Chapter 2

Stream Ciphers

This chapter introduces some basic notions about stream ciphers and describes some keystream generators. Section 2.1 is devoted to the description of synchronous and self-synchronous stream ciphers, and some approaches to the construction of stream ciphers based on block ciphers. Section 2.2 introduces some keystream generators. Section 2.3 considers some cryptographic aspects of sequences, such as linear complexity, weight complexity, sphere complexity, autocorrelation and crosscorrelation functions, pattern distribution, quadratic span and maximum order complexity. Section 2.4 shows the consistency and harmony of the binary natural sequence generator which is the main topic of this book. Section 2.5 considers the security of and attacks on stream ciphers generally.

2.1 Stream Cipher Systems

Ciphering systems are generally classified into block and stream ciphers, in analogy to error-correcting codes which are subdivided into block and convolutional codes. The essential distinction between block and stream ciphers is the memory, as is shown in Figures 2.1(a) and 2.1(b).

A block cipher breaks each plaintext message into successive blocks and enciphers each block M under the control of a key k into a ciphertext block $C = (c_1, \cdots, c_n)$, where the plaintext and ciphertext alphabet are usually identical. Each block is typically several characters long. Simple substitution and homophonic substitution ciphers [103] are examples of block ciphers, even though the unit of encipherment is a single character. This is because the same key is used for each character. A stream cipher specifies a device with internal memory that enciphers the jth digit m_j of the message stream into the jth digit c_j of the ciphertext stream by means of

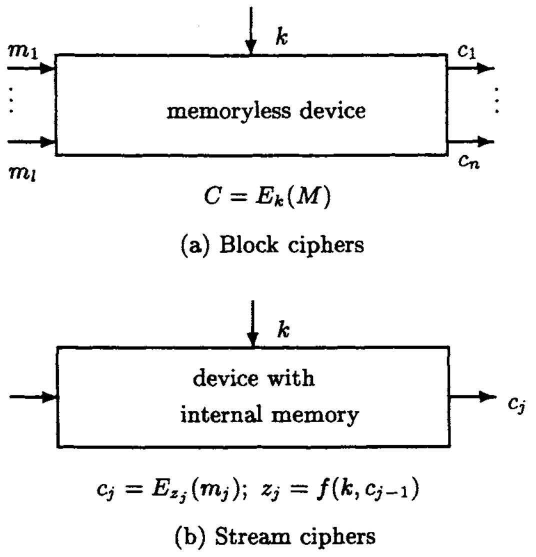

Figure 2.1: The difference between block and stream ciphers.

a function which depends on both the secret key k and the internal state of the stream cipher at time j. The sequence $z^\infty = z_0 z_1 \cdots$ which controls the enciphering is called the ***key stream*** or ***running key***. The deterministic automaton which produces the key stream from the actual key k and the internal state is called the ***running-key generator*** or ***keystream generator***.

A stream cipher is ***periodic*** if the keystream repeats after d characters for some fixed d; otherwise it is nonperiodic. Ciphers generated by Rotor and Hagelin machines [103] are periodic stream ciphers. The Vernam cipher is an example of a nonperiodic stream cipher.

There are two different approaches to stream encryption: synchronous methods and self-synchronous methods. In a ***synchronous stream cipher***, the next state depends only on the previous state and not on the input so that the succession of states is independent of the message stream. The key stream is therefore generated independently of the message stream. Consequently, the enciphering transformation is memoryless, but time-varying. Thus, if a ciphertext character is lost during transmission, the sender and receiver must resynchronize their generators before they proceed further. Furthermore, this must be done in a way which ensures that no part of the key stream is repeated (thus the keystream generator should not be reset to an earlier state). It is therefore natural, in a synchronous stream cipher,

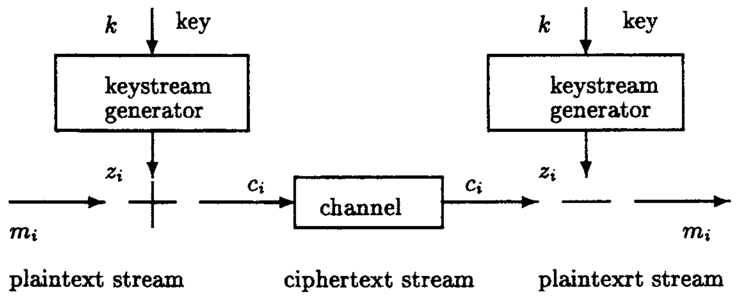

Figure 2.2: Additive synchronous stream ciphers.

to separate the enciphering transformation from the generation process of time-varying parameters which control the deciphering transformation.

In a ***self-synchronous stream cipher***, each keystream character is derived from a fixed number n of preceding cipher characters. Thus, if a ciphertext character is lost or altered during transmission, the error propagates forward for n characters, but the cipher resynchronizes itself after n correct ciphertext characters have been received. Self-synchronous stream ciphers are nonperiodic because each key character is functionally dependent on the entire preceding message stream.

Figures 2.1.a and 2.1.b depict block and stream ciphers respectively. In Figure 2.1.a M and C stand for plaintext and ciphertext block respectively, and E_k is the encryption transformation specified by a key k. In Figure 2.1.b m_j and c_j are the plaintext and ciphertext character respectively, z_j the keystream character at time j, f a function for producing the keystream, and E_{z_j} a function for combining the keystream character z_j and the plaintext character m_j. A practical difference between a block and a stream cipher is that the redundancy of a natural language may remain in the ciphertext under a block cipher, while it has been usually made very small with a well-designed stream cipher. This may explain why stream ciphers are still popular in practice.

2.1.1 Additive Synchronous Stream Ciphers

As mentioned above, in a synchronous stream cipher, the key stream, $z^\infty = z_0z_1\cdots$, is independent of the message stream. The algorithm that generates the stream must be deterministic so that the stream can be reproduced for decipherment. One important kind of synchronous stream ciphers is the ***additive synchronous stream ciphers*** depicted by Figure 2.2, where the

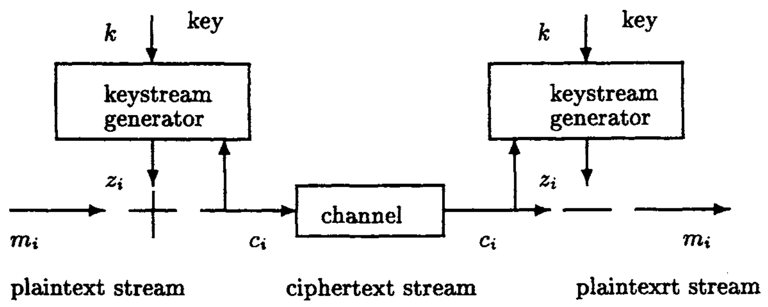

Figure 2.3: Additive self-synchronous stream ciphers.

characters of the key stream are from an Abelian group $(G, +)$ and the ciphertext character c_i is the addition of the key stream character z_i and message stream character m_i, and "−" denotes the inverse operation of "+".

The main design problem for this kind of stream cipher is the design of the keystream generator. Because the way to combine plaintext and ciphertext characters is very simple, keystream generators for additive synchronous stream ciphering should be strong enough.

2.1.2 Additive Self-Synchronous Stream Ciphers

In a self-synchronous stream cipher each keystream character is derived from a fixed number n of preceding ciphertext characters. The idea of this kind of cipher traces back to the time of Vigenère in the 16th Century. Autokey ciphers and cipher feedback systems are examples of additive self-synchronous stream ciphers [103].

An *autokey cipher* is one in which the key is derived from the message it enciphers. Another important class of self-synchronous stream ciphers consists of those where the cipher is fed back to the keystream generator as depicted in Figure 2.3. The main problems concerning this kind of stream ciphers are the design of the keystream generator and the way in which the feedback ciphertext character is used in the keystream generator. This kind of stream cipher is rather difficult to design and analyze because of the feedback approach.

2.1.3 Nonadditive Synchronous Stream Ciphers

There are advantages and disadvantages in both block ciphering and additive stream ciphering. Additive synchronous stream ciphers have the disadvantage that a ciphertext-plaintext character pair immediately reveals the corresponding keystream character under which the plaintext character is encrypted. This makes possible various kinds of key-recovering attacks such as correlation attacks and collision attacks, equivalent-machine attacks such as the one based on the Berlekamp-Massey algorithm, approximate-machine attacks such as those based on linear approximations. One of their advantages is that the keystream is time-varying, which ensures that the same plaintext character usually corresponds to different ciphertext characters at different times. This usually conceals some statistical properties of the plaintext. Block ciphers have the disadvantage that their keys cannot be changed very frequently due to the problem of key management. In addition, the same block of message corresponds always to the same ciphertext block if one key is selected and fixed. This may make many attacks such as differential attacks on some block ciphers applicable. One of their advantages is that the detection of the modification of messages may be possible owing to the fact that messages are encrypted block by block.

To keep the merits of both additive stream ciphering and block ciphering, but to get rid of the demerits of both approaches, a dynamic block ciphering approach is described as follows. With this approach a keystream generator and a conventional (one-key) block cipher are combined in such a way that some output characters of the keystream generator are employed to serve as the dynamic key of the block cipher for each message block.

For a block cipher of plaintext block length n, let $E_k(.)$ and $D_k(.)$ denote respectively the encryption and decryption transformation specified by a key k. To use the block cipher to encipher and decipher dynamically, a dynamic key k_i for the block cipher is produced by a sequence generator SG as $(z_{ti}, z_{ti+1}, \cdots, z_{ti+t-1})$, where t is a positive integer, and z^∞ denotes the sequence produced by the SG. The parameter t could be 1 or another assigned constant. Thus, the encryption and decryption are done respectively by

$$\begin{aligned} c_i &= E_{k_i}(m_i), \\ m_i &= D_{k_i}(c_i), \end{aligned}$$

where m_i is the plaintext block, c_i the ciphertext block at time i. Since the key k_i is time-varying, this is a dynamic block ciphering, and therefore a nonadditive synchronous stream ciphering approach. The key of the system consists of that of the keystream generator SG.

In this ciphering system it is not necessary to require large linear complexity for the output sequence of the SG if the underlying block cipher is properly designed. One cryptographic idea behind the design is cooperation. The SG and the block cipher should be designed so that they can protect each other. This kind of ciphering approach is intended to thwart as many attacks on block and/or additive synchronous stream ciphers as possible. Indeed, if the system is well designed, it seems that known attack approaches to additive stream ciphers and block ciphers do not apply to the system. To attack the system, one needs to develop new approaches.

Another aim of this system is to get fast ciphering algorithms. It is possible to use fast sequence generators and fast block ciphers in this system to get fast and secure ciphering algorithms.

Additive synchronous stream ciphers and all block ciphers can be regarded as special cases of the system. If the underlying block cipher of the above dynamic ciphering system is chosen as the term-wise addition of the key and the plaintext block, then the system is the usual additive synchronous stream cipher. In this case, we use one of the worst block ciphers in the system. If the SG is chosen such that the keystream is a constant sequence, then it is the usual block ciphering approach. In this case, we employ the worst keystream generator. Thus, the usual block ciphers and additive stream ciphers are special cases of the above approach, and in fact two extreme cases of the system.

2.1.4 Stream Ciphering with Block Ciphers

There are several kinds of modes of using block ciphers. The most studied four are the Electronic Codebook (ECB) mode, the Cipher Block Chaining (CBC) mode, the Cipher Feedback Chaining (CFB) mode, and the Output Feedback Chaining (OFB) mode [216].

In the ECB mode, a block cipher is applied block by block independently. Let $M = M_1M_2 \cdots M_t$ be the plaintext, then the encryption is carried out as

$$C_i = E_k(M_i) \text{ for } i = 1, 2, \cdots, t.$$

Thus, the corresponding ciphertext is $C = C_1C_2 \cdots C_t$. The decryption is then described by

$$M_i = D_k(C_i) \text{ for } i = 1, 2, \cdots, t,$$

where $D_k(x)$ is the inverse transformation of $E_k(x)$. This is a rather straightforward way to use block ciphers.

In the CBC mode the blocks are chained together with an initial value IV. In this mode we assume that the plaintext and ciphertext block space are identical, and that this block space is an Abelian group with an operation $+$. The first ciphertext block is defined as

$$C_1 = E_k(M_1 + IV),$$

where IV is an initial value from the block space. The other ciphertext blocks are then computed as follows:

$$C_i = E_k(M_i + C_{i-1}) \text{ for } i = 2, 3, \cdots, t.$$

To decrypt, the first plaintext block is obtained as

$$M_1 = D_k(C_1) - IV,$$

where "$-$" is the inverse operation of "$+$". The other plaintext blocks are then calculated as

$$M_i = D_k(C_i) - C_{i-1} \text{ for } i = 2, 3, \cdots, t.$$

Clearly, the CBC mode makes a block cipher into a stream cipher which has internal memory.

The CFB mode also uses a block cipher for stream ciphering. Assume that we have a block cipher with both plaintext and ciphertext block space A^n, where the alphabet $(A, +)$ is an Abelian group. Let $E_k(x)$ be the encryption transformation, rchop_u denote the function that drops the u rightmost characters of its argument, and lchop_u denote the function that drops the u leftmost characters of its argument. A simple variant of the CFB mode is described as follows. Choose m to be any integer between 1 and n. The stream cipher based on the block cipher has then the alphabet $(A^m, +)$, where the operation "$+$" of A^m is a natural extension of the operation of A, i.e.,

$$(x_1, \cdots, x_m) + (y_1, \cdots, y_m) = (x_1 + y_1, \cdots, x_m + y_m),$$

where $(x_1, \cdots, x_m) \in A^m$ and $(y_1, \cdots, y_m) \in A^m$. Under the choice of an initial value X_1, the encryption of the ith plaintext character $M_i \in A^m$ is carried out as

$$C_i = M_i + \text{rchop}_{n-m}(E_k(X_i)), \quad X_{i+1} = \text{lchop}_m(X_i) \mid\mid C_i,$$

where $\mid\mid$ denotes the concatenation. The decryption is as follows:

$$M_i = C_i - \text{rchop}_{n-m}(E_k(X_i)), \quad X_{i+1} = \text{lchop}_m(X_i) \mid\mid C_i.$$

An internal register is needed to update X_i.

The OFB mode uses also a block cipher for stream ciphering. As in the CFB mode, we have first a block cipher with both plaintext and ciphertext block space A^n, where the alphabet $(A, +)$ is an Abelian group. The stream cipher based on the block cipher is described as follows. The plaintext and ciphertext alphabet of the stream cipher are A^m, where m can be arbitrarily chosen between 1 and n. The stream cipher has an internal register for updating the values $X_i \in A^n$. Let X_1 be the initial value of the register. The encryption of the ith plaintext character $M_i \in A^m$ is carried out as

$$C_i = M_i + \text{rchop}_{n-m}(E_k(X_i)), \quad X_{i+1} = E_k(X_i).$$

The decryption is defined by

$$M_i = C_i - \text{rchop}_{n-m}(E_k(X_i)), \quad X_{i+1} = E_k(X_i).$$

Note that the only difference between the CFB and OFB is the updating of the internal register.

Among the above four modes of operations for block ciphers three of them result in stream ciphers. Naturally, there are many other ways to use block ciphers for stream ciphering. A nonadditive synchronous stream cipher based on block ciphers was described in the previous section. Another approach to the construction of stream ciphers based on block ciphers will be described in the following section.

2.1.5 Cooperatively Distributed Ciphering

There are advantages and disadvantages in both block and additive stream ciphering, as made clear in Section 2.1.3. To keep the advantages of both block and additive stream ciphering and to get rid of their disadvantages, a cooperatively distributed (briefly CD) ciphering system was described by Ding and Salomaa in [135].

The cooperatively distributed ciphering system consists of s components: s conventional block ciphers of the same block length, and a control device which is a sequence generator with internal memory, SG for short, which produces sequences over the alphabet $Z_s = \{0, 1, \cdots, s-1\}$.

Let $k_0, \cdots, k_{s-1}$ be the keys respectively; $E_0(k_0, \cdot), \cdots, E_{s-1}(k_{s-1}, \cdot)$ the encryption transformations specified by the keys; $D_0(k_0, \cdot), \cdots, D_{s-1}(k_{s-1}, \cdot)$ the decryption transformations specified by the keys respectively. Let k_{sg} be the key of the sequence generator, z_i be the output character of the SG at time i. The key of the CD cipher system is $k = (k_{sg}, k_0, \cdots, k_{s-1})$. At each time unit only one of the block ciphers is active, i.e., doing the encryption (respectively decryption). So we have

$$c_i = E_{z_i}(k_{z_i}, m_i),$$

where m_i and c_i are the ith plaintext block and ciphertext block. Similarly, the decryption is defined by

$$m_i = D_{z_i}(k_{z_i}, c_i).$$

In this CD cipher system the SG determines the action of each component block cipher, and it is possible for the encryption algorithms $E_0, \cdots, E_{s-1}$ to be the same, but in this case the keys $k_0, k_1, \cdots, k_{s-1}$ should be pairwise different.

The security of the system can be analyzed as follows. First we consider attacks on block ciphers. All the attacks on block ciphers are done under the assumption that the key is fixed and there is only one encryption (respectively decryption) algorithm. Among such attacks are differential attacks and linear attacks. All of those attacks could not apply in a simple way to this CD cipher system, since we have at least two different encryption (resp. decryption) algorithms or at least two different keys for the underlying block ciphers. Second, though there are a number of attacks on stream ciphers, most of them apply only to additive ones, and consequently to those keystream generators for additive stream ciphers. If the CD cipher system is designed properly, those attacks should not apply.

The CD cipher system is a stream ciphering one, though it is a combination of block and stream ciphers, since a message usually corresponds to different ciphertexts at different times. The purpose of cooperation and distribution is to make infeasible as many known attacks on both block and additive stream ciphers as possible. Given a piece of ciphertext, it is usually difficult for the enemy to know how many times a component block cipher has contributed and where it has distributed.

If the system is designed properly, it is possible to get a very strong cipher by choosing some very weak block ciphers and a weak sequence generator. This shows again the power of cooperation and distribution.

The components and the control device in the CD system should be chosen carefully. In what follows we consider the system consisting of two component block ciphers.

Let K_0 and K_1 be the key spaces of the two block ciphers respectively. Assume that each key of K_0 (resp. K_1) is equally likely. Let $p_0 = \Pr(z = 0)$, $p_1 = \Pr(z = 1)$ and

$$n_i(m,c) = |\{k_i \in K_i | E_i(k_i, m) = c\}|, \ i = 0, 1.$$

Also let $\Pr(m, c)$ denote the probability that c is a corresponding ciphertext block of the plaintext block m. Then it is not difficult to see that

$$\Pr(m,c) = p_0 \frac{n_0(m,c)}{|K_0|} + p_1 \frac{n_1(m,c)}{|K_1|},$$

$$\Pr(z = i; (m,c)) = p_i \frac{n_i(m,c)}{|K_0|}, \ i = 0, 1.$$

It follows that we have the conditional probabilities

$$\Pr(z=0|(m,c)) = \frac{|K_1|p_0n_0(m,c)}{|K_1|p_0n_0(m,c)+|K_0|p_1n_1(m,c)}$$
$$\Pr(z=1|(m,c)) = \frac{|K_0|p_1n_1(m,c)}{|K_1|p_0n_0(m,c)+|K_0|p_1n_1(m,c)}$$

Hence, we have the following expression for the average mutual information

$$\begin{aligned} I(z;(m,c)) &= -\frac{|K_1|p_0n_0(m,c)}{|K_1|p_0n_0(m,c)+|K_0|p_1n_1(m,c)} \\ &\quad \times \log\frac{|K_1|p_0n_0(m,c)}{|K_1|p_0n_0(m,c)+|K_0|p_1n_1(m,c)} \\ &\quad -\frac{|K_0|p_1n_1(m,c)}{|K_1|p_0n_0(m,c)+|K_0|p_1n_1(m,c)} \\ &\quad \times \log\frac{|K_0|p_1n_1(m,c)}{|K_1|p_0n_0(m,c)+|K_0|p_1n_1(m,c)}. \end{aligned}$$

To minimize the above average mutual information, we have to ensure that

$$p_0\frac{n_0(m,c)}{|K_0|} = p_1\frac{n_1(m,c)}{|K_1|}. \tag{2.1}$$

Note that

$$\sum_{c\in C}\frac{n_0(m,c)}{|K_0|} = \sum_{c\in C}\frac{n_1(m,c)}{|K_1|} = 1.$$

It follows that

$$p_0 = \sum_{c\in C}p_0\frac{n_0(m,c)}{|K_0|} = \sum_{c\in C}p_1\frac{n_1(m,c)}{|K_1|} = p_1.$$

Hence, $p_0 = p_1 = 1/2$, and furthermore

$$\frac{n_0(m,c)}{|K_0|} = \frac{n_1(m,c)}{|K_1|}. \tag{2.2}$$

With the above analysis, we have obtained the following design principle. For the CD cipher system with two component block ciphers, the parameters should be chosen such that

1. $p_0 \approx \frac{1}{2}$;

2. $\frac{n_0(m,c)}{|K_0|} \approx \frac{n_1(m,c)}{|K_1|}$, and if one of $n_0(m,c)$ and $n_1(m,c)$ is zero, so must be the other.

Clearly, a cipher is secure against ciphertext-only attacks if it is secure against known plaintext attacks. Given some plaintext-ciphertext block pairs, a cryptanalyst may first try to get a piece of keystream and then try to recover the key of the SG or to construct a generator which produces the same control sequence, by analyzing the parameters $n_0(m,c)$ and $n_1(m,c)$ of the two block ciphers for the given plaintext-ciphertext pairs. If the two block ciphers are not well designed, and the cryptanalyst gets to know $n_0(m,c) = 0$, then he/she knows immediately that the control digit under which a block cipher is selected is 1. If an attack on the SG is successful, then it remains only to attack the two block ciphers in the usual sense. At this stage the meaning of cooperation is lost. The above design principle is intended to make infeasible this kind of divide-and-conquer attack.

On the other hand, the SG should be designed so that its output sequences have good pattern distributions. If the control sequence is $111\cdots1000\cdots0$, then the cooperation is obviously very bad.

A CD system can be much more secure than the underlying block ciphers. If the SG is well designed, some weak block ciphers can be employed. It is important that the two block ciphers should have many similarities, just like "twins". This indicates that using only a two-key cooperation within one well-designed algorithm seems better, but in this approach one has to guarantee that the two keys do not specify the same encryption transformation; otherwise there is no cooperation within the system.

2.2 Some Keystream Generators

Finite state machines are important mathematical objects for modeling electronic hardware. Furthermore, due to their recursiveness finite state machines are convenient means for realizing infinite wordfunctions built over finite alphabets. Many keystream generators can be modeled by finite state machines [112, 343]. In a synchronous stream cipher, the running-key generator may generally be viewed as an autonomous finite state machine as depicted in Figure 2.4.

The keystream generator as a finite state machine consists of an output alphabet and a state set, together with two functions and an initial state. The *next state function* f_s maps the current state S_j into a new state S_{j+1} from the state set, and the output function f_0 maps the current state S_j into an output symbol z_j from the output alphabet. The key may determine the next state function and the output function as well as the initial state.

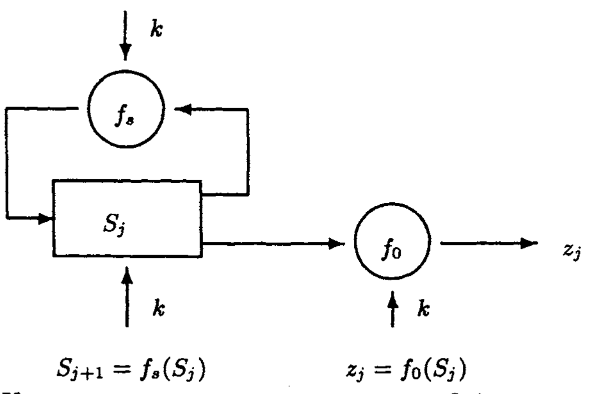

Figure 2.4: Keystream generators as autonomous finite state machines.

The fundamental problem of designing a keystream generator is to find a next state function f_s and an output function f_0 which are guaranteed to produce a running key z^∞ that satisfies certain cryptographic requirements such as large linear complexity and good linear complexity stability, good autocorrelation, uniform pattern distribution, etc. In some cases, the output function f_0 should possess the good difference property with respect to some binary operation of the state vector space and good nonlinearity with respect to the binary operation of the state vector space and that of the output alphabet space. These binary operations depend on the realization of the next state function f_s. The actual specific requirements for the next state and output function depend on the system in which the generator is used.

In order to meet certain requirements, special classes of finite state machines have been employed as running-key generators. Unfortunately, the theory of autonomous automata whose change of state function is nonlinear has not been well developed. There are many kinds of proposed keystream generators. Some are easy to implement, but their security may be difficult to control; some are secure against certain kinds of attacks, but may have a relatively slow implementation. In what follows we shall give a brief description of some number-theoretic generators and counter generators.

2.2.1 Generators Based on Counters

A counter, one of the simplest automata, has a period, which is often taken to be q^n, where q is a positive integer. A counter of period N counts the numbers $0, 1, ..., N-1$ cyclically. Diffie and Hellman suggested applying

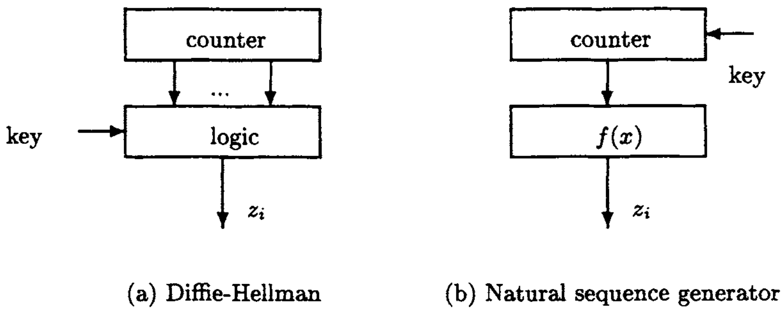

Figure 2.5: Some counter generators.

a nonlinear function to a counter to construct keystream generators [112, p.416], as depicted in Figure 2.5(a). In this kind of generator, the key is used to control the function. The initial values of the counter may be taken as part of the key or as a random value sent as an indicator. A specific proposal given by Diffie and Hellman is to use a fixed component of a block cipher algorithm as the function for the generator of Figure 2.5(a) [112].

If we consider counters of arbitrary period N and use a fixed function $f(x)$ from Z_N to an Abelian group G, we have the generator of Figure 2.5(b). In this generator, the key k is one of the integers $0, 1, ..., N-1$, and the counter begins its cyclical counting at the key value. The arguments x of $f(x)$ are the successive integer values provided by the counter. Thus the output sequence or keystream in G is given by

$$z_i = f((i+k) \bmod N),$$

where the residue modulo N is taken to be an integer between 0 and $N-1$. There are slight differences between the two generators. In the generator of Figure 2.5(a), the key or part of the key is used to control the function, while in the generator of Figure 2.5(b) the function $f(x)$ is fixed and the key is simply the initial value of the register. The generator of Figure 2.5(b) is called the ***natural sequence generator*** (briefly, NSG) because every periodic sequence can be realized by this generator in a natural way and many security aspects of the generator can be analyzed and controlled. Synchronous additive stream ciphers based on this kind of generator are called ***additive natural stream ciphers*** [122].

In [122] the differential cryptanalysis and design of the additive natural stream ciphers were studied. It was shown that an improperly designed natural keystream generator could be broken by a differential attack. Other possible attacks, such as key determining attacks based on decision trees, partial-key attacks, linear approximation attacks with respect to the additions of Z_N and the Abelian group over which the key stream is built and key (key stream) correlation attacks, were also possible for this generator if the design parameters are not chosen properly [122]. If the generator is properly designed, the NSG may resist all possible attacks mentioned above. This book is mainly concerned with the design and analysis of this generator.

2.2.2 Some Number-Theoretic Generators

"Pseudorandom" numbers are needed not only in cryptography, but also in numerical simulations for Monte Carlo methods, sampling, numerical analysis, testing computer chips for defects, decision making, and programming slot machines [256, 245]. However, different applications require different random properties of the numbers. For instance, pseudorandom numbers for simulations are different from those for cryptographic purposes in a number of senses. There are several proposed number-theoretic generators. One of these is the ***multiplicative generator***, which is described by

$$x_{n+1} = ax_n + b \bmod M,$$

where $0 \leq x_n \leq M - 1$. Here (a, b, M) are the parameters describing the generator and x_0 is the seed. More generally, one can consider polynomial recurrences (mod$N$), or vector-valued polynomial recurrences as done by Lagarias and Reeds [255]. These linear congruential generators are widely used in practice in Monte Carlo methods [245, 375, 288] but they are cryptographically weak. It has been pointed out by Lagarias [256] that for the above linear congruential generator, the parameters a and b can be recovered from three consecutive iterates x_1, x_2, x_3 if M is known. When a, b and M are not known, there is a polynomial algorithm which, given the output $(x_1, \cdots, x_n)$ of a linear congruential generator (mod$M$), will generate a prediction x'_{n+1}. This algorithm has the property that if this is done for $n = 1, 2, 3, \cdots$, it will make at most $3 + \log M$ mistakes. For details about the algorithm we refer to Lagarias [256] and Boyar [30].

If we expand the rational

$$\frac{1}{p} = .d_0 d_1 d_2 \cdots d_j d_{j+1} \cdots$$

in base d, we have the *$1/p$ generator*. Here (p, d) are parameters describing the generator, and the seed is a specified position j of the initial digit; i.e., set $x_n = d_{j+n}$. Details about the generator can be found in [26, 256], however we shall consider this generator in Chapter 14.

There is also the so-called *power generator* described by

$$x_{n+1} = x_n^d \bmod N,$$

where (d, N) are parameters describing the generator and x_0 is the seed. There are two special cases of the power generator, both occurring when $N = p_1 p_2$ is a product of two distinct odd primes. If d is chosen such that $\gcd(d, \phi(N)) = 1$, then the map $x \to x^d$ is a permutation of Z_N^*, and the generator is called the *RSA generator* by Lagarias [256]. If we choose $d = 2$ and $N = p_1 p_2$ with $p_1 = p_2 = 3 \bmod 4$, this is the *square generator.* Properties of these generators can be found in Lagarias [256] and Blum, Blum, and Shub [26]. We shall consider the Blum-Blum-Shub generator in Section 14.8.

A number-theoretic generator based on the exponential operation is the following one described by

$$x_{n+1} = g^{x_n} \bmod N,$$

where (g, N) are parameters describing the generator and x_0 is the seed. For more about this generator, one should consult [256].

The generators of this section could be quite slow, when the modulus is large. By modifying the above generators, one may obtain some number-theoretic bit generators. Among them are the RSA bit generator [3, 256], the modified Rabin bit generator [3, 359, 256], the discrete exponential generator [25, 278, 256].

2.3 Cryptographic Aspects of Sequences

Sequences for stream ciphering purposes are very different from those for other purposes. It is often the case that sequences used in one stream cipher are required to have some properties which are different from those required for some other sequences employed in another stream cipher. Thus, cryptographic sequences may be different in some aspects. However, for keystream sequences for additive synchronous stream ciphers there are some common cryptographic measures of their strength such as the linear complexity (linear span or linear equivalence), sphere complexity, pattern distribution, and autocorrelation property. This section introduces some of these measures and illustrates their cryptographic importance. Here only sequences over finite fields are discussed.

2.3.1 Minimal Polynomial and Linear Complexity

To introduce linear complexity, we need the shift operator on sequences. A left shift operator E is defined by $Es_i = s_{i-1}$ for all possible i. In this way we can define recursively the operators E^l for $l > 1$. Thus, for a polynomial $f(x)$ of $GF(q)[x]$ the polynomial operator $f(E)$ is well defined, if we write $E^0 = 1$, the identity operator. If a sequence is over a finite field $GF(q)$ and $f(x)$ is a polynomial with coefficients in $GF(q)$ given by

$$f(x) = c_0 + c_1x + \cdots + c_{L-1}x^{L-1},$$

then we define

$$f(E)s_j = c_0s_j + c_1s_{j-1} + \cdots + c_{L-1}s_{j-L+1}.$$

Let s^n denote a sequence $s_0s_1 \cdots s_{n-1}$ of length n over a finite field $GF(q)$. For a finite sequence, the n is finite; for a semi-infinite sequence the n is ∞. A polynomial $f(x) \in GF(q)[x]$ of degree $\leq l$ with $c_0 \neq 0$ is called a *zero polynomial* or *characteristic polynomial* of the sequence s^n if

$$f(E)s_j = 0, \text{ for all } j \text{ with } j \geq l. \tag{2.3}$$

If the above equations hold for l, then they hold also for $l+1$. Thus, for every zero polynomial there is a least $l \geq \deg(f)$ such that the above equations hold. We call the smallest l the *associated recurrence length* of $f(x)$ with respect to the sequence. It is easy to see that there are zero polynomials of a sequence such that their associated recurrence length is minimal. Such a zero polynomial is called a *minimal polynomial* of the sequence, and the associated recurrence length is called the *linear span* or *linear complexity* of the sequence, which is denoted by $\mathrm{L}(s^n)$ hereafter. It follows immediately from this definition that $\mathrm{L}(s^n) = 0$ iff $s^n = 0^n$, where 0^n denotes the all-zero sequence of length n. Another immediate consequence is that, for $0 < n < \infty$, $\mathrm{L}(s^n) = n$ iff $s^{n-1} = 0^{n-1}$ and $s_{n-1} \neq 0$. If a semi-infinite s^∞ is periodic, then its minimal polynomial is unique if we require that $c_0 = 1$. The linear complexity of a periodic sequence is equal to the degree of its minimal polynomial.

The engineering interpretation of linear complexity is as the length of the shortest linear feedback shift register (LFSR) that generates the sequence, see Figure 2.6. Such an LFSR is said to be non-singular when $c_1 \neq 0$, i.e., when it corresponds to a linear recursion of order $\mathrm{L}(s^n)$. The minimal polynomial is actually a feedback polynomial of the LFSR. In the LFSR of length L in Figure 2.6 the boxes containing $s_{j-1}, s_{j-2}, \cdots, s_{j-L+1}, s_{j-L}$ are memory units, and with each clock tick the content of the right-most memory unit is output, while the contents of other memory units are shifted to their right-hand neighboring memory units respectively, the left-most memory unit is then occupied by the new element

$$s_j = -c_1s_{j-1} - \cdots - c_Ls_{j-L}.$$

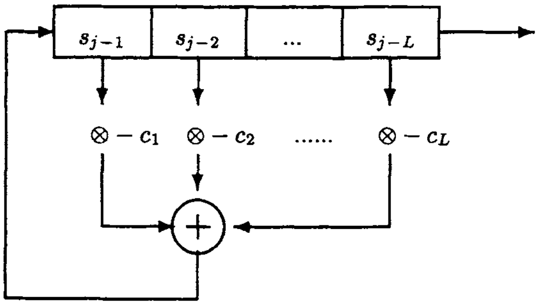

Figure 2.6: The linear feedback shift register interpretation of recursion (2.3).

The polynomial $c(x) = c_0 + c_1 x + \cdots + c_L x^L$ is called the ***connection polynomial*** or ***feedback polynomial*** of the LFSR, and the sequence s^n is said to be produced by the LFSR.

The cryptographic significance of the linear complexity of keystream sequences is well known due to the Berlekamp-Massey algorithm. If the linear complexity of a key stream is L, then $2L$ consecutive characters of the sequence could be used to construct the whole key stream with the Berlekamp-Massey algorithm. Thus, it is cryptographically necessary but not sufficient to require keystream sequences for additive stream ciphers to have large linear complexity. This will be illustrated when we introduce the sphere complexity. It should be mentioned, however, that for some nonadditive stream ciphers large linear complexity of the keystream is not a necessary cryptographic requirement (see Section 2.1.3).

Recall that in the definition of linear complexity above, the coefficients of the feedback polynomial and the entries of the sequence are required to be in the same field. Let $s^n = s_0 s_1 \cdots s_{n-1}$ be a sequence over the field $GF(q^m)$. The linear complexity of s^n with respect to the subfield $GF(q)$, here and hereafter denoted as $\mathrm{L}_{GF(q)}(s^n)$, is defined as the smallest nonnegative integer L such that there exist $c_1, c_2, \cdots, c_L \in GF(q)$ for which

$$s_j + c_1 s_{j-1} + \cdots + c_L s_{j-L} = 0, \text{ for all } L \leq j < n. \tag{2.4}$$

In this definition [138], the coefficients $c_1, c_2, \cdots, c_L$ are required to be in $GF(q)$. If $m = 1$, then the two complexities are identical. Thus, the linear complexity $\mathrm{L}_{GF(q)}$ is a generalization of the usual linear complexity. The following inequality clearly holds:

$$\mathrm{L}(s^n) \leq \mathrm{L}_{GF(q)}(s^n). \tag{2.5}$$

We now turn to the cryptographic importance of this generalized linear complexity. It is well known that $GF(q^m)$ can be regarded as a linear space of dimension m over $GF(q)$. Let $u_1, u_2, \cdots, u_m$ be a basis of $GF(q^m)$ over $GF(q)$, then each u of $GF(q^m)$ can be expressed as

$$u = \sum_{i=1}^{m} a_i u_i, \ a_1, \cdots, a_m \in GF(q).$$

Assume that for every j we have

$$s_j = \sum_{i=1}^{m} s_{i,j} u_i, \quad s_{i,j} \in GF(q),$$

then recursion (2.4) is equivalent to

$$\begin{array}{l} s_{i,j} + c_1 s_{i,j-1} + \cdots + c_L s_{i,j-L} = 0, \\ \text{for all } L \leq j \leq n, \text{ and for all } i = 1, \cdots, m. \end{array} \tag{2.6}$$

This means that the linear complexity $L_{GF(q)}(s^n)$ is the shortest length of the LFSRs which can generate the lower field sequences $s_{i,0}s_{i,1}\cdots s_{i,n-1}, i = 1, 2, \cdots, m$, at the same time only with different initial states. The determination of the shortest LFSR which generates the m sequences is called the LFSR synthesis of multisequences, which is useful in decoding cyclic codes. Several algorithms for the LFSR synthesis of multisequences have been developed [72, 154, 155, 156, 118, 138]. If we use multisequences to encipher a message stream in parallel or to use a matrix sequence to encipher, then this kind of generalized linear complexity is cryptographically important.

Now we turn to the usual linear complexity of periodic sequences. As already mentioned, the linear complexity of periodic sequences over finite fields is precisely the degree of their minimal polynomials. Cryptographically, we need to know not only the linear complexity of a sequence, but the minimal polynomial also. To introduce some results about the minimal polynomials of periodic sequences, we need the concept of generating functions.

The *formal power series* or *generating function* of a semi-infinite sequence s^∞ over $GF(q)$ is defined by

$$s(x) = \sum_{i=0}^{\infty} s_i x^i.$$

If s^∞ is periodic with period N, then we have

$$(1 - x^N)s(x) = s^N(x) = \sum_{i=0}^{N-1} s_i x^i.$$

It follows that the following proposition holds.

Proposition 2.3.1 *The generating function of each periodic sequence s^∞ can be expressed as*

$$s(x) = g(x)/f(x) \tag{2.7}$$

with $f(0) \neq 0$ *and* $\deg(g) < \deg(f)$.

The expression in (2.7) is called a ***rational form*** of the generating function $s(x)$ and of the sequence s^∞. If $\gcd(g(x), f(x)) = 1$, then it is called a ***reduced rational form***. Let us stipulate that f_s will denote the minimal polynomial of a sequence s^∞. The following two classic propositions are very useful; their proofs can be found, for example, in [276, 390, 138].

Proposition 2.3.2 *Let s^∞ be a periodic sequence over $GF(q)$ and*

$$s(x) = r(x)/f(x), \ f(0) = 1$$

a rational form of the generating function of s^∞. Then $f(x)$ is the minimal polynomial of the sequence iff $\gcd(r(x), f(x)) = 1$.

Concerning the minimal polynomial of the sum of two periodic sequences we have the following conclusion, which follows easily from Proposition 2.3.2.

Proposition 2.3.3 *Let the reduced rational forms of two periodic sequences s^∞ and t^∞ be respectively*

$$s(x) = r_s(x)/f_s(x), \ t(x) = r_t(x)/f_t(x).$$

Then the minimal polynomial of the sum sequence of the two sequences is given by

$$f_{s+t} = f_s f_t / \gcd(f_s f_t, r_s f_t + r_t f_s).$$

2.3.2 Pattern Distribution of Key Streams

Let s^∞ be a sequence of period N over $GF(q)$, where N is not necessarily the least period. The vector $(s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}})$ is called a *pattern of length* k with distances $(\tau_1, \tau_2 - \tau_1, \cdots, \tau_{k-1} - \tau_{k-2})$. A pattern $(s_t, s_{t+\tau})$ was also called a ***τ-bigram*** by Selmer [390]. The τ-bigrams were first introduced by Zierler under another name for the purpose of studying the autocorrelation of maximum period length sequences [472]. We adopt the usual terminology and call these maximum-length sequences or m-sequences.

Consider the following sequence s^∞ of period 7:

$$s^\infty = \underbrace{0111001}\underbrace{0111001}\ldots$$

and the pattern $0*1**0$, where $*$ indicates an arbitrary element. It is easily seen that this pattern appears only once in a period of the sequence.

The notion of a multiplier was first introduced by Carmichael [66]. If s^∞ is a sequence of least period N, an element $M \neq 1$ in $GF(q), q > 2$, such that

$$M \cdot s^\infty = s_\tau^\infty, \ 0 < \tau < N, \tag{2.8}$$

is called a ***multiplier*** of the sequence, where s_τ^∞ is the τ-shift version of s^∞. This multiplier is related to the multiplier of residue difference sets [15]. The τ here was called the ***span*** of M by Ward [434]. For maximum-length sequences over $GF(q)$, Zierler showed that the τ-bigrams are evenly distributed if τ is not the span of some multiplier $M \neq 1$. It is important to observe that no such multiplier exists for binary sequences.

Bigram, trigram and τ-gram are terms from linguistic studies. We will use the term pattern instead. The distribution of some special patterns was investigated by Golomb [167, 169], i.e., the runs of 0's and 1's, which were called ***gap*** and ***block*** respectively. He proved that in a binary maximum-length sequence of period $2^n - 1$, there are 2^n runs. Half the runs have length 1, one fourth have length 2, one-eighth have length 3, etc., until two runs of length $n-2$; for each of these lengths, there are equally many gaps and blocks. Finally, there is one gap of length $n-1$ and one block of length n. These are Golomb's three randomness postulates.

Why is the pattern distribution property of a keystream cryptographically important? Some intuitive facts can only give us some superficial reasons. For example, the pattern distribution of length 1 is in fact the distribution of the elements of the field in the sequence. Thus, if there are many more 1's than 0's in a binary sequence, then the sequence is not cryptographically good. To see the cryptographic importance of a roughly equally likely distribution for certain patterns, we first prove a conservation law of patterns.

Consider now the patterns of length k with distances $(\tau_1, \tau_2 - \tau_1, \cdots, \tau_{k-1} - \tau_{k-2})$. For a sequence with least period N over $GF(q)$, the vector variable $(s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}})$ takes on vectors of $GF(q)^k$ when t ranges from 0 to $N-1$. Let $n((s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}}) = a)$ denote the number of times with which the vector variable $(s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}})$ takes on $a \in GF(q)^k$ when t ranges from 0 to $N-1$. It is straightforward to see that the following theorem holds.

Theorem 2.3.4 (The conservation law of patterns) *Let the symbols be the same as before, then*

$$\sum_{a \in GF(q)^k} n((s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}}) = a) = N.$$

Clearly, this theorem means that these $n((s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}}) = a)$ are conservative. The constant $n((s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}}) = a)/N$, denoted as $\Pr(a)$, is the probability that $(s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}})$ takes on a. It follows that

$$\sum_{a \in GF(q)^k} \Pr(a) = 1. \tag{2.9}$$

In general, *bad patterns* refer to those which appear with small probability in the key streams. If in a sequence of period N over $GF(q)$ the almost equally likely distribution of a pattern of length k with distances $(\tau_1, \tau_2 - \tau_1, \cdots, \tau_{k-1} - \tau_{k-2})$ is required, this means that $n((s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}}) = a)$ is approximately a constant, namely N/q^k. Now the question is why such a uniform pattern distribution should be cryptographically required. This can be shown by the differential attack on the natural sequence generator of Figure 2.5(b) [122]. The idea behind the attack is that bad patterns give much more information about the key than other patterns. That differential cryptanalysis implies the following general randomness requirement for keystream sequences of the NSG:

Randomness requirements: In a sequence of least period N over $GF(q)$ for each k with $1 \leq k \leq \lfloor \log_q N \rfloor$, the pattern $((s_t, s_{t+\tau_1}, \cdots, s_{t+\tau_{k-1}}) = a)$ of length k appears approximately $\lfloor N/q^k \rfloor$ times when t ranges from 0 to $N-1$.

This requirement may be reasonable only for some applications. It is known that uniform pattern distributions in a sequence result in good autocorrelation properties of the sequence, which will be seen in the next subsection.

2.3.3 Correlation Functions

Many problems in ranging systems, radar systems, spread-spectrum communication systems, multiple-terminal system identification, and code-division multiple-access communications systems require sets of signals which have one or both of the following properties:

- Each signal in the set is easy to distinguish from a time-shifted version of itself;

- each signal in the set is easy to distinguish from (a possibly time-shifted version of) every other signal in the set.

This leads to an intensive study of the periodic autocorrelation and crosscorrelation functions of sequences (see Sarwate [381], Helleseth and Kumar [200], also [417, 419, 420]).

Let $GF(q)$ be a finite field. We need the idea of an additive character of $GF(q)$ (see [276], p. 190). Let χ be an additive character of $GF(q)$, and s^∞ and t^∞ be two sequences of period respectively N and M and $P = \text{lcm}\{M, N\}$. Then the ***periodic crosscorrelation function*** of the two sequences is defined by

$$\text{CC}_{s,t}(l) = \sum_{i=0}^{P-1} \chi(s_i - t_{i+l}) = \sum_{i=0}^{P-1} \chi(s_i)\overline{\chi(t_{i+l})}. \tag{2.10}$$

If the two sequences are identical, then $P = M = N$ and the crosscorrelation function is the so-called ***periodic autocorrelation function*** of s^∞ described by

$$\text{AC}_s(l) = \sum_{i=0}^{N-1} \chi(s_i - s_{i+l}) = \sum_{i=0}^{N-1} \chi(s_i)\overline{\chi(s_{i+l})}. \tag{2.11}$$

If $q = 2$, then $\chi(a) = (-1)^a$ is an additive character of $GF(2)$, here we identify $GF(2)$ with Z_2. Then (2.10) and (2.11) are the usual crosscorrelation and autocorrelation functions of binary sequences.

For two sequences s^∞ and t^∞ of period respectively N and M and $P = \text{lcm}\{M, N\}$, the ***aperiodic crosscorrelation function*** of the two sequences is defined by

$$\text{ACC}_{s,t}(l, u, v) = \sum_{i=u}^{v} \chi(s_i - t_{i+l}) = \sum_{i=u}^{v} \chi(s_i)\overline{\chi(t_{i+l})}, \tag{2.12}$$

where $\overline{x}$ denotes the complex conjugate of x. If the two sequences are identical, then $P = M = N$ and the crosscorrelation function is the so-called ***aperiodic autocorrelation function*** of s^∞ described by

$$\text{AAC}_s(l, u, v) = \sum_{i=u}^{v} \chi(s_i - s_{i+l}) = \sum_{i=u}^{v} \chi(s_i)\overline{\chi(s_{i+l})}. \tag{2.13}$$

Here our definitions of aperiodic functions are slightly different from those used for communication purposes [381], which are defined to be

$\mathrm{AAC}_{s,t}(l,0,v)$ and $\mathrm{AAC}_s(l,0,v)$ respectively. Aperiodic autocorrelation results may be much more important cryptographically than periodic autocorrelation results, since the former reflect local randomness and the latter reflect global randomness: what we actually use for stream ciphering is only a small piece of periodic sequences. Generally speaking, the periodic autocorrelation is relatively much easier to control than the aperiodic autocorrelation. The connections between the autocorrelation function and other cryptographic notions will be described in detail for the binary case in Section 2.4

2.3.4 Sphere Complexity and Linear Cryptanalysis

Let x be a finite sequence of length n over $GF(q)$. The *weight complexity* [137, 138] of the finite sequence is defined by

$$\mathrm{WC}_u(x) = \min_{\mathrm{WH}(y)=u} \mathrm{L}(x+y), \tag{2.14}$$

where $\mathrm{WH}(y)$ denotes the Hamming weight of y, i.e., the number of components of y that are different from zero.

Consider now the space $GF(q)^n$ with Hamming distance d_H. Denoting $S(x,u) = \{y : d_H(x,y) = u\}$, by definition we have

$$\mathrm{WC}_u(x) = \min_{y\in S(x,u)} \mathrm{L}(y).$$

This means that the weight complexity is the maximum lower bound of linear complexities of all the sequences of length n on the sphere surface $S(x,u)$. The name of this kind of complexity comes from this geometrical meaning.

Let $O(x,u) = \{y : 0 < d_H(x,y) \le u\}$ be the sphere with center x. The *sphere complexity* [137, 138] is defined by

$$\mathrm{SC}_u(x) = \min_{y\in O(x,u)} \mathrm{L}(y) = \min_{0<v\le u} \mathrm{WC}_v(x). \tag{2.15}$$

Similarly, let s^∞ be a sequence of period N (not necessarily least period) over $GF(q)$. The weight and sphere complexity of periodic sequences with respect to N are defined respectively by

$$\mathrm{WC}_u(s^\infty) = \min_{\mathrm{WH}(t^N)=u,\mathrm{per}(t^\infty)=N} \mathrm{L}(s^\infty + t^\infty) \tag{2.16}$$

$$\mathrm{SC}_u(s^\infty) = \min_{0<v\le u} \mathrm{WC}_v(s^\infty), \tag{2.17}$$

where $\mathrm{per}(t^\infty) = N$ denotes that t^∞ has period N.

These two complexities were introduced to measure the stability of the linear complexity function, in analogy to the derivative of functions in Euclidean spaces. The cryptographic background of these complexities is that some key streams with large linear complexity can be approximated by some sequences with much lower linear complexity [137, 138]. The sphere and weight complexity are based on the LFSR approximation model. In contrast to the linear complexity which is based on the shortest LFSR that produces a sequence, the sphere complexity $\mathrm{SC}_k(s^\infty)$ is based on the shortest LFSR that produces another sequence with a probability of agreement no less than $(1 - k/N)$, where N is a period of the sequence s^∞ with which the sphere complexity is concerned. The weight complexity $\mathrm{WC}_k(s^\infty)$ is based on the shortest LFSR that produces another sequence with a probability of agreement equal to $(1 - k/N)$.

To illustrate the difference between the linear complexity and sphere complexity, we consider the binary sequence of period N:

$$s^\infty = \underbrace{0....01}_{N}\underbrace{0....01}_{N}...$$

The linear complexity of the sequence is N by definition since no LFSR of length less than N can produce it. However its sphere complexity $\mathrm{SC}_1(s^\infty) = 1$ by definition since there is an LFSR of length one that produces the all-zero sequence having the probability $(1 - 1/N)$ of agreement with the sequence s^∞.

These LFSR approximation model complexities are cryptographically important only if there is an efficient algorithm to find the LFSR for approximating the original generator. To see the cryptographic importance of these complexities, we describe an attack on all synchronous additive stream ciphers [123].

Suppose that a cryptanalyst has a number of consecutive ciphertext-plaintext pairs of a synchronous additive stream cipher which enable him to derive a piece of key stream, say $z_0 z_1 \cdots z_{n-1}$. Suppose also that he knows nothing else but the plaintext source code of the enemy's messages. What can he do under these assumptions in order to decipher the enemy's ciphertext? The best the cryptanalyst can do may be the construction of a new generator which produces a sequence with a large probability of agreement with the original key stream.

To make things simple, we assume that the key stream is binary. Under the assumption that the linear complexity of the enemy's key stream is very unstable, the cryptanalyst can try to construct an LFSR to approximate the original keystream generator according to the following procedure:

Step 1 Use the Berlekamp-Massey algorithm to construct an LFSR which produces the sequence $z^n = z_0 z_1 \cdots z_{n-1}$. Then use the constructed LFSR to decipher a large piece of ciphertext. If only ϵ percent (this constant can be flexible, say less than 15) of the deciphered ciphertext makes no sense, then accept the LFSR and stop; otherwise, go to Step 2.

Step 2 For $i = 0$ to $n-1$, do the following: Change z_i into $z_i \oplus 1$. Apply the Berlekamp-Massey algorithm to the new sequence to construct an LFSR which produces the new sequence. Then use the constructed LFSR to decipher a large piece of ciphertext. If only ϵ percent (this constant can be flexible, say less than 15) of the deciphered ciphertext makes no sense, then accept the LFSR and stop; otherwise, repeat this step for $i+1$ if $i < n-1$, and go to Step 3 if $i = n-1$.

Step 3 For a possible pair (i, j) with $i < j$ and $i, j \in \{0, 1, ..., n-1\}$, change z_i into $z_i \oplus 1$ and z_j into $z_j \oplus 1$. Then apply the Berlekamp-Massey algorithm to the new sequence to construct an LFSR which produces the new sequence. Then use the constructed LFSR to decipher a large piece of ciphertext. If only ϵ percent (this constant can be flexible, say less than 15) of the deciphered ciphertext makes no sense, then accept the LFSR and stop; otherwise, repeat this step for the next pair (i, j) with $i < j$ if there is a remaining pair, and print "fail" and stop if there is no pair remaining.

Since the complexity of Berlekamp-Massey algorithm for sequences of length n is $O(n^2)$, the complexity of this attack is $O(n^4)$. Thus, if s^∞ is a key stream such that its linear complexity is very large (say, for example, 2^{40}) and $\mathrm{SC}_k(s^\infty)$ is small enough (say less than 1000 for example) for some very small k, then this attack must succeed. The basic idea of this attack is that, we expect that the keystream sequence can be expressed as

$$z^\infty = u^\infty + v^\infty$$

such that u^∞ and u^∞ are of period N, the $\mathrm{WH}(v^N)/N$ is very small and the sequence u^∞ has small linear complexity. This can be done when the linear complexity of the key stream is very unstable. In this case, we expect that the known key stream z^n can be expressed as

$$z^n = u^n + v^n$$

with $\mathrm{WH}(v^n) \leq 2$ if $n < 2N/k$. Furthermore, we may also use the regular decimation sequences of z^n to replace z^n, then derive the minimal polynomial of u^n from the decimated sequences. Thus, it follows that the designer of a synchronous additive stream cipher must ensure that for very small

k's, the sphere complexity $\mathrm{SC}_k(s^\infty)$ is large enough. In other words, the designer of an additive synchronous stream cipher should make sure that his key stream cannot be well approximated by a sequence with small linear complexity, since the above polynomial-time algorithm can be used to find an LFSR to approximate the original keystream sequence if its linear complexity is very unstable. This shows why sphere complexity is cryptographically important. For the purpose of measuring the linear complexity stability of sequences, fixed-complexity distance and variable-complexity distance were introduced. The connection between these measures and some lower bounds on these measures for some sequences can be found in [138].

The linear complexity stability problem for sequences was also considered by Stamp and Martin [412] under the name of k-error linear complexity which is defined to be $\min\{\mathrm{SC}_k(s), \mathrm{L}(s)\}$ and is essentially the same as sphere complexity.

Note that the motivation behind the sphere complexity is linear cryptanalysis on two kinds of stream ciphers introduced by Ding, Xiao and Shan in 1988 [137], see also [119, 138]. The basic idea of linear cryptanalysis for stream ciphers is to use a related linear system to approximate the original highly nonlinear system, or in other words to use linear circuits to approximate nonlinear circuits. For details about the linear cryptanalysis of two kinds of stream ciphers and specific examples we refer to [138]. We note that the linear cryptanalysis for stream ciphers was done earlier than that for block ciphers.

2.3.5 Higher Order Complexities

Linear complexity (also called *linear span*) of a sequence is defined to be the length of the shortest LFSR that generates the sequence. A sequence with very large linear span may be generated by a much shorter FSR (feedback shift register) if nonlinear terms are allowed in the feedback function. If only quadratic terms and linear terms are allowed, then we have the *quadratic span* [157, 69]. If general terms are allowed, then we have the *maximum order complexity* [219] or generally *span* [69].

The quadratic span and other nonlinear spans are cryptographically important only when there are efficient algorithms for finding the shortest nonlinear FSRs or an FSR that is short enough. Suppose there is an efficient algorithm for finding the shortest quadratic FSR that generates any given sequence; then we need to investigate further the relations between the linear span and quadratic span. It is obvious that the linear span of any sequence is greater than or equal to the quadratic span of the sequence. If the relation

$$L(s^\infty) \geq Q(s^\infty) \geq \sqrt{L(s^\infty) + a},$$

holds for every periodic binary sequence, where $Q(s^\infty)$ denotes the quadratic span of the sequence s and a is a constant, then control of the linear span results also in control of the quadratic span. Thus investigation of the following research problem is interesting.

Research Problem 2.3.5 *Investigate whether there are constants a and b such that*

$$Q(s^\infty) \geq \sqrt{L(s^\infty) \times b + a}$$

for each sequence of period N over $GF(q)$, where the constants depend only on the period and q.

2.4 Harmony of Binary NSGs

The NSG of Figure 2.5(b) is cryptographically attractive because not only can every periodic sequence be produced with a proper choice of the parameters of the generator, but also many of its security aspects are consistent. For the binary natural sequence generator the following cryptographic analyses are equivalent:

1. differential analysis of the cryptographic function $f(x)$;
2. nonlinearity analysis of the cryptographic function $f(x)$;
3. autocorrelation analysis of the cryptographic function $f(x)$;
4. autocorrelation analysis of the output sequence;
5. two-bit pattern distribution analysis of the output sequence;
6. stability analysis of the mutual information $I(i;\ z_i z_{i+t-1})$ (here and hereafter z^∞ denotes the output sequence of the NSG);
7. transdensity analysis of the additive stream cipher system with this NSG as the keystream generator (by which we mean the analysis of the probability of agreement between two encryption resp. decryption transformations specified by two encryption resp. decryption keys [122]).

Equivalence is understood in the sense that one analysis gives another analysis, and conversely.

We now prove the equivalence between the above seven analyses and show that the "ideal difference property" of the cryptographic function $f(x)$ ensures automatically

- ideal nonlinearity of the cryptographic function $f(x)$,
- ideal autocorrelation property of $f(x)$,
- ideal autocorrelation property of the output sequence z^∞,
- ideal two-bit pattern distribution property of the output sequence z^∞, and
- ideal balance between the mutual information $I(i;\ z_i z_{i+t-1})$ for all possible pairs $(z_i, z_{i+t-1}) \in Z_2 \times Z_2$, where t is arbitrary.

In what follows Z_N denotes the residue class ring modulo an integer N. Our notation for the autocorrelation function in this section is different from the one in the last section for the sake of simplicity.

Consider now the NSG of Figure 2.5(b). Assume that $(G, +)$ is the Abelian group over which the keystream sequence is constructed, and $|G| = n$. For each $g_i \in G$ let

$$C_i = \{x \in Z_N : f(x) = g_i\}.$$

The ordered set $\{C_0, C_1, \cdots, C_{n-1}\}$ is called the *characteristic class.* For any ordered partition $\{C_0, C_1, \cdots, C_{n-1}\}$ of Z_N, there exists a function $f(x)$ with this partition as its characteristic class. The differential analysis of the system is the analysis of the following *difference parameters:*

$$d_f(g_i, g_j; w) = |C_i \cap (C_j - w)|, \quad (g_i, g_j) \in G \times G,\ w \in Z_N.$$

We say that f has the *ideal difference property* if the values $d_f(g_i, g_j; w)$ are approximately the same for all possible $(g_i, g_j; w)$.

To see why the analysis of the difference parameters can be regarded as a kind of differential analysis, we take $(G, +) = (Z_2, +)$. Consider the input pairs (x, y) such that $x - y = a$, and consider the difference of the corresponding output pairs. Then we have the following expressions

$$\frac{|\{(x,y): f(x) - f(y) = 1,\ x - y = a\}|}{|\{(x,y): x - y = a\}|} = \frac{d_f(0,1;a)}{N} + \frac{d_f(1,0;a)}{N}$$
$$\frac{|\{(x,y): f(x) - f(y) = 0,\ x - y = a\}|}{|\{(x,y): x - y = a\}|} = \frac{d_f(0,0;a)}{N} + \frac{d_f(1,1;a)}{N},$$

These two expressions show that the difference parameters can be regarded as partial differentials or directional differentials of the function $f(x)$.

In what follows we prove the equivalence between the above seven analyses for the binary NSG (natural sequence generator).

Between differential and nonlinearity analysis

Let $g(x)$ be a mapping from an Abelian group $(G, +)$ to another one $(H, +)$. The nonlinearity of g is measured by

$$P_g = \max_{0 \neq a \in G} \max_{b \in H} \Pr(f(x+a) - f(x) = b),$$

where $\Pr(A)$ denotes the probability of the occurrence of event A. Here $P_g(a)$ could be called the differential of $g(x)$ at a. However, elementary calculus shows that differentials are ideal measures for nonlinearities. We shall deal with highly nonlinear functions in details in Chapter 6.

The nonlinearity analysis of the cryptographic function $f(x)$ refers to the analysis of the probability $\Pr(f(x+a) - f(x) = b)$. It can be easily seen that

$$\begin{aligned} N \Pr(f(x) - f(x-a) = 1) &= d_f(0,1;a) + d_f(1,0;a), \\ N \Pr(f(x) - f(x-a) = 0) &= d_f(0,0;a) + d_f(1,1;a) \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} 2d_f(0,0;-a) &= |C_0| - |C_1| + N \Pr(f(x+a) - f(x) = 0), \\ 2d_f(1,1;-a) &= |C_1| - |C_0| + N \Pr(f(x+a) - f(x) = 0), \\ 2d_f(1,0;-a) &= 2d_f(0,1;-a) = N - N \Pr(f(x+a) - f(x) = 0). \end{aligned} \tag{2.19}$$

Then formulae (2.18) and (2.19) show the equivalence.

Between differential and autocorrelation analysis

The autocorrelation analysis of $f(x)$ refers to the analysis of the (normalized) autocorrelation function

$$\mathrm{AC}_f(a) = \frac{1}{N} \sum_{x \in Z_N} (-1)^{f(x+a) - f(x)}.$$

It is easily verified that

$$N\mathrm{AC}_f(a) = N - 4d_f(1,0;a) \tag{2.20}$$

and

$$\begin{aligned} 4d_f(0,0;a) &= 4|C_0| - N + N\mathrm{AC}_f(a), \\ 4d_f(1,1;a) &= 4|C_1| - N + N\mathrm{AC}_f(a), \\ 4d_f(1,0;a) &= 4d_f(0,1;a) = N - N\mathrm{AC}_f(a). \end{aligned} \tag{2.21}$$

Combining formulae (2.20) and (2.21) proves the equivalence between the differential and autocorrelation analysis of $f(x)$.

The autocorrelation analysis of the output binary sequence z^∞ refers to the analysis of the autocorrelation function

$$\mathrm{AC}_z(a) = \frac{1}{N} \sum_{i \in Z_N} (-1)^{z_{i+a} - z_i}.$$

Clearly by the definition of the NSG we have

$$\mathrm{AC}_z(a) = \mathrm{AC}_f(a), \quad \text{for each } a.$$

Thus, the above formulae (2.20) and (2.21) are also true if we replace $C_f(a)$ with $C_z(a)$. This fact shows the equivalence between the differential analysis and the autocorrelation analysis of the output sequence z^∞.

Between differential and two-bit pattern distribution analysis

The two-bit pattern distribution analysis of z^∞ is concerned with how the two-bit patterns are distributed (see Section 2.3.2). For each fixed t with $0 < t \le N-1$ the vector (z_i, z_{i+t}) takes on elements of $Z_2 \times Z_2$ when i ranges from 0 to $N-1$. Let $n[(z_i, z_{i+t}) = (a,b)]$ denote the number of times which the vector (z_i, z_{i+t}) takes on $(a,b) \in Z_2 \times Z_2$ when i ranges from 0 to $N-1$. Then we have obviously

$$n[(z_i, z_{i+t}) = (a,b)] = d_f(a, b; -t). \tag{2.22}$$

Thus, for the binary NSG each difference parameter represents in fact the number of times with which a two-bit pattern appears in a segment of length N of the binary output sequence z^∞.

Between differential and mutual information analysis

We are given two bits z_i and z_{i+t} of the output sequence of the binary NSG. It is cryptographically interesting to know how much information these two bits give to the content of the register of the counter in the binary NSG at the time the output bit z_i was produced. It is easy to verify

$$I(i; z_i z_{i+t}) = \log_2 N - \log_2 d_f(z_i, z_{i+t}; -t) \quad \text{bits} \tag{2.23}$$

and

$$d_f(z_i, z_{i+t}; -t) = N 2^{-I(i; z_i z_{i+t})}, \tag{2.24}$$

where the mutual information $I(i; z_i z_{i+t})$ is measured in bits. Formulae (2.23) and (2.24) clearly show the equivalence. In addition they show that the difference parameters are in fact a measure of uncertainty.

Between differential and transdensity analysis

In a cipher system it is possible for two keys to determine the same encryption (resp. decryption) transformation. Even if the two transformations are distinct, it is cryptographically interesting to know the probability of agreement between the ciphertexts given by the two transformations. The control of this probability of agreement may protect a cipher from a key approximation attack, that is, the use of one key to decrypt a message encrypted by another key. Let E_k (resp. D_k) denote the encryption (resp. decryption) transformation specified by the key k. The analysis of the density (briefly, transdensity analysis) of a cipher system refers to the analysis of the probability of agreement $\Pr(E_k(m) = E_{k'}(m))$, where m can be restricted to plaintext blocks or without restriction [122].

For the additive binary stream cipher with the binary NSG as its keystream generator this probability can be expressed easily as

$$\Pr(E_k = E_{k'}) = \mathrm{AC}_z(k - k' \bmod N) = \mathrm{AC}_f(k - k' \bmod N), \qquad (2.25)$$

because of the additive structure of the additive stream cipher and the fact that the keystream sequences specified by all keys are shift versions of each other. Thus, the equivalence follows easily from formula (2.25).

So far we have proved the equivalence between differential analysis and the other six analyses. Thus, equivalence among the seven analyses follows. In addition, there is no trade-off between all the above seven properties and the linear complexity and its stability for this generator (we will see this fact in later chapters). This means that it is possible to design the NSG so that it is not only ideal with respect to all seven properties, but also has large linear complexity and ideal linear complexity stability for the output sequence. It is because of these facts and because every periodic sequence can be produced by the natural sequence generator that the generator is called a natural one [122].

Formulae 2.18–2.25 clearly show that to ensure ideal behavior with respect to all seven properties, it suffices to control the difference property of the cryptographic function $f(x)$. Thus, in later chapters we will concentrate on the control of the difference property of $f(x)$, of the linear complexity, and of the sphere complexity of the output sequence for each specific NSG.

2.5 Security and Attacks

Without attacks on cipher systems there would be no problem of security. Security is associated with attacks and is usually relative to attacks. Attacks are also relative to the assumptions about a cryptanalyst's knowledge of a cipher system.

Attacks can be classified according to the assumed available information about a cipher that the cryptanalyst has. This kind of classification results in three types of attacks: (A) ***ciphertext-only attacks*** under the assumption that only pieces of ciphertext are known to a cryptanalyst; (B) ***known-plaintext attacks*** under the assumption that a piece of ciphertext with corresponding plaintext is known; (C) ***chosen-plaintext attacks*** under the assumption that a cryptanalyst has a chosen piece of plaintext with corresponding ciphertext.

Suppose that a cryptanalyst has got a piece of keystream sequence, then there are two further assumptions concerning attacks on stream ciphers:

B1: It is assumed that the cryptanalyst knows only a piece of keystream. In this case there are the following possible attacks: (1) ***equivalent-machine attacks***, which make use of the piece of key stream to construct a new generator which produces the same key stream. For example, if the linear complexity of the key stream is not very large, the Berlekamp-Massey algorithm can be used to construct an LFSR which produces the same key stream. (2) ***Approximate-machine attacks***, which make use of the known piece of key stream to construct another generator to approximate the original generator. One example of these attacks is the attack based on the linear complexity stability of the key stream described in Section 2.3.4.

B2: Apart from a piece of keystream it is assumed that the cryptanalyst also knows the type of generator and the cryptographic algorithm. Under these assumptions attacks are flexible in forms and in techniques, for example, those in [4, 5, 138, 122, 471]. Attacks under these assumptions depend on specific systems and on the technique a cryptanalyst uses. In this case, there is one more type of attack than in the case (B1), i.e., ***key recovering attacks***, which aim to recover the original key or equivalent keys.

For ciphertext-only attacks, some other information about a cipher is usually assumed to be known to a cryptanalyst. For example, under the assumption that the structure of the keystream generator is known there are several kinds of correlation attacks [401, 306] and key-recovering attacks.

The general idea of the key-recovering attacks is to make use of known data about a cipher to get information about the key. The techniques of attack vary with the structure of the ciphers and with the known data. The following "mother problem" may illustrate the flexibility of such attacks.

Suppose that there are ten children in such an order $z_1 z_2 \cdots z_{10}$ that z_i is older than or has the same birthday as z_{i+1}. We are told that these children have the same father F which is known and also the same mother which could be any one of the 100 mothers $\{M_1, \cdots, M_{100}\}$. If we have further information about the pairs (F, M_i), then we can make use of it to make the set containing the mother smaller or to determine the mother. For instance, by studying the known information about all (F, M_i), suppose we know that twins are only possible for (F, M_1). Then if twins are found in the 10 children, the mother must be M_1. Of course, it may be technically difficult to find this character of (F, M_1). In fact we can identify the keys with mothers, the algorithm with the father, and the ciphertext with the children.

To determine the mother, one person may try to study the distribution of sexes among the children because she has some technique to study the sex of possible children of each (F, M_i). Another person may try to study the color of the hair of the mothers, father and children. Others may study the blood types of father, mothers and children and the age distribution of the mothers and children. Every method must use the idea of "consistency" or "correlation" either in an obvious way or in a hidden way. Perhaps everyone will have her own technique for getting information about the mother.

Keystream generators are flexible and diverse. Each generator may have its own special security aspects, although some common requirements exist (e.g., linear and sphere complexity). Some cipher systems are easy to implement, but may have tradeoffs between known security parameters; some are relatively difficult to implement, but their security may be easy to control; others may have both an easy implementation and ideal security, but be slow. Of course, fewer tradeoffs make the design easier. In designing secure cipher systems the most important problems are:

1. How can we build systems which have as few security tradeoffs as possible?

2. What are the tradeoffs or conflicts in a given system?

3. How do we manage tradeoffs and conflicts?

4. How do we coordinate security and performance?

We consider such questions in later chapters of this book.