Reihe Informatik
04-2004

Cryptanalysis of LFSR-based Pseudorandom Generators - a Survey

Erik Zenner

Cryptanalysis of LFSR-based Pseudorandom Generators - a Survey

Erik Zenner

Theoretische Informatik
University of Mannheim (Germany)
e-mail: `zenner@th.informatik.uni-mannheim.de`

Abstract. Pseudorandom generators based on linear feedback shift registers (LFSR) are a traditional building block for cryptographic stream ciphers. In this report, we review the general idea for such generators, as well as the most important techniques of cryptanalysis.

1 Security Model

1.1 Shannon's model

Basic setting: The most basic task of cryptography is encryption. The setting was captured by Shannon in [47] as a modification of his well-known communication model, proposed in [46]. Consider two entities, named *sender* and *receiver*, who want to transmit an arbitrary message at an arbitrary point in time in complete privacy. There are two communication channels available:

– The *secret channel* is completely confidential. No information that is transmitted using this channel can be observed by a third party. However, the secret channel has the disadvantage of being available only at fixed points in time (e.g., when sender and receiver meet in person).
– The *public channel* can be observed by any interested third party. Thus, all information transmitted using this channel can be considered public. As opposed to the secret channel, the public channel is available at any time.

It is obvious that a confidential message can not be sent across the secret channel, since it might not be available at the desired time. Nor can it be sent across the public channel, since it can be observed by third parties.

Encryption schemes: The use of an *encryption scheme* (or *cipher*) is the traditional solution to the above problem. Such a scheme consists of the following components:

1. A set $\mathcal{M}$ of *messages*, a set $\mathcal{C}$ of *ciphertexts* and a set $\mathcal{K}$ of *keys*.
2. A pair of functions $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and $D : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$, being computable by efficient algorithms and satisfying the following property:

$$D(k, E(k, m)) = m \qquad \forall m \in \mathcal{M},\ k \in \mathcal{K} \tag{1}$$

E is denoted as *encryption function* and D as *decryption function*. Note that in order to meet condition (1), $E(k, \cdot)$ has to be a permutation and $D(k, \cdot)$ its inverse for all $k \in \mathcal{K}$.

In a first step, sender and receiver agree on such an encryption scheme, using the public channel. They also exchange a key $k \in \mathcal{K}$, using the secret channel. Note that from now on, the knowledge about the key is all that distinguishes a legitimate sender and receiver from an arbitrary third party (Kerckhoff's principle).

Now sender and receiver are prepared to communicate privately as follows. Given a message $m \in \mathcal{M}$, the sender encrypts m under the key k by calculating $c = E(k, m)$. The ciphertext c is then transmitted using the public channel. On the receiver's side, *decryption* is performed by converting c back into $m = D(k, c)$, thus yielding the original message.

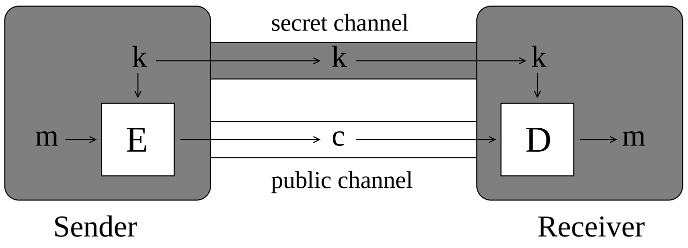

Fig. 1. Shannon's model

The process of encryption, transmission and decryption is depicted in figure 1. Note that all information on white background is visible to all interested parties, while information on gray background is only available to the sender or receiver, respectively. In particular, a casual observer is aware of the functions E and D and of the ciphertext c. In some cases, he may be able to derive information about the message m or the key k from this data (e.g., if $E(k, \cdot)$ is the identical permutation). Informally, such an encryption will be called "insecure". However, in order to find "secure" encryption functions, an informal notion is not enough. What we need is a more precise concept of security.

1.2 Attacker model

In order to gain an understanding of security, we have to introduce a malign third party that has access to all public information in the above model and tries to derive some of the secret information from it. Such a third party will be denoted as an *attacker*, the algorithm employed by him as *attack*. Once the attacker is defined, the notion of security is derived in a straightforward way: A system is secure if the attacker is unable to achieve his goal.

By definition, the attacker knows the encryption and decryption functions E and D. He also has access to all information transmitted over the public channel. He can not, however, do anything but listen to the communication channel and do his own computations. In particular, he must not remove, change or add data on the public communication channel. Thus, he is called a *passive attacker*.

Type of attack: The attacker can mount known-plaintext attacks. This means that the attacker knows the ciphertext c and part of the corresponding message m. Note that such an attacker is stronger than an attacker who is limited to ciphertext-only attacks, increasing the probability of finding security problems.

Computational resources: We assume the attacker to operate on a uniform computational model, like a Turing machine, a Random-access machine, or a personal computer. He is able to conduct all computational operations that run faster than a brute-force search on the key space. Similarly, he is limited to a memory space that is smaller than what would be necessary in order to save all keys.

As a first indication, the computational requirements of an attack are given in asymptotical form. However, in order to avoid the pitfalls of asymptotics (like hidden large factors), all attacks are also implemented. Running time or memory space estimates will be given based on experimental data for small key lengths.

Notion of security: Given the ciphertext c and a piece of the message, there are two possible goals for the attacker:

1. Finding the set $\mathcal{M}' \subseteq \mathcal{M}$ of all *message candidates*. A message $m' \in \mathcal{M}$ is a message candidate if it matches the known piece and if $\exists k \in \mathcal{K}$ such that $E(k, m') = c$.
2. Finding the set $\mathcal{K}' \subseteq \mathcal{K}$ of all *key candidates*. A key candidate k' is defined via
$$k' \in \mathcal{K}' \quad \Leftrightarrow \quad \exists m' \in \mathcal{M}' \, : \, E(k', m') = c$$

Note that the second goal is more ambitious. Once $\mathcal{K}'$ is known, it is possible to reconstruct $\mathcal{M}'$ by calculating $m' = D(k', c)$ for all $k' \in \mathcal{K}'$. On the other hand, deriving the set of key candidates from the set of message candidates is usually not feasible.

1.3 One-time pad and pseudorandom generators

One-time pad (OTP): In [51], G. Vernam introduced a simple encryption algorithm. Let $m, c, k \in \{0,1\}^n$, then the encryption function is $E(k, m) = k \oplus m$ and the corresponding decryption function is $D(k, c) = k \oplus c$. Here, $\oplus$ denotes the bitwise xor of its operands.

It can be proven [47] that this encryption scheme is indeed perfectly secure if a random key is available that is never re-used for a second encryption (thus the name *one-time pad*). This implies, however, that the key must be as long as the message to be encrypted. As mentioned in section 1.2, managing keys of appropriate size is usually not feasible.

Pseudorandom generator (PRG): A *pseudorandom generator* is a function $G : \{0,1\}^l \to \{0,1\}^*$ that expands a short *seed* into a bit sequence of arbitrary length. In order to be of cryptographic interest, G has to be computable by an efficient algorithm. In practice, it is implemented by a finite state machine with output, as displayed in figure 2. The components of such a generator are (see, e.g., [42]):

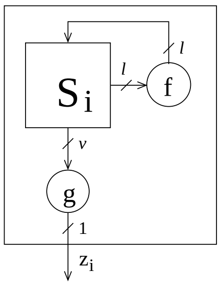

Fig. 2. Pseudorandom Generator

1. An inner state $S_i \in \{0,1\}^l$,
2. an update function $f : \{0,1\}^l \to \{0,1\}^l$ that modifies the inner state between two outputs, and
3. an output function $g : \{0,1\}^v \to \{0,1\}$, $v \leq l$, that computes the next output bit from (part of) the current inner state.

Note that the seed value S_0 and the relation $S_i = f(S_{i-1})$ form a recurrence, defining the sequence of all inner states that the generator assumes over time. Also note that the generator can assume at most 2^l different inner states, yielding an upper bound on the least period of 2^l.

Deployment of PRG: Given a PRG G, a seed value S_0 can be expanded into a *keystream* $z = G(k)$ of arbitrary length. This allows us to construct a *pseudo-OTP*, using an encryption function $E(k, m) = z \oplus m$ and a corresponding decryption function $D(k, m) = z \oplus c$, as described in figure 3.

Note that in general, the seed S_0 must not be confused with the key k. In practice, S_0 is generated from k (and possibly some additional information) by some initialisation function. Thus, overall security depends both on this initialisation function and on G.

Remember, however, that for cryptographic systems, every component should be as strong as possible (cf., e.g., [9]), independently of the other building blocks. Thus, when considering the security of the PRG, we ignore the existence of an initialisation function and assume that the seed is equal to the key, i.e. $S_0 = k \in \{0,1\}^l$. For this reason, in the next sections, the terms "seed" and "key" will be used synonymously when considering the security of a PRG.

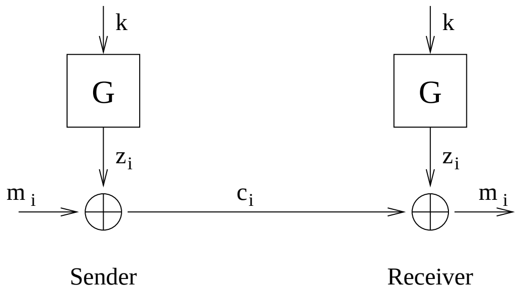

Fig. 3. Deployment of Pseudorandom Generator

Security of a pseudorandom generator: In cryptography, a pseudorandom generator G is secure iff a pseudo-OTP using G is secure.

Remember that the attacker can mount known-plaintext attacks, meaning that he knows the ciphertext $c = c_1, \ldots, c_n$ and some message bits $m_{i_1}, \ldots, m_{i_s}$, where $\{i_1, \ldots, i_s\} \subset [n]$. Note that this is equivalent to giving the attacker the corresponding keystream bits $z_{i_1}, \ldots, z_{i_s}$ right away.

Success is defined as the ability to find either the set of message candidates or the set of key candidates.

1. Finding the set $\mathcal{M}'$ of consistent messages is equivalent to finding the set $\mathcal{Z}'$ of consistent keystreams, which is defined as follows:
$$z' \in \mathcal{Z}' \quad \Leftrightarrow \quad z'_i = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} \;\; \text{and} \;\; \exists k' \in \mathcal{K} : G(k') = z'$$
2. If finding the set $\mathcal{K}'$ of consistent keys is possible, it can be reconstructed from $z_{i_1}, \ldots, z_{i_s}$ directly. Thus, the set $\mathcal{K}'$ of consistent keys can be redefined by
$$k' \in \mathcal{K}' \quad \Leftrightarrow \quad G_i(k') = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} ,$$
where $G_i(k)$ denotes the i-th output bit of generator G under key k.

1.4 Linear feedback shift registers

Sequences from linear recurrences: Remember that the sequence of inner states $(S_0, S_1, \ldots)$ is defined recursively via $S_0 = k, S_i = f(S_{i-1})$. It would be desireable to choose f such that the least period of the sequence $(S_0, S_1, \ldots)$ is 2^l, i.e. that $S_{2^l} = S_0$ and $S_j \neq S_0$ for $0 < j < 2^l$.

A class of recursions that is particularly well understood are *linear recursions*. A linear recursion is defined by a matrix M via
$$S_i = M \cdot S_{i-1}.$$
Note that no linear recursion can iterate through all 2^l possible states, since for all M, it holds that $M \cdot \mathbf{0} = \mathbf{0}$, where $\mathbf{0}$ is the all-zero vector. On the other hand, if the key $\mathbf{0}$ is disallowed, it is possible to construct linear recurrences that iterate through all of the remaining $2^l - 1$ inner states.

Linear feedback shift registers (LFSR): Let $S_i = (s^i_0, \ldots, s^i_{l-1})$ for arbitrary i. Consider a special kind of linear recursion, as defined by the following matrix operation:

$$\begin{pmatrix} s^i_0 \\ s^i_1 \\ \vdots \\ s^i_{l-2} \\ s^i_{l-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \ldots 0 \\ 0 & 0 & 1 & \ldots 0 \\ & \vdots & & \ddots \vdots \\ 0 & 0 & 0 & \ldots 1 \\ a_0 & a_1 & a_2 & \ldots a_{l-1} \end{pmatrix} \cdot \begin{pmatrix} s^{i-1}_0 \\ s^{i-1}_1 \\ \vdots \\ s^{i-1}_{l-2} \\ s^{i-1}_{l-1} \end{pmatrix}$$

A more intuitive description of the recursion is as follows:

$$s^i_j = \begin{cases} s^{i-1}_{j+1} & \text{if } 0 \leq j < l-1 \\ \sum_{k=0}^{l-1} a_k s^{i-1}_k & \text{if } j = l-1 \end{cases}$$

This means that the bits of the inner state are shifted to the left, as displayed in figure 4, with the leftmost bit being discarded and the rightmost bit being replaced by a linear combination of the previous inner state bits. Computation of n keystream bits takes $O(l \cdot n)$ computational steps and is easily parallelised in hardware. The overall construction is denoted as *linear feedback shift register* (LFSR).

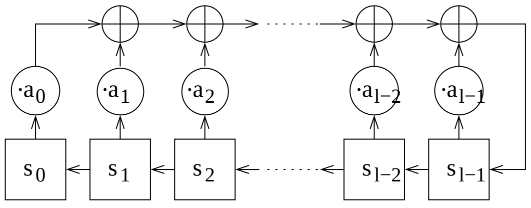

Fig. 4. Linear Feedback Shift Register

LFSR and m-sequences: Linear feedback shift registers are mathematically well understood. In particular, the *feedback vector* $(a_0, a_1, \ldots, a_{l-1})$ can be chosen in such a way that the sequence $(S_0, S_1, \ldots)$ iterates through all $2^l - 1$ possible inner states[1]. This makes maximum period LFSR good building blocks for pseudorandom generators.

Consider the immediate use of such an LFSR as pseudorandom generator, creating the output sequence via $z_i = g(S_i) = s^i_k$ for a fixed $k, 0 \leq k \leq l-1$. It can be shown that the resulting sequence satisfies Golomb's criteria [24], which are defined as follows:

– The output sequence has the same period as the inner states, i.e. $2^l - 1$.

[1] For a proof and further details on LFSR, cf. [24].

– Fix an arbitray integer r, $1 \leq r \leq l$, and consider a full period of output bits. Then every bit pattern of length r occurs exactly 2^{l-r} times, with the exception of 0^r, which occurs $2^{l-r} - 1$ times. We say that the sequence has *ideal statistics*.
– Consider one full period of the output sequence and shift it cyclically by r positions, $1 \leq r < 2^l - 1$. Then the Hamming distance between the original sequence and its shifted versions is $2^{l-1} - 1$ for all shifts r.

A sequence meeting the above requirements is sometimes denoted as an *m-sequence*, and the generating LFSR is called *m-LFSR*.

Cryptographic limitations Notwithstanding the good statistical properties, m-sequences do not make good keystreams. Note that the dependency between the output bits and the inner state S_0 can be modelled by a system of linear equations, implying the following attacks:

– If the attacker knows the feedback vector $(a_0, \ldots, a_{l-1})$, he can reconstruct the seed S_0 from l arbitrary keystream bits by solving a system of linear equations. This can be done in $O(l^3)$ computational steps (compared to $O(l^2)$ steps for the generation of l bits) and is feasible for any realistic parameter l.
– If the feedback vector is unknown, the seed S_0 can be reconstructed given $2l$ consecutive keystream bits, solving a system of $2l$ linear equations. Thus, this attack requires $O(l^3)$ computational steps too, being slower than the attack with known feedback only by a small constant factor.

Concluding, m-LFSR can be a useful building block for PRG, but some further work is required to prevent attacks that make use of the inherent linearity.

1.5 Introducing nonlinearity

If the update function of a PRG is modelled by an m-LFSR, nonlinearity has to be introduced into the keystream. In cryptographic literature and practice, there are a number of standard techniques that can be used to transform an m-sequence into a highly nonlinear output sequence. Note that all techniques presented below can be combined in the construction of a PRG.

Nonlinear filtering: The most obvious construction uses an m-LFSR to model the update function f, i.e. $f(S) = M \cdot S$ for an LFSR-type matrix M. In this case, the only possibility to introduce nonlinearity into the keystream is the use of a nonlinear output function g. Such a generator is denoted as *filtering generator* [44].

Nonlinear combination: A similar approach is the use of two or more m-LFSR with pairwise differing lengths and feedback vectors. In this design, the output function g uses part of the inner states of all LFSR in order to generate the output. Such a generator is called *combination generator* [44].

Nonlinear update: Filtering and combination generators have strictly linear inner states; nonlinearity is introduced using the output function g. It is, however, also possible to add nonlinearity to the inner states without sacrificing the advantages of m-LFSR. In this case, memory is partitioned in l_1 linear and l_2 nonlinear bits, with $l_1 + l_2 = l$. There are two update functions, where $f_1 : \{0,1\}^{l_1} \rightarrow \{0,1\}^{l_1}$ is an LFSR-type matrix, while $f_2 : \{0,1\}^{l} \rightarrow \{0,1\}^{l_2}$ is a suitably chosen nonlinear function. Note that in order for the keystream to be nonlinear, the output function must use at least some of the nonlinear bits. For historical reasons, PRG of this type are denoted as *generators with memory*.

Irregular clocking: Another method of introducing nonlinearity directly into the inner state is irregular clocking. A clock control function $c : \{0,1\}^{l} \rightarrow \mathbb{Z}$ uses part of the inner state to determine, how often the update function is applied before the next valid inner state is reached. More formally:

$$c_i = c(S_{i-1}), \qquad S_i = f^{c_i}(S_{i-1}).$$

Note that in some cases, c_i may be negative. Surprisingly, even very simple clock-control designs lead to strongly nonlinear inner state recurrences, making this technique a powerful tool in PRG design. The general class of PRG using irregular clocking is denoted as *clock control generators*.

2 Generic Attacks

2.1 Introduction

Two-step security analysis: In section 1, the attacker was defined as operating in the empirical security model. In order to provide security against such an attacker, the designer of a pseudorandom generator has to provide two kinds of analysis:

1. In a first step, security of the generator against previously known attacks has to be tested. In order to do so, the designer has to be aware of known attack techniques against pseudorandom generators. Only if none of these techniques can be applied successfully to the new generator, the second phase is entered.
2. In the second phase, the designer has to search for new attacks against his specific generator. Since this task is much more difficult than the application of existing attacks, the designer is well advised to get the help of as many experts as he can find. This is true even if the designer himself is an expert, on the simple grounds that four eyes see more than two.

Note that the diligent and successful completion of both analysis phases does not provide a security guarantee. Resistance against attacks both old and new is a necessary, but not a sufficient criterion for security.

Generic vs. specific attacks: The first part of this thesis will provide a survey of existing attack techniques against pseudorandom generators. In this context, we distinguish two broad classes of attacks:

– *Generic attacks* are applicable even if the attacker does not know the design of the generator. Most generic attacks date back to the beginnings of public cryptographic research, and for many years, the security of pseudorandom generators was measured against them. Generic attacks will be discussed in the current section.
– In contrast, in order to apply *specific attacks*, the attacker has to know the internal build of the generator. Specific attacks are more recent than the generic ones, and some of them can only be directed against specific generator designs. They will be discussed in sections 3 to 4. Furthermore, the novel attack techniques presented in part two of this thesis fall into this category as well.

2.2 Statistical testing

First and foremost, the attacker must not be able to observe any regularities in the keystream. If this was the case, he could predict additional bits of the keystream sequence, yielding an attack on the generator. For this reason, it must not be possible to tell the keystream apart from a truly random sequence. This concept is formalised by the notion of statistical hypothesis testing.

Hypothesis testing: Let $z \in \{0,1\}^s$ be a bitstring that is either random (hypothesis H_0) or pseudo-random (hypothesis H_1). Further, let $X : \{0,1\}^s \to \mathbb{R}$ be a random variable that can be efficiently computed from z. Then we denote the probability distribution of $X(z)$ by D_0 if z was drawn according to H_0, and by D_1 otherwise.

Given an observation x for the random variable $X(z)$, the attacker's goal is to decide whether x was drawn according to distribution D_0 or D_1. Note that this is only possible if the distributions D_0 and D_1 differ. This difference is measured by the *statistical distance* between D_0 and D_1, which can be defined as

$$|D_0 - D_1| = \sum_x |D_0(x) - D_1(x)| \ .$$

The larger the statistical distance is, the weaker is the pseudorandom generator. For a distinguishing attack, a *decision rule* $R : \mathbb{R} \to \{0,1\}$ is used to decide whether a given x was drawn according to D_0 or D_1. The quality of such a decision rule is measured by the advantage

$$\operatorname{adv}(R) = \frac{1}{2}\left(\Pr_{x \leftarrow D_0}(R(x) = 0) - \Pr_{x \leftarrow D_1}(R(x) = 0)\right) \ .$$

The decision rule that achieves the greatest advantage is the maximum likelihood rule, which outputs 0 iff $D_0(x) > D_1(x)$. The advantage achieved by this rule is $\frac{1}{4} \cdot |D_1 - D_2|$. Note, however, that the distributions $D_0(x)$ and $D_1(x)$ must be known in order to implement this rule, which is not always the case.

Sample randomness tests: A *statistical test* is a combination of a suitable random variable and decision rule, where both must be efficiently computable. Several statistical tests have been proposed that should be passed by every pseudo-random generator. As an example, the following tests due to Beker and Piper [BePi82] formed the basis for the FIPS 140-2 statistical tests of randomness [NIST02], until the specification of concrete test procedures was removed from the standard in december 2002.

- *Poker test:* For values $m \ll s$, a random bitstring is expected to contain all possible bit patterns i of length m equally often. Let $k = \lfloor \frac{s}{m} \rfloor$ and divide the bitstring z into k non-overlapping parts of length m. If we denote the number of occurrences of i by s_i, the random variable

$$X = \frac{2^m}{k} \left(\sum_{i=(0..0)}^{(1..1)} s_i^2 \right) - k \ ,$$

  follows a χ^2 distribution[2] with $2^m - 1$ degrees of freedom for random sequences.
  A special case is $m = 1$, which tests whether the number of zeroes roughly equals the number of ones in the sequence. With $m = 1$ and $k = s$, the random variable simplifies to

$$X = \frac{(s_0 - s_1)^2}{s}$$

  and follows a χ^2 distribution with 1 degree of freedom. This case is also known as *frequency test* or *monobit test*.
- *Runs test:* A *run* is a maximum-length substring that consists only of zeroes (*gap*) or ones (*block*). Denote by G_i and B_i the number of gaps and blocks of length i, resp. Then for $k \ll s$ and $1 \leq i \leq k$, a random sequence should have the property that G_i and B_i should be close to the expected value, which is $e_i = (s - i + 3)/2^{i+2}$. Thus, the random variable

$$X = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

  should follow a χ^2 distribution with $2k - 2$ degrees of freedom for random bitstrings.
- *Autocorrelation test:* A random sequence should bear no similarity to its shifted versions. For a given shift d, $1 \leq d \leq \lfloor \frac{n}{2} \rfloor$, this is measured by the hamming distance

$$D_d = \sum_{i=0}^{s-d-1} z_i \oplus z_{i+d} \ .$$

[2] Details on the χ^2 distribution and its use for testing purposes can be found in any textbook on statistics.

For random bitstrings, D_d is $(\frac{s-d}{2}, \frac{s-d}{4})$ normal distributed, implying that

$$X = 2\left(D_d - \frac{s-d}{2}\right)/\sqrt{s-d}$$

follows a standard normal distribution.

Other tests have been proposed by Golomb [24], Beker and Piper [1], Knuth [30], Maurer [32], and many others. It is important to remember that all of these tests are necessary, but by no means sufficient criteria for good keystream sequences.

2.3 Period and linear complexity

Remember that a pseudorandom generator is a finite state machine with at most 2^l inner states. Thus, a keystream sequence generated by such a generator must become cyclic after at most 2^l output bits. As a consequence, the more significant bits of the sequence can be modelled as a function of the less significant bits by a suitable recurrence relation.

Let R be a recurrence relation with $x_i = R(x_{i-1}, \ldots, x_{i-k})$. Then we denote k as the *length* of R. If a keystream sequence can be described by such a relation and if the attacker has at least k consecutive keystream bits at his disposal, he can easily predict the full keystream sequence. In the following, two particularly simple types of recurrences will be discussed.

Period: Consider an infinite bitstream $z = (z_0, z_1, \ldots)$. If there exist values $\rho, \theta \in \mathbb{N}$ such that $z_i = z_{i+\rho}$ for all $i \geq \theta$, the sequence is said to be ρ-periodic with pre-period θ. The smallest value ρ for which z is ρ-periodic is called the *least period* or simply *period* of z.

Since pseudorandom generators have at most 2^l inner states, it holds that $\theta + \rho \leq 2^l$. Since for all $i \geq \theta + \rho$, the attacker can use the recurrence $z_i = z_{i-\rho}$ to predict additional bits of the keystream, it is paramount that at most $\theta + \rho$ bits of keystream are generated with the same key. Thus, all sequences generated by a pseudorandom generator should have large periods.

Linear complexity In some cases, the periodic part of the sequence z can be described by a linear recurrence relation R such that $k < \rho$. The length k of the shortest such linear recurrence is denoted as *linear complexity*[3] $\mathrm{LC}(z)$. Put another way, the linear complexity is the length of the smallest LFSR that generates the sequence z. Note that the period recurrence itself is a linear recurrence, such that $\mathrm{LC}(z) \leq \rho$.

There exists an efficient algorithm by Berlekamp and Massey [31] that constructs the shortest linear recurrence describing z. Since this algorithm needs only $2 \cdot \mathrm{LC}$ keystream bits and takes only $O(\mathrm{LC}^2)$ computational steps, an attacker can easily simulate a keystream sequence with a low linear complexity by a linear relation. Thus, high linear complexity is a necessary requirement for all sequences generated by a pseudorandom generator.

[3] Sometimes also the term *linear equivalence* is used.

3 Specific Attacks

3.1 Two sample generators

As discussed in section 2, in order to apply a specific attack to a generator, its inner workings have to be known. This, however, does not mean that the wheel has to be re-invented for every generator. A number of attack techniques are known that can be used against a PRG, implying that resistance against such attacks is a minimal requirement of any such generator.

In order to illustrate the workings of these attacks, two simple pseudorandom generators will be considered in the next to sections: The Geffe generator and the $\{1, 2\}$-clocked generator. For both generators, the underlying LFSR can be chosen such that resistance against the generic attacks presented in section 2 is provided.

Throughout the descriptions, LFSR are denoted by capital letters. The length of LFSR X is denoted as l_X, and the output sequence generated by X is $x = (x_0, x_1, \ldots)$.

Geffe generator: The Geffe generator was introduced in [13]. It is a nonlinear combination generator, as discussed in subsection 1.5. It consists of three m-LFSR A, B and C, producing m-sequences a, b and c. Keystream bit z_i is generated using the Boolean function

$$\begin{aligned} z_i &= (c_i \wedge a_i) \vee (\overline{c_i} \wedge b_i) \\ &= c_i \cdot a_i \oplus c_i \cdot b_i \oplus b_i \ . \end{aligned}$$

This means that $z_i = a_i$ if $c_i = 1$, and $z_i = b_i$ otherwise. Thus, c_i is denoted as control bit and register C as control register. For an illustration of the generator, see figure 5.

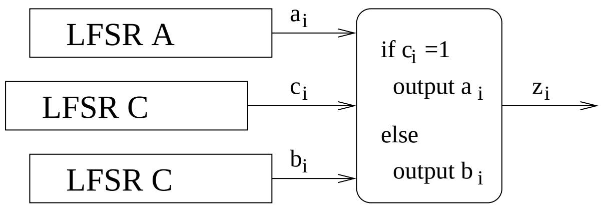

Fig. 5. Geffe Generator

$\{1, 2\}$*-clocked generator:* This generator is a special case of the basic clock-controlled shift register arrangement proposed by Gollmann and Chambers in [23]. It consists of two m-LFSR A and C, where C is called the control register of the arrangement. If $c_i = 0$, register A is clocked by one step; otherwise, A

is clocked by two steps. Thus, the generator is sometimes named *step1-step2 generator*. The output is taken directly from register A, meaning that $z_i = a_{s(i)}$, where $s(i) = \sum_{k=0}^{i}(c_i + 1) = (\sum_{k=0}^{i} c_i) + i + 1$. An alternative description is as follows: Given the inner bitstream a and a pointer to the last used bit, the next keystream bit z_i is obtained by deleting c_i bits from a and using the next available bit as output. An illustration of the generator is given in figure 6.

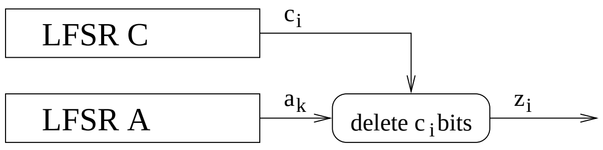

Fig. 6. $\{1, 2\}$-Clocked Generator

3.2 Guessing attacks

Brute force search: In the context of pseudorandom generators, a brute force search is conducted by guessing all l bit of the inner state. For each of his guesses, the attacker runs the generator to generate l output bits and compares the result with the known keystream. If they differ in at least one bit, the guess is discarded as being wrong. Otherwise, the guess is added to the set of *key candidates*. For a generator that passes the standard statistical tests (see section 2.2), the number of false guesses in this set should be close to zero. If there exists more than one key candidate, the correct one is determined by running the generator to decrypt all the message.

Remember from section 1.2 that the attacker is not able to conduct a complete brute force attack. Thus, valid attacks in our security model must use strictly less computational steps than are necessary to run the generator 2^l times. One option for such an attack is as follows: The attacker conducts a brute force attack over *part* of the inner state, tries to derive as much information as possible, and carries on. The following are examples for such attacks.

Guess-and-verify: First, the attacker guesses $l' < l$ bit of the inner state. If it is possible to efficiently discard a fraction q of these guesses as being wrong ($0 < q < 1$), the attacker is left with only $(1-q) \cdot 2^{l'}$ candidate partial guesses. If for each such candidate, he does a complete search over the remaining $l - l'$ bit, he ends up with a total running time of $2^{l'} + (1-q) \cdot 2^{l'} \cdot 2^{l-l'} = 2^{l'} + (1-q) \cdot 2^{l}$ computational steps. If $q > 2^{l'-l}$, the computational effort for this attack is strictly less than 2^l. To illustrate, consider the following attack on the Geffe generator:

– *Geffe generator:* The attacker guesses the inner states of registers A and B. Thus, he can construct the complete sequences a and b. Whenever $a_i =$

$b_i \neq z_i$ for a given index i, he has identified a wrong guess. Given enough keystream (slightly more than $4 \cdot (l_A + l_B)$ bit), he can uniquely identify the correct seed for A and B amongst the guesses. It only remains to do a brute force search over C, yielding a total running time of $2^{l_A + l_B} + 2^{l_C}$. If $l_A = l_B = l_C = l/3$, the attack takes roughly $2^{2l/3}$ computational steps.

Guess-and-determine: Here, too, the attacker guesses $l' < l$ bit of the seed. Instead of verifying directly, however, he tries to construct additional parts of the inner state from the keystream. Only after doing so, the brute force search is completed by guessing the missing part of the seed. If on average, the attacker derives m bit from the keystream, this attack takes roughly $2^{l'} \cdot 2^{l-l'-m} = 2^{l-m}$ computational steps. As an example, consider the Geffe and $\{1,2\}$-clocked generators:

- *Geffe generator:* The attacker guesses the complete inner state of register C. Now, however, he does know which output bit z_i stems from which register A or B. As long as these bits are part of the seed, the attacker does not have to guess this part anymore. Assuming that $l_A = l_B = l_C = l/3$, the attacker obtains an average of $l/3$ keystream bits that can be written directly into registers A and B. Thus, the overall work effort of the attack is only $2^{l-l/3} = 2^{2l/3}$.
- $\{1,2\}$*-clocked generator:* Here, too, the attacker guesses the complete inner state of register C. This way, for each bit z_i, he can determine the corresponding position k in the inner bitstream $a = (a_0, a_1, \ldots)$. Note that on average, 2 out of 3 seed bits of register A can be read from the keystream. Thus, the attacker has a total effort of $2^{l-2l_A/3}$ guesses. If $l_A = l_C = l/2$, this yields a computational effort of $2^{2l/3}$ generator runs.

Linear consistency test The linear consistency test (LCT) was proposed by Zeng, Yang and Rao [56]. It can be considered as a combination of the above guessing attacks, making use of the linearity of the inner bitstreams. To carry out the test, the attacker guesses $l' < l$ bit of the inner state in such a way that the relationship between the remaining bits and the output bits can be modelled by a system of linear equations. If this equation system is contradictory, the initial guess is discarded. If the equation system has maximum rank, it can be solved, yielding the unknown bits of the key candidate. On the other hand, if some linear equations are linearly dependend on the others, it is usually possible to construct additional equations until the system has full rank and can be solved. Remembering that solving a linear equation system in n unknowns requires $O(n^3)$ computational steps using Gauss' elimination algorithm, it follows that the running time of this attack is in $O(2^{l'} \cdot (l - l')^3)$. For illustration, consider the following examples:

- *Geffe generator:* As in the case of a guess-and-determine attack, the attacker guesses the full inner state of register C. Thus, for each output bit z_i, he knows whether it stems from the inner bitstream a or b. In a first step, consider l_A output bits that stem from register A, i.e. $a_i = z_i$. Now note that each bit a_i can be written as linear combination of the initial state $(a_0, \ldots, a_{l_A-1})$, yielding one linear equation. Given slightly more than l_A such output bits, the resulting equation system is either contradictory or has full rank with high probability. Thus, the attacker can either discard his guess for C, or he can construct the complete inner state of register A. In a second step, he proceeds analogously for register B. Thus, the overall running time of the attack is in $O(({l_A}^3 + {l_B}^3) \cdot 2^{l_C})$.

– $\{1, 2\}$*-clocked generator:* Analogously, the attacker guesses the full inner state of register C. For each output bit z_i, this yields an index j such that $z_i = a_j$. Again, if all a_j are seen as linear combinations of the initial state $(a_0, \ldots, a_{l_A-1})$, slightly more than l_A output bits should suffice to construct a linear equation system that is either contradictory or can be solved uniquely. The running time of this attack is in $O({l_A}^3 \cdot 2^{l_C})$.

Extensions: Depending on the generator at hand, the attacks presented in this section can sometimes be refined by using a backtracking approach. As an example, in the guess-and-verify scenario, the attacker guesses a limited number of bits, verifies, then guesses more bits, verifies again and so on. In the case of the Geffe generator, he would guess pairs (a_i, b_i) and immediately verify whether $a_i = b_i \neq z_i$ before guessing more bits. Since for one in four guesses, this condition is violated, he can reduce the effort for reconstruction of A and B to $\left(\frac{3}{4} \cdot 2\right)^{l_A+l_B} = 2^{0.58(l_A+l_B)}$ steps. Such backtracking improvements are also applicable for the linear consistency test, with examples being given in [17, 59, 58].

3.3 Algebraic attacks

Preliminaries: While the linear consistency test uses linear equations to reconstruct the seed of a pseudorandom generator, algebraic attacks use *nonlinear equations*. A nonlinear equation in $x_1, \ldots, x_l$ is of the form

$$\sum_{i=1}^{n} M_i = c \ ,$$

where $c \in \{0, 1\}$ is a constant, n is a positive integer and the M_i are monomials of the form

$$M_i = \prod_{j=1}^{l} {x_j}^{c(i,j)} \ , \qquad c(i,j) \in \{0, 1\} \ .$$

The degree d_i of a monomial M_i is defined as $d_i = \sum_{j=1}^{l} c(i,j)$. The degree of an equation is the maximum over all monomial degrees d_i in the equation. Analogously, the degree of an equation system is the maximum over all monomial degrees in the equation system.

In principle, nonlinear equations can be used to describe each output bit as a nonlinear combination of the generator's seed. There are, however, two problems associated with this approach:

– While a linear equation in variables $x_1, \ldots, x_l$ can have at most l monomials, a nonlinear equation of degree d has up to

$$N_d := \sum_{k=1}^{d} \binom{l}{k} \in O(l^d)$$

monomials. In the worst case, for $d = l$, up to 2^l monomials can occur in one single equation. Thus, working with nonlinear equations can only be efficient if the number of monomials in each equations is not too large.

– Even worse, solving systems of nonlinear equations is known to be NP-hard [12]. Thus, we can not expect to find an algorithm that efficiently solves all nonlinear equation systems. On the other hand, finding such a universal algorithm is not the attacker's goal anyway. In our model, he is successful if he can solve a significant part of the equation systems that occur in this special context.

The linearisation technique: Assume that the attacker has a system of nonlinear equations at his disposal, where each equation describes the dependency between one output bit and the corresponding seed bits. He could try to solve this system by *linearisation*, replacing each nonlinear monomial by a single dummy variable. This way, he ends up with a linear equation system which can be solved using standard techniques like the Gaussian eliminiation algorithm. Finally, the dummy variables have to be replaced by the original monomials again, in the hope that a unique solution (or at least a small set of solution candidates) can be identified.

Note that during linearisation, the number of variables in the equation system increases dramatically. Thus, the attacker needs up to N_d linearly independent linearised equations, i.e. the number of required keystream bits can be very large. At the same time, the attacker is throwing away valuable information contained in the monomials of high degree. For an example, consider the information loss when replacing the nonlinear equation $x_1x_2x_3x_4 = 1$ by the linearised equation $M_1 = 1$.

The extension technique: An important improvement over mere linearisation is the extension technique proposed by Kipnis and Shamir [29]. Given a system of nonlinear equations, the attacker constructs additional equations by multiplying the existing ones with monomials of small degree. If the degree of the resulting equation is not greater than a specified threshold, the equation is added to the equation system. This way, a nonlinear system with a lot of redundant information is generated.

In a second step, the extended equation system is linearised as described above. This time, however, the attacker has a lot more equations at his disposal, reducing the number of output bits required. Still, for the attack to work, the original equation system has to be over-specified. In [8], some evidence (though no formal proof) is given that the attack requires more than l output bits, but that the number of additional bits is small.

Sample attack: Remember that for the *Geffe generator*, an output bit z_i can be described by the nonlinear equation

$$z_i = c_i \cdot a_i \oplus c_i \cdot b_i \oplus b_i \ . \tag{2}$$

Thus, each output bit contributes one equation of degree 2 to the equation system. If the conjecture by Kipnis and Shamir is correct, it would suffice to collect slightly more than $N_2 = \frac{l^2+l}{2}$ output bits in order to build a solvable equation system of this basic type. Using the Gaussian elimination algorithm, the computational effort for such an attack would be in the order of $O({N_2}^3) = O(l^6)$, yielding a polynomial time attack on the Geffe generator.

However, using the extension technique, the number of output bits required can be reduced even further. As an example, multiplying equation (2) with c_i, $a_i \cdot b_i$, $c_i \cdot a_i$ (resp.) yields the new equations

$$\begin{aligned} 0 &= c_i \cdot a_i \oplus z_i \cdot c_i \ , \\ 0 &= a_i \cdot b_i \oplus z_i \cdot a_i \cdot b_i \ , \\ 0 &= c_i \cdot a_i \oplus z_i \cdot c_i \cdot a_i \ , \end{aligned}$$

all of which also have degree 2 once z_i is replaced by a bit value from the output stream. Now, each output bit contributes 4 equations to the equation system, reducing the overall number of known output bits needed to perform the attack.

Concluding remarks: The attack presented above is the most simple form of algebraic attack. It is sometimes denoted as *XL attack*, from its components e$\underline{\text{X}}$tension and $\underline{\text{L}}$inearisation. Note, however, that more efficient variations exist, although it seems hard to find precise estimates for the required running time or keystream bits.

All aspects of algebraic attacks are currently a very active field of research. Research topics include how to find nonlinear equations of low degree, how to solve these equations as efficiently as possible, how to estimate the resources required by the algorithms, how to apply algebraic attacks against specific ciphers, and others.

3.4 Time-Memory-Data tradeoffs

By definition in section 1.2, the attacker is not able to conduct a brute force search over the key space. He may, however, attempt a search over a small part of the key space, hoping that the produced output string is observed in the known keystream. However, if the available keystream is small (say, little more than l bit), his probability of success is negligibly small. The picture changes, though, if a sufficiently long string of keystream bits is available. In this case, the attacker can make use of a time-memory-date tradeoff attack, with the probability of finding a pre-computed value amongst the observed keystream being close to one.

The birthday problem: Such collision-based attack techniques are surprisingly successful not only in stream cipher cryptanalysis, but also in attacking other cryptographic primitives like block ciphers or hash functions. The mathematical reason for this success lies in a number of results from probability theory that have been termed *birthday problem*. The cryptographically most relevant instances of the birthday problem are defined as follows:

1. *Collision within one set:* Let an urn contain M balls numbered 1 to M. One ball is drawn at a time, with replacement, the number is written down. What is the expected number N of draws until the first collision occurs, i.e. the same ball is drawn for the second time?
2. *Collisions between two sets:* Let an urn contain M balls numbered 1 to M. First, a set of N_1 balls is drawn without replacement, the numbers are written down. Then the balls are placed back into the urn. Now balls are drawn, with replacement, and the number is compared to the numbers of the list. What is the expected number N_2 of draws before a collision with the list occurs?

Since exact collision probabilities for the birthday problem are difficult to handle in practice, asymptotic estimates are being used for cryptographic purposes. In the case of a collision within one set, the expected number of necessary draws can be approximated as $N \approx \sqrt{M}$ for large values of M. For collisions between two sets, the estimate $N_2 \approx M/N_1$ is used.

Basic Time-Memory tradeoff attack: In the *precomputation phase*, the attacker selects N_1 different keys at random, and for each key computes the first l output bits produced by the generator. The resulting tupel of output string and key is saved in a hash table, indexed by the output string. During *realtime phase*, the attacker is given $N_2 + l - 1$ bits of generator output. From this, he generates N_2 overlapping output strings of length l. Each string is looked up in the hash table, and if a match is found, the corresponding key can be read directly from the table. Note that if $N_1 \cdot N_2 > 2^l$, the attacker is likely to succeed using this method.

The computational effort during precomputation is determined by running the generator N_1 times and storing $2l \cdot N_1$ bits in a hash table. In the realtime phase, an expected N_2 table lookups generate the main bulk of work. Thus, the overall effort is roughly $N_1 + N_2$ computational steps. In the best case, $N_1 \approx N_2 \approx 2^{l/2}$, allowing the attacker to break the system in roughly $2^{l/2+1}$ computational steps, using $2l \cdot 2^{l/2}$ bits of memory.

Improvements: In the basic time-memory tradeoff attack, both time and memory requirements for the pre-computation phase are in the order of N_2. In practice, however, computation time is considerably cheaper than memory. This problem is solved by the *time-memory-data* tradeoff [3], which allows for a more sophisticated choice of the attack parameters. Using this technique, the parameters T (realtime computation time), P (pre-processing computation time), D (number of known keystream bits), and M (number of memory bits available) can be chosen in any way, as long as the conditions $P = 2^l/N$, $D^2 \leq T \leq 2^l$, and $TM^2D^2 = 2^{2l}$ are satisfied.

As another problem arising in practice, realtime computation time is determined by the number of table lookups. Since the table of samples is very large, it must be stored on hard disk, and disk access is slower than RAM access by a factor of about 4 million (or 2^{22}). While in theory, this constant factor is often neglected, it slows down a practical attack considerably. A solution to this problem is *sampling*, where only states that generate certain output patterns are stored on disk. As a consequence, only those output strings that display this pattern have to be looked up, keeping the overall computational time constant, but reducing the number of disk accesses.

4 Correlation Attacks

4.1 Basic correlation attack

Consider the Geffe generator or any other generator that is based on a number of LFSR and a nonlinear combining function g. While g must be balanced for all generators that pass the statistical tests given in section 2.2, there is a more subtle danger. For some choices of g, a correlation between an input bit a and the corresponding output bit z can be observed. As an example, consider the combining function

$$g(a, b, c) = (c \wedge a) \vee (\bar{c} \wedge b)$$

of the Geffe generator. While the output is balanced, the probability that $z = g(a, b, c) = a$ equals $3/4$.

An analogy to coding theory: For any combination generator and any input bit a to the combining function g, the relation between a and output bit z can be modelled in a coding theoretic setting as a noisy channel,[4] as shown in figure 7. Each output bit z can be seen as a noisy version of the input bit a, i.e.

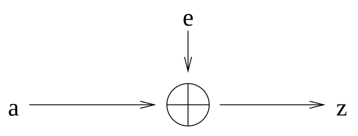

Fig. 7. A noisy channel

z can be modelled as $z = a \oplus e$ for some noise bit e with $\Pr(e = 1) =: p_e$. Let $z_0, z_1, \ldots, z_{n-1}$ be the known output bits. It is known that the corresponding vector $(a_0, a_1, \ldots, a_{n-1})$ was generated by an LFSR of length l_A. Thus,

[4] For an introduction to the theory of linear codes, see any textbook on coding theory, e.g. [41],[50].

$(a_0, a_1, \ldots, a_{n-1})$ can be considered as a codeword in a linear code, with l_A information bits and $n - l_A$ checking bits. The problem of reconstructing the contents of register A is equivalent to finding the codeword $(a_0, a_1, \ldots, a_{n-1})$ with the least Hamming distance to the known output word $(z_0, z_1, \ldots, z_{n-1})$.

However, the general problem of reconstructing the nearest codeword in an arbitrary linear code is NP-hard [2]. In coding theory, the problem was solved by deliberately choosing the linear code in such a way that decoding is easy. A surprising consequence was that when the duality of coding theory and cryptanalysis was discovered by Siegenthaler in 1984 [48, 49], no generic algorithms for the decoding of arbitray linear codes existed. Ever since, cryptographers have developed algorithms for the decoding problem, enabling increasingly powerful correlation attacks.

Siegenthaler's attack: The first algorithm for correlation attacks was proposed by Siegenthaler [49]. Consider a combining function g and input variable a such that $p_e := \Pr(a \neq z) < 1/2$. Given output bits $z_0, \ldots, z_{n-1}$, the attacker proceeds as follows. He guesses the complete contents of register A and generates n bits of internal bitstream $a_0, \ldots, a_{n-1}$. Then, he computes the Hamming distance

$$D_a = \sum_{i=0}^{n-1} (a_i \oplus z_i) \ ,$$

where $\oplus$ denotes bitwise addition over GF(2), while the overall sum is computed over the integers.

Note that the distribution of D_a differs, depending on whether the guess for A is correct or not. If the guess was right, D_a is binomially distributed with expected value $\mu = n \cdot p_e$ and variance $\sigma^2 = n \cdot p_e \cdot (1 - p_e)$. If it was wrong, however, the vector $(a_0, \ldots, a_{n-1})$ behaves like a random n-bit string, and we have $\mu = n/2$ and $\sigma^2 = n/4$.

These differing distributions yield a statistical test on our guess for A. Depending on the values for n and p_e, the attacker will set a threshold D' such that a guess is accepted as a partial key candidate if $D > D'$. Note that the distinguishing power of this test increases with growing n and $|p_e - 1/2|$. In the best case, exactly one candidate guess for A will be derived, immediately yielding the correct contents of register A. Otherwise, several candidate guesses remain, making additional tests have necessary in order to identify the correct one. Nonetheless, if n and $|p_e - 1/2|$ are large enough, the number of steps required to retrieve the contents of register A do not significantly differ from 2^{l_A}.

Observe that Siegenthaler's technique is a special case of a guess-and-verify attack, as described in section 3.2. However, in the case of the Geffe generator, the overall running time of a correlation attack is only $2^{l_A} + 2^{l_B} + 2^{l_C}$ (as opposed to $2^{l_A + l_B} + 2^{l_C}$ steps that are required for the simple attack presented in section 3.2).

Extensions and limitations: The above attack technique can be extended to correlations between the output z and linear combinations of input bits $x^{(i)}$. If the generator consists of k LFSRs $X^{(1)}, \dots, X^{(k)}$ and there exists an index set $I \subset [k]$ such that

$$\Pr\left(\bigoplus_{i \in I} x^{(i)} \neq z\right) < \frac{1}{2} \ , \tag{3}$$

then the attacker can guess the initial states of all registers $X^{(i)}$ with $i \in I$. Next, he generates the first n bits generated by each such register and calculates $x_j = \bigoplus_{i \in I} x_j^{(i)}$ for $j = 0, \dots, n-1$. Computing the Hamming distance between $(x_0, \dots, x_{n-1})$ and $(z_0, \dots, z_{n-1})$, the attack proceeds as above. Note, however, that the computational effort for this phase has gone up to 2^{λ} steps, where $\lambda = \sum_{i \in I} l_{X^{(i)}}$.

An obvious protection against correlation attacks that guess at most r registers ($1 \leq r < k$) is the choice of a combining function g that is correlation-immune of r-th order, i.e. no set $I \subset [k]$ with $|I| \leq r$ exists that meets condition (3). It is known, however, that a high correlation-immunity leads to a low linear complexity, and vice versa [45]. Thus, all practical combining generators with an acceptable linear complexity will be vulnerable against correlation attacks to some extend.

4.2 Fast correlation attacks

Underlying idea: Consider again the case of a combination generator where the output a_i of a single register A is correlated with the output z_i of the generator. The attack proposed by Siegenthaler basically requires a brute force search over all possible initial states of register A, yielding an effort of $2^{|A|}$ computational steps. In [33, 34], Meier and Staffelbach proposed to use techniques from coding theory [11] in order to speed up the reconstruction of register A.

First observe that each bit of the vector $a = (a_0, a_1, \dots, a_{n-1})$ produced by A is part of a number of linear relations. For example, if the simple feedback recurrence $a_i = a_{i-l_A} \oplus a_{i-l_A+1}$ is used, each bit a_k is contained in the three relations

$$\begin{aligned} a_k &= a_{k-l_a} \oplus a_{k-l_a+1} \\ a_{k+l_A} &= a_k \oplus a_{k+1} \\ a_{k+l_A-1} &= a_{k-1} \oplus a_k \ . \end{aligned}$$

Additional linear relations can be constructed, for example by addition of known ones. Note that the number of such relations grows in n, but that most of them will contain a large number of different variables.

Now remember that the output vector $z = (z_0, z_i, \dots, z_{n-1})$ can be seen as the intermediate vector $a = (a_0, a_1, \dots, a_{n-1})$, masked by an error vector $e = (e_0, e_1, \dots, e_{n-1})$ with $\Pr(e_i = 1) < 1/2$. The basic observation is as follows: If $e = \mathbf{0}$, then $z = a$ and z meets all linear relations that are fulfilled for a. If only one bit $e_k = 1$ in e, all equations containing z_k are contradictory, while all others are satisfied. But even if the Hamming weight of e increases, some linear relations in the z_i will be fulfilled. As a rule of thumb, we expect $z_k = a_k$ for a given k if the share of satisfied relations amongst those that contain z_k is large. On the other hand, we expect $z_k \neq a_k$ if the share of satisfied relations is small. This simple observation can be used for a variety of reconstruction algorithms for the inner state of A.

An exponential time algorithm: Note that in order to reconstruct a, it is sufficient to reconstruct l_A bits of a. The remaining bits can be computed using systems of linear equations. A simple algorithm proposed by Meier and Staffelbach [33, 34] proceeds as follows:

1. Construct a reference set of linear relations in the z_i of equal Hamming weight.
2. For each z_i, $i = 0, \dots, n-1$, compute the probability p^* that this bit is correct, given the number of linear relations it satisfies.
3. Choose l_A bits for a reference guess $\hat{a}$ by picking those z_i with the highest values p^*.
4. Find the correct guess by modifying $\hat{a}$ by $1, 2, \dots$ bit and constructing the full vector a. Compute the Hamming distance between a and z. If this distance is close to the expected value, output a and stop.

The average running time of this algorithm is determined by step 4, which takes about

$$N_d = \sum_{i=0}^{d} \binom{l_A}{d}$$

trials, with d being the expected number of wrong digits in the reference guess $\hat{a}$. Since this value is clearly smaller than l_A, reconstructing register A takes $2^{c \cdot l_A}$ steps with $c < 1$.

A polynomial time algorithm: A number of improvements over the above algorithm are possible. In particular, note that when correcting the guess $\hat{a}$ in step 4, all bits are treated equal. We may, however, assume that those bits that satisfy a large number of equations are more likely to be correct than those that satisfy less equations. Thus, a variant algorithm also proposed by Meier and Staffelbach [33, 34] proceeds as follows:

1. Construct a reference set of linear relations in the z_i of equal Hamming weight.
2. For each z_i, $i = 0, \dots, n-1$, compute the probability p^* that this bit is correct, given the number of linear relations it satisfies.
3. Negate those bits z_i whose probability p^* is under a certain threshold. If the resulting vector z does not satisfy all relations, go back to step 2.

Note that this description is a simplified version of the full algorithm. Nonetheless, in all cases the running time for step 1 is linear in the length of the registers. Step 2 and 3 are even independent of the register length, instead, the running time is determined by the Hamming weight of the linear relations used, by the error probability $\Pr(e_i = 1)$ and by the number n of output bits available. While no closed mathematical expression for the running time could be found, it was observed that for relations of small weight (up to 8), the attack was extremely fast in practice. As a consequence, the use of LFSR with a small number of feedback taps is strongly discouraged.

Improvements: Following the publication of [34], a number of improvements have been proposed. These can be subdivided into two categories:

- In step 1 of the above algorithm, linear relations of low degree have to be found. The efficiency of steps 2 and 3 can be increased if more care is spent on this preprocessing step. Proposals on how to find more or better linear relations were given, e.g., by Mihaljević and Golić [39], Chepyzhov and Smeets [6], and Penzhorn [40].
- In addition, the iterative decoding procedure in step 2 and 3 was improved by several proposals, such as the algorithms given by Zeng, Huang, Yang and Rao [55, 57], Mihaljević and Golić [39], Chepyzhov and Smeets [6] or Živković [53]. As opposed to the original algorithm, many of these proposals also contain a proof of their convergence.

However, all of these proposals are efficient only if the feedback vectors of the LFSRs under consideration have low weight. This limitation was done away with by a set of completely different algorithms to be discovered in subsequent years. Johansson and Jönsson use convolutional codes [26, 28], Turbo Codes [25] or algorithms from learning theory [27] in order to reconstruct the inner state. Canteaut and Trabbia [4] gave an algorithm to construct linear relations of low weight for arbitrary feedback vectors. Chepyzhov, Johansson and Smeets [5] approximate the LFSR output by a linear code of smaller dimension, but with higher error probability. A similar approach is chosen by Filiol [10], who proposes a d-decimating attack, considering only every d-th output bit of the LFSR.

Depending on the combination generator considered, the above attack techniques can be of varying efficiency. For the majority of generators, however, the most efficient algorithm to date is a combination of several of the above concepts, as proposed by Mihaljević, Fossorier and Imai [37, 38] and improved by Chose, Joux and Mitton [7].

4.3 Correlation attacks and memory

Correlation immunity: The efficiency of correlation attacks makes it necessary to harden combination generators against such attacks. The most obvious solution is to choose the combining function g in such a way that no correlations between the output and a linear combination of a small number of internal bits exist. More formally, a function $g : \{0,1\}^k \to \{0,1\}$ with input vector $x = (x^{(1)}, \ldots, x^{(k)})$ is said to be *correlation immune of k'-th order* if no linear combination L of up to $k' < k$ variables exist such that $\Pr(L(x) = g(x)) \neq 1/2$. The following tradeoffs, however, make it difficult to strengthen the generator in this way:

– It was shown by Siegenthaler, Xiao and Massey in [48, 54] that an increase in correlation immunity leads to a decrease in linear complexity, and vice versa. Thus, a highly correlation immune combination generator can be attacked using the Berlekamp-Massey-algorithm.
– Let $\{L_i \mid 1 \le i \le 2^k\}$ be the set of linear functions in up to k variables. The *correlation coefficient* between g and L_i is defined as $c_i = 2 \cdot p_i - 1$, with $p_i = \Pr(L_i(x) = g(x))$. It was proven by Meier and Staffelbach [35] that

$$\sum_{i=1}^{2^k} {c_i}^2 = 1 \ . \tag{4}$$

  This means that if g has high correlation immunity (i.e. g is not correlated to any linear function in few variables), it is at the same time strongly correlated to linear functions with a higher number of variables. Thus, by choosing the optimal algorithm, a correlation attack is always possible.

Improved correlation immunity from nonlinear memory: In order to destroy the dependency between correlation immunity and linear complexity, Rueppel [43] introduced the generator with (nonlinear) memory. As described in section 1.5, the memory of such a generator consists of two parts: While the majority is made up of LFSRs, some bits are updated by a nonlinear function f_2. It was shown that for a good choice of f_2, such a function can achieve maximum correlation immunity while at the same time having maximum linear complexity.

However, it was proven by Meier, Staffelbach and Golič in [36, 14, 16], that for such a generator, too, a tradeoff similar to (4) can be found. This time, however, several consecutive input bits from each register have to be considered, increasing the number of variables in the linear approximation function L. As a consequence, correlation attacks against combiners with memory are indeed less efficient, but not entirely impossible as was hoped for originally.

4.4 Correlation attacks and clock control

A new notion of correlation: Instead of using nonlinear memory, some LFSR-based generators use nonlinear clocking, i.e. some or all LFSR are irregularly clocked, depending on the internal state of the generator. Note that this way, the attacker can not see which internal bit x_i contributes to which output bit z_j. Thus, measuring the the Hamming distance between $(x_1, \ldots, x_n)$ and $(z_1, \ldots, z_n)$ becomes meaningless, and correlation attacks in the above sense are no longer applicable.

However, other measures of correlation can be used. Golić and Mihaljević [19, 20] proposed to replace the Hamming distance by the so-called *Levenshtein distance*. This distance measures the minimum number of elementary operations (insertion, deletion, and substitution) required to transform one sequence into a prefix of the other. Given such a notion of distance, the standard correlation attack as defined by Siegenthaler can be deployed.

Correlation attacks: Depending on the cipher design, some edit operations may not be allowed. Thus, it may be necessary to define a so-called *Constrained Levenshtein Distance* (CLD). Note, for example, that the Hamming distance is a CLD where only substitutions are allowed. For the decimation generator, only repetitions (a restricted kind of insertion) are applicable. In any case, an efficient dynamic programming algorithm for the computation of the CLD was given in [20]. Given a target sequence of length n, the algorithm computes the CLD in the order of $O(n^2)$ computational steps.

A number of modifications of this attack have been proposed against specific generators [52, 22, 21, 18], but the general method remains the same. In [15], Golić proposes an algorithm similar to the fast correlation attack by Meier and Staffelbach. However, step 1 of the algorithm (finding suitable linear relations) proved to be difficult, except for very special generators. Thus, a full algorithmic specification of a fast correlation attack on general irregularly clocked generators remains an unsolved research problem to date.

References

1. H. Beker and F. Piper. *Cipher Systems - The Protection of Communications.* Northwood Books, London, 1982.
2. E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, IT-24(3):384–386, May 1978.
3. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Proc. Asiacrypt 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000.
4. A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Proc. Eurocrypt 2000*, volume 1807 of *LNCS*, pages 573–588. Springer, 2000.
5. V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In B. Schneier, editor, *Proc. Fast Software Encryption 2000*, volume 1978 of *LNCS*, pages 181–195. Springer, 2000.
6. V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. In D. Davies, editor, *Proc. Eurocrypt '91*, volume 547 of *LNCS*, pages 176–185. Springer, 1991.
7. P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In L. Knudsen, editor, *Proc. Eurocrypt 2002*, volume 2332 of *LNCS*, pages 209–221. Springer, 2002.
8. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In B. Preneel, editor, *Proc. Eurocrypt 2000*, volume 1807 of *LNCS*, pages 392–407. Springer, 2000.
9. N. Ferguson and B. Schneier. *Practical Cryptography.* Wiley, 2003.
10. E. Filiol. Decimation attack of stream ciphers. In B. Roy and E. Okamoto, editors, *Proc. Indocrypt 2000*, volume 1977 of *LNCS*, pages 31–42. Springer, 2000.
11. R. Gallager. *Low-density Parity Check Codes.* MIT Press, Cambridge, 1963.
12. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

13. P. Geffe. How to protect data with ciphers that are really hard to break. *Electronics*, 46(1):99–101, Jan. 1973.
14. J. Golić. Correlation via linear sequential circuit approximation of combiners with memory. In R. Rueppel, editor, *Proc. Eurocrypt '92*, volume 658 of *LNCS*, pages 113–123. Springer, 1993.
15. J. Golić. Towards fast correlation attacks on irregularly clocked shift registers. In L. Guillou and J.-J. Quisquater, editors, *Proc. Eurocrypt '95*, volume 921 of *LNCS*, pages 248–262. Springer, 1995.
16. J. Golić. Correlation properties of a general binary combiner with memory. *Journal of Cryptology*, 9(2):111–126, 1996.
17. J. Golić. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Proc. Eurocrypt '97*, volume 1233 of *LNCS*, pages 239–255. Springer, 1997.
18. J. Golić. Correlation analysis of the shrinking generator. In J. Kilian, editor, *Proc. Crypto 2001*, volume 2139 of *LNCS*, pages 440–457. Springer, 2001.
19. J. Golić and M. Mihaljević. A noisy clock-controlled shift register cryptanalytic concept based on sequence comparison approach. In I. Damgard, editor, *Proc. Eurocrypt '90*, volume 473 of *LNCS*, pages 487–491. Springer, 1990.
20. J. Golić and M. Mihaljević. A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance. *Journal of Cryptology*, 3(3):201–212, 1991.
21. J. Golić and L. O'Connor. Embedding and probabilistic attacks on clock-controlled shift registers. In A. De Santis, editor, *Proc. Eurocrypto '94*, volume 950 of *LNCS*, pages 230–243, Berlin, 1995. Springer.
22. J. Golić and S. Petrović. A generalized correlation attack with a probabilistic constrained edit distance. In R. Rueppel, editor, *Proc. Eurocrypt '92*, volume 658 of *LNCS*, pages 472–476. Springer, 1993.
23. D. Gollmann and W. Chambers. Clock-controlled shift registers: A review. *IEEE J. Selected Areas Comm.*, 7(4):525–533, May 1989.
24. S. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills (CA), revised edition, 1982.
25. T. Johansson and F. Jönsson. Fast correlation attacks based on Turbo Code techniques. In M. Wiener, editor, *Proc. Crypto '99*, volume 1666 of *LNCS*, pages 181–197. Springer, 1999.
26. T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In J. Stern, editor, *Proc. Eurocrypt '99*, volume 1592 of *LNCS*, pages 347–362. Springer, 1999.
27. T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In M. Bellare, editor, *Proc. Crypto 2000*, volume 1880 of *LNCS*, pages 300–315. Springer, 2000.
28. T. Johansson and F. Jönsson. Theoretical analysis of a correlation attack based on convolutional codes. In *Proc. International Symposium on Information Theory*, Sorrento, Italy, June 2000.
29. A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In M. Wiener, editor, *Proc. Crypto '99*, volume 1666 of *LNCS*, pages 19–30. Springer, 1999.
30. D. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms.* Addison-Wesley, 2 edition, 1981.
31. J. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, IT-15:122–127, Jan. 1969.

32. U. Maurer. A universal statistical test for random bit generators. In A. Menezes and S. Vanstone, editors, *Proc. Crypto 1990*, volume 537 of *LNCS*, pages 409–420. Springer, 1991.
33. W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In C. Günther, editor, *Proc. Eurocrypt '88*, volume 330 of *LNCS*, pages 301–314. Springer, 1988.
34. W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–167, 1989.
35. W. Meier and O. Staffelbach. Nonlinearity criteria for cryptographic functions. In J.-J. Quisquater and J. Vandewalle, editors, *Proc. Eurocrypt '89*, volume 434 of *LNCS*, pages 549–562. Springer, 1990.
36. W. Meier and O. Staffelbach. Correlation properties of combiners with memory in stream ciphers. *Journal of Cryptology*, 5(1):67–86, 1992.
37. M. Mihaljević, M. Fossorier, and H. Imai. A low-complexity and high-performance algorithm for the fast correlation attack. In B. Schneier, editor, *Proc. Fast Software Encryption 2000*, volume 1978 of *LNCS*, pages 196–212. Springer, 2001.
38. M. Mihaljević, M. Fossorier, and H. Imai. Fast correlation attack algorithm with list decoding and an application. In M. Matsui, editor, *Proc. Fast Software Encryption 2001*, volume 2355 of *LNCS*, pages 196–210. Springer, 2002.
39. M. Mihaljević and J. Dj. Golić. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence. In J. Seberry and J. Pieprzyk, editors, *Proc. Auscrypt '90*, volume 453 of *LNCS*, pages 165–175. Springer, 1990.
40. W. Penzhorn. Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes. In D. Gollmann, editor, *Proc. Fast Software Encryption '96*, volume 1039 of *LNCS*, pages 159–172. Springer, 1996.
41. V. Pless. *Introduction to the Theory of Error-Correcting Codes.* John Wiley, 1982.
42. R. Rueppel. *Analysis and Design of Stream Ciphers.* Springer, 1986.
43. R. Rueppel. Correlation immunity and the summation generator. In H. Williams, editor, *Proc. Crypto '85*, volume 218 of *LNCS*, pages 260–272. Springer, 1986.
44. R. Rueppel. Stream ciphers. In G. Simmons, editor, *Contemporary Cryptology - The Science of Information Integrity*, pages 65–134. IEEE Press, 1992.
45. R. Rueppel and O. Staffelbach. Products of sequences with maximum linear complexity. *IEEE Transactions on Information Theory*, IT-33(1):124–131, 1987.
46. C. Shannon. A mathematical theory of communication. *Bell Systems Techn. Journal*, 27:623–656, 1948.
47. C. Shannon. Communication theory of secrecy systems. *Bell Systems Techn. Journal*, 28:656–715, 1949.
48. T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, 1984.
49. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Information Theory*, C-34(1):81–85, January 1985.
50. J. van Lint. *Introduction to Coding Theory.* Springer, 1982.
51. G. Vernam. Cipher printing telegraph system for secret wire and radio telegraphic communications. *Journal of American Institute of Electrical Engineers*, 45:109–115, 1926.
52. M. Živković. An algorithm for the initial state reconstruction of the clock-controlled shift register. *IEEE Transactions on Information Theory*, 37(5):1488–1490, Sep. 1991.
53. M. Živković. On two probabilistic decoding algorithms for binary linear codes. *IEEE Transactions on Information Theory*, 37(6):1707–1716, Nov. 1991.

54. G. Xiao and J. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory*, 34(3):569–571, May 1988.
55. K. Zeng and M. Huang. On the linear syndrome method in cryptanalysis. In S. Goldwasser, editor, *Proc. Crypto '88*, volume 403 of *LNCS*, pages 469–478. Springer, 1990.
56. K. Zeng, C. Yang, and T. Rao. On the linear consistency test (LCT) in cryptanalysis with applications. In G. Brassard, editor, *Proc. Crypto '89*, volume 435 of *LNCS*, pages 164–174. Springer, 1990.
57. K. Zeng, C. Yang, and T. Rao. An improved linear syndrome algorithm in cryptanalysis with applications. In A. Menezes and S. Vanstone, editors, *Proc. Crypto '90*, volume 537 of *LNCS*, pages 34–47. Springer, 1991.
58. E. Zenner. On the efficiency of clock control guessing. In P. J. Lee and C. H. Lim, editors, *Proc. ICISC '02*, volume 2587 of *LNCS*, pages 200–212. Springer, 2003.
59. E. Zenner, M. Krause, and S. Lucks. Improved cryptanalysis of the self-shrinking generator. In V. Varadharajan and Y. Mu, editors, *Proc. ACISP '01*, volume 2119 of *LNCS*, pages 21–35. Springer, 2001.