixes, an intermediate state in some frame could be identified. Then, A5/1 runs backwards until getting the initial state of this particular frame. The key can be extracted from any frame initial state reversing the effect of the known frame counter. At any rate, for this cryptanalytic attack the requirements of intercepted keystream sequence and space in the table were unrealistic.

Nevertheless, keeping in mind all these ideas but in a more refined way, Biryukov et al. succeeded in performing an outstanding cryptanalysis [4] that revealed the insecurity of the A5/1 generator. In fact, they introduced the concept of "special states" in A5/1 or states able to produce output bits starting with a particular pattern *alpha* of length $k = 16$. The idea was scanning the intercepted sequence until such a particular pattern was encountered. Once an intermediate state had been identified, the rest of the cryptanalytic attack was the same as that one described by Golic [37]. Indeed, the fact of using just special states allowed the cryptanalysts to reduce dramatically the amount of intercepted sequence and size of the precomputed table until rather realistic levels. In brief, the easy generation of the special states, the frequent A5/1 setup routine where the same key is repeated in each frame initialization and the high probability of getting a coincidence between intercepted and stored patterns (guaranteed by the birthday paradox) are the most remarkable weaknesses of this keystream generator.

The A5/2 generator is a modified version of the previous generator including a fourth LFSR. Figure 1.12 depicts the general scheme of the A5/2 algorithm. In fact, the registers $LFSR_i$ $(i = 1, 2, 3)$ are the same as those ones used in the A5/1 generator, while the additional register $LFSR_4$ is an LFSR of length $L_4 = 17$ whose taps are at bit positions 12 and 17. Notice that $LFSR_4$ controls the shift of the remaining registers: each $LFSR_i$ is clocked when its corresponding clocking variable c_i equals 1. According to Fig. 1.12, the majority function F defined as before appears four times in different parts of the scheme. At each clock cycle, one output bit is produced as the mod 2 addition of the most significant bits in the three LFSRs plus the results of three majority functions taking values in the stages of $LFSR_i$ $(i = 1, 2, 3)$.

As most representative cryptanalysis of the A5/2 generator, the algebraic attack described in [85] can be referenced. It consists in writing out a system of equations that relates the state variables of the $LFSR_i$ $(i = 1, 2, 3)$ with the output bit by

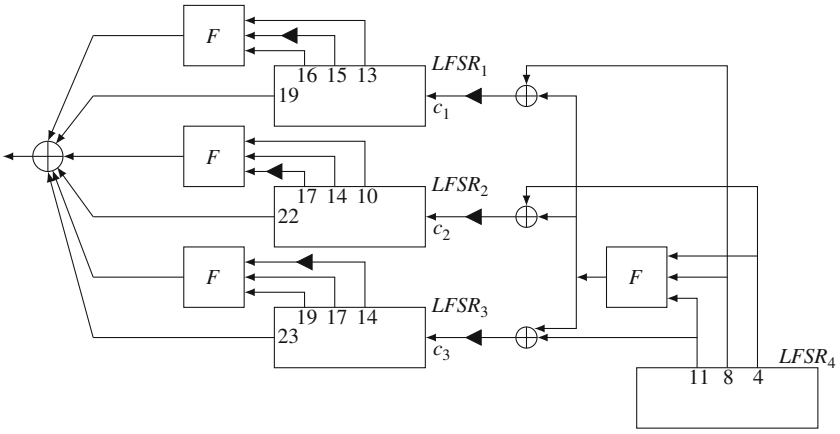

Fig. 1.12 Algorithm A5/2

means of a clock-control sequence produced by the register $LFSR_4$ starting at a particular initial state. In the worst case, all the $2^{17} - 1$ possible initial states of $LFSR_4$, paradoxically the shortest register, must be considered. Each one gives rise to a different system of equations. The linearization of these equations is performed by substitution of the non-linear terms by new and linear variables. After having written about 620 equations in each one of those linearized systems, many linearly dependent equations appear. The knowledge of four frames of keystream sequence and the linear dependences allow the cryptanalysts to reconstruct the following bits. Then, comparing reconstructed sequences with the intercepted sequence, the right keystream sequence is selected. The time complexity of this attack is proportional to 2^{17}. Frequent reinitialization of the frames, small number of skipped bits (just 100 bits rejected) in the initialization process and/or a bad distribution of taps in the LFSRs seem to be the origin of these linear dependences that this attack successfully exploits.

1.3 eSTREAM

The eSTREAM, the European stream cipher project, was a multi-year effort launched by the ECRYPT (European Network of Excellence in Cryptology) in November 2004. The goal of eSTREAM was to "promote the design of efficient and compact stream ciphers suitable for widespread adoption". After public discussions at the State of the Art of Stream Ciphers (SASC) a workshop held in Bruges (October 2004), ECRYPT published its call for stream cipher primitives and the result was the eSTREAM project [96].

The stream cipher proposals were classified into two different profiles:

1. Profile 1. Stream ciphers for software applications with high throughput requirements.
2. Profile 2. Stream ciphers for hardware applications with restricted resources such as limited storage, gate count, or power consumption.

After the call for primitives, 34 candidates were submitted to eSTREAM. Apart from a panel of experts, it was the cryptographic community at large who, after a formal evaluation in three phases, selected the most suitable stream cipher proposals for the final portfolio.

The main evaluation criteria were:

1. Security.
2. Performance when compared to AES in some appropriate mode (e.g., counter mode).
3. Performance when compared to other submissions.
4. Justification and supporting analysis.
5. Simplicity and flexibility.
6. Completeness and clarity of the submission.

Moreover, several requirements concerning the length of the key and the initialization vector (IV) as well as further technical characteristics were also defined.

The required parameter values were:

1. Profile 1.
   A key length of 128 bits must be accommodated.
   An IV length of at least one of 64 or 128 bits must be accommodated.
2. Profile 2.
   A key length of 80 bits must be accommodated.
   An IV length of at least one of 32 or 64 bits must be accommodated.

Software performance is an aspect particularly significant for Profile 1 candidates. In fact, software performance can be measured in many different ways. In order to make comparisons as fair as possible, eSTREAM developed a testing framework to assure that all stream cipher proposals were submitted to the same tests under the same circumstances. This testing framework and documentation available in [94] has been and continues to be used by other researchers outside of eSTREAM.

As a result of the project, a portfolio of eight new and promising stream ciphers was announced in April 2008, see Table 1.2. Later the eSTREAM portfolio was revised in September 2008 when M. Hell and T. Johansson [44] published a cryptanalytic attack in real time against the F-FCSR-H v2. As a consequence, the ECRYPT had to eliminate the cipher F-FCSR-H v2 from the previous list. At the present moment, the eSTREAM portfolio contains the ciphers listed in Table 1.3 with seven stream ciphers.

Descriptions, possible weaknesses and reports on software/hardware performance of not only the proposals in the portfolio but also of all eSTREAM candidates can be found in the corresponding links of the official eSTREAM website [94]. The portfolio is periodically revised as the algorithms mature. Different reviews of the eSTREAM portfolio were published in October 2009, January 2012 and another one recently, see [94]. At the same time, a volume published by Springer [87] in 2008 provides full specifications of all 16 ciphers that reached the final phase of the

Table 1.2 First eSTREAM portfolio

Profile 1 (SW)	Profile 2 (HW)
HC-128	Grain v1
Rabbit	MICKEY v2
Salsa20/12	Trivium
SOSEMANUK	F-FCSR-H v2

Table 1.3 Final eSTREAM portfolio

Profile 1 (SW)	Profile 2 (HW)
HC-128	Grain v1
Rabbit	MICKEY v2
Salsa20/12	Trivium
SOSEMANUK	

eSTREAM project. In fact, it is a very detailed survey covering both the software- and the hardware-oriented finalists. In addition, a prototype of an ASIC containing all Profile 2 candidates was designed and fabricated on 0.18 μm CMOS, as part of the eSCARGOT project [95].

Keeping in mind that the goal of eSTREAM was to stimulate works in the area of stream ciphers, undoubtedly the project has been a great success. It served to:

1. revitalize the field of stream ciphers after the widespread deployment of AES, and
2. identify two areas where a dedicated stream cipher design might offer advantages over block ciphers:
   - areas where exceptionally high throughput is required in software, and
   - areas where exceptionally low resource consumption is required in hardware.

Over the following years the eSTREAM proposals have been assessed with regard to both security and practicality by the cryptographic community, and the results presented at major conferences and specialized workshops dedicated to the state of the art of stream ciphers. Until now, the stream ciphers included in the portfolio list remain unchanged.