Chapter 1

Classic Pseudorandom Generators: Congruential Generators and Feedback Shift Registers

"Classic" (pseudo-) random generators are algorithms that generate "pseudorandom" numbers or bits for use in statistical applications or simulations instead of "true" random numbers or bits. For this kind of applications their statistical properties are excellent. The standard references are [2] and [3].

However the requirements of cryptology are much stronger. The methods of cryptanalysis mercilessly reveal the weaknesses of classic pseudorandom generators, and make them useless for naive direct cryptographic application.

This chapter introduces the best known classic pseudorandom generators and derives their most important properties.

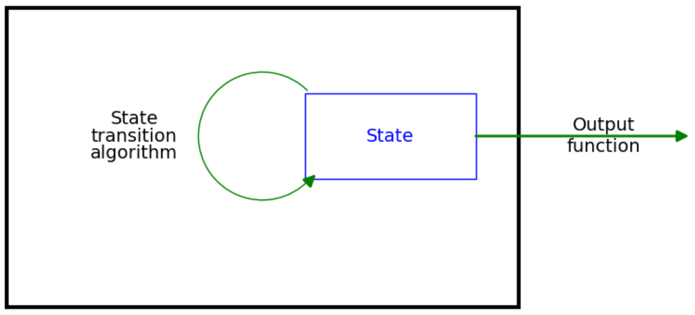

Figure 1.1: A simple model of pseudorandom generation. The state is an element of a set $\mathcal{M}$, changing after each step according to the state transition algorithm $\mathcal{M} \longrightarrow \mathcal{M}$. The output (of each step) is an element of an output alphabet Σ.

1.1 General Discussion of Bitstream Ciphers

As a first example of a bitstream cipher we encountered XOR in Part I of these lectures. SageMath code is in Appendix E.1 of Part II.

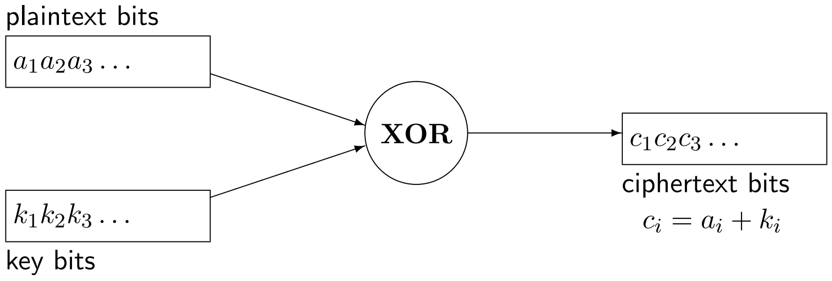

Figure 1.2: The principle of XOR encryption

In the twenties of the 20th century XOR ciphers were invented to encrypt teleprinter messages. These messages were written on five-hole punched tapes as in Figure 1.3, each column representing a five-bit-block. Another punched tape provided the key stream. Vernam filed this procedure as a U. S. patent in 1918. He used a key tape whose ends were glued together, resulting in a periodic key stream. Mauborgne immediately recognized that a nonperiodic key is obligatory for security.

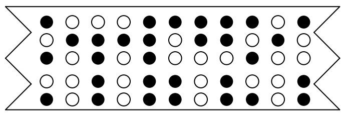

Figure 1.3: Punched tape—each column represents a five-bit character

In its strongest form, the one-time pad, XOR encryption is an example of perfect security in the sense of Shannon, see Part I, Section 10. As algorithm A5 or E_0 XOR helps to secure mobile phones or the Bluetooth protocol for wireless data transmission. As RC4 it is part of the SSL protocol that (often) encrypts client-server communication in the World Wide Web, and of the PKZIP compression software. There are many other current applications, not all of them satisfying the expected security requirements.

The scope of XOR encryption ranges from simple ciphers that are trivially broken to unbreakable ciphers.

Advantages of XOR ciphers

- Encryption and decryption are done by the same algorithm: Since $c_i = a_i + k_i$ also $a_i = c_i + k_i$. Thus decryption consists of adding key stream and ciphertext (elementwise binary).
- The method is extremely simple to understand and to implement
- ... and very fast—provided that the key stream is available. For high transfer rates one may precompute the key stream at both ends of the line.
- If the key stream is properly chosen the security is high.

Pitfalls

- XOR ciphers are vulnerable under known plaintext attacks: each correctly guessed plaintext bit reveals a key bit.
- If the attacker knows a piece of plaintext she also knows the corresponding piece of the key stream, and then is able to exchange this plaintext at will. For example she might replace “I love you” by “I hate you”, or replace an amout of 1000$ by 9999$. In other words the integrity of the message is poorly protected. (To protect message integrity the sender has to implement an extended procedure.)
- XOR ciphers provide no diffusion in the sense of SHANNON’s criteria since each plaintext bit affects the one corresponding plaintext bit only.
- Each reuse of a part of the key sequence (also in form of a periodic repetition) opens the door for an attack. The historical successes in breaking stream ciphers almost always used this effect, for example the attacks on encrypted teleprinters in World War II, or the project VENONA during the Cold War.

A remark on the first item, the vulnerability for attacks with known plaintext: The common ISO character set for texts has a systematic weakness. The 8-bit codes of the lower-case letters `a..z` all start with `011`, of the upper-case letters `A..Z`, with `010`. A supposed sequence of six lower-case letters (no matter which) reveals $6 \cdot 3 = 18$ key bits.

> By the way the appearance of many zeroes in the leading bits of the bytes is an important identifying feature of texts in many European languages.

In other words: We cannot prevent the attacker from getting or guessing a good portion of the plaintext. Thus the security against an attack with known plaintext is a fundamental requirement for an XOR cipher, even more than for any other cryptographic procedure.

This being said the crucial question for a pseudorandom sequence, or for the pseudorandom generator producing it, is:

> *Is it possible to determine some more bits from a (maybe fragmented) chunk of the sequence?*

The answer for the "classic" pseudorandom generators will be YES. But we'll also learn about pseudorandom generators that—supposedly—are cryptographically secure in this sense.

1.2 Methods for Generating a Key Stream

The main naive methods for generating the key stream are:

- periodic bit sequence,
- running-text,
- “true” random sequence.

A better method uses a

- pseudorandom sequence

and leads to really useful procedures. The essential criterion is the quality of the random generator.

Note We sometimes use the term “random generator” for an algorithm that produces pseudorandom sequences (of bits or numbers). The more correct denomination is “pseudorandom generator”.

Periodic Bit Sequence

A periodically repeated (longer or shorter) bit sequence serves as key. Technically this is a Bellaso cipher over the alphabet $\mathbb{F}_2$. The classical cryptanalytic methods for periodic polyalphabetic ciphers apply, such as period analysis or probable word.

For an example see XOR in Part I.

Known or probable plaintext easily breaks periodic XOR encryption.

MS Word and Periodic XOR

The following table (generated ad hoc by simple character counts) shows the frequencies of the most frequent bytes in MS Word files.

byte (hexadecimal)	**bits**	**frequency**
`00`	00000000	7–70%
`01`	00000001	0.8–17%
`20` (space)	00100000	0.8–12%
`65` (e)	01100101	1–10%
`FF`	11111111	1–10%

Note that these frequencies relate to the binary files, heavily depend on the type of the document, and may change with every software version. The variation is large, we often find unexpected peaks, and all bytes `00`–`FF` occur. But all this doesn’t matter here since we observe *long chains of* `00` *bytes.*

For an MS Word file that is XOR encrypted with a periodically repeated key the ubiquity of zero bits suggests an efficient attack: Split the stream of ciphertext bits into blocks corresponding to the length of the period and add the blocks pairwise. If one of the plaintext blocks essentially consists of 0's, then the sum is readable plaintext. Why? Consider the situation

	…	block 1			…	block 2			…
plaintext:	…	a_1	…	a_s	…	0	…	0	…
key:	…	k_1	…	k_s	…	k_1	…	k_s	…
ciphertext:	…	c_1	…	c_s	…	c'_1	…	c'_s	…

where $c_i = a_i + k_i$ and $c'_i = 0 + k_i = k_i$ for $i = 1, \ldots, s$. Thus the key reveals itself in block 2, however the attacker doesn't recognize this yet. But tentatively paarwise adding all blocks she gets (amongst other things)

$$c_i + c'_i = a_i + k_i + k_i = a_i \quad \text{for } i = 1, \ldots, s,$$

that is, a plaintext block. If she realizes this (for example recognizing typical structures), then she recognizes the key $k_1, \ldots, k_s$.

Should it happen that the sum of two ciphertext blocks is zero then the ciphertext blocks are equal, and so are the corresponding plaintext blocks. The probability is high that both of them are zero. Thus the key could immediately show through. To summarize:

XOR encryption with a periodic key stream is quite easily broken for messages with a known structure.

This is true also for a large period, say 512 bytes = 4096 bits, in spite of the hyperastronomically huge key space of 2^{4096} different possible keys.

Running-Text Encryption

A classical approach to generating an aperiodic key is taking an existing data stream, or file, or text, that has at least the length of the plaintext. In classical cryptography this method was called running-text encryption. We won't repeat the cryptanalytic techniques but summarize:

XOR encryption with running-text keys is fairly easily broken.

True Random Sequence

The extreme choice for a key is a true random sequence of bits as key stream. Then the cipher is called **(binary) one-time pad (OTP)**. In particular no part of the key stream must be repeated at any time. The notation "pad" comes from the idea of a tear-off calendar—each sheet is destroyed after use. This cipher is unbreakable, or "perfectly secure". SHANNON gave a formal proof of this, see Part I, Section 10.

Without mathematical formalism the argument is as follows: The ciphertext divulges no information about the plaintext (except the length). It could result from *any* plaintext of the same length: simply take the (binary) difference of ciphertext and alleged plaintext as key. Consider the ciphertext $c = a + k$ with plaintext a and key k, all represented by bitstreams and added bit by bit as in Figure 1.2. For an arbitrary different plaintext b the formula $c = b + k'$ likewise shows a valid encryption using $k' = b + c$ as key.

This property of the OTP could be used in a scenario of forced decryption (also known as “rubber hose cryptanalysis”) to produce an innocuous plaintext, as exemplified in Figure 1.4.

If the one-time pad is perfect—why don't we use it in any case and forget of all other ciphers?

- The key management is unwieldy: Key agreement becomes a severe problem since the key is as long as the plaintext and awkwardly to memorize. Thus the communication partners have to agree on the key stream prior to transmitting the message, and store it. Agreeing on a key only just in time needs a secure communication channel—but if there was one why not use it to transmit the plaintext in clear?

- The key management is inappropriate for mass application or multi-party communication because of its complexity that grows with each additional participant.

- The problem of message integrity requires an extended solution for OTP like for any XOR cipher.

There is another, practical, problem when encrypting on a computer: How to get random sequences? “True random” bits arise from physical events like radioactive decay, or thermal noise on an optical sensor. The apparently deterministic machine “computer” can also generate true random bits, for instance by special chips that produce usable noise. Moreover many events are unpredictable, such as the exact mouse movements of the user, or arriving network packets that, although not completey random, contain random ingredients that may be extracted. On Unix systems these random bits are provided by `/dev/random`.

However these random bits, no matter how “true”, are not that useful for encryption by OTP. The problem is on the side of the receiver who cannot reproduce the key. Thus the key stream must be transmitted independently.

There are other, useful, cryptographic applications of “true” random bits: Generating keys for arbitrary encryption algorithms that are unpredictable for the attacker. Many cryptographic protocols rely on “nonces” that have no meaning except for being random, for example the initialization vectors of the block cipher modes of operation, or the “challenge” for strong authentication (“challenge-response protocol”).

```
Plain bits and text:
01010100 01101000 01101001 01110011 00100000 01101101   This m
01100101 01110011 01110011 01100001 01100111 01100101   essage
00100000 01101001 01110011 00100000 01101000 01100001    is ha
01111010 01100001 01110010 01100100 01101111 01110101   zardou
01110011 00101110                                       s.

Key bits:
11001000 11010110 00110011 11000000 00111011 10001110
00001000 11101111 01001001 11100101 10111100 10111001
00010010 11000110 01110011 11010111 11000100 01100000
11100110 00010111 01101010 10111011 00010101 11011000
11110000 01000010

Cipher bits:
10011100 10111110 01011010 10110011 00011011 11100011
01101101 10011100 00111010 10000100 11011011 11011100
00110010 10101111 00000000 11110111 10101100 00000001
10011100 01110110 00011000 11011111 01111010 10101101
10000011 01101100

Pseudokey bits:
11001000 11010110 00110011 11000000 00111011 10001110
00001000 11101111 01001001 11100101 10111100 10111001
00010010 11000110 01110011 11010111 11000101 01101111
11110010 00011001 01111011 10101010 00010101 11011000
11110000 01000010

Pseudodecrypted bits and text:
01010100 01101000 01101001 01110011 00100000 01101101   This m
01100101 01110011 01110011 01100001 01100111 01100101   essage
00100000 01101001 01110011 00100000 01101001 01101110    is in
01101110 01101111 01100011 01110101 01101111 01110101   nocuou
01110011 00101110                                       s.
```

Figure 1.4: XOR encryption of a hazardous message, and an alleged alternative plaintext

Pseudorandom Sequence

For XOR encryption—as approximation to the OTP—algorithmically generated bit sequences are much more practicable. But the attacker should have no means to distinguish them from true random sequences. This is the essence of the concept of pseudorandomness, and generating pseudorandom sequences is of fundamental cryptologic relevance.

XOR encryption with a pseudorandom key stream spoils the perfect security of the one-time pad. But if the pseudorandom sequence is cryptographically strong (Chapter 4) the attacker has no chance to exploit this fact.

To be useful for cryptographic purposes the pseudorandom key stream must depend on parameters the attacker has no access to and that represent (parts of) the cryptographic key. Such parameters might be, see Figure 1.5 that extends the basic model of a pseudorandom generator:

- the initial value of the state,
- parameters the transition algorithm depends on.

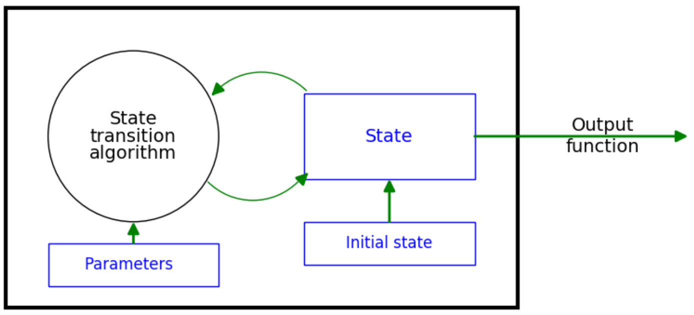

Figure 1.5: Secret parameters for a pseudorandom generator

1.3 Linear Congruential Generators

As a first important class of elementary—“classical”—pseudorandom number generators we consider one-step recursive formulas that use linear congruences. They are very fast, have long periods, and their quality is easily analyzed due to their plain structure.

This simple formula generates a sequence of pseudorandom numbers:

$$x_n = ax_{n-1} + b \bmod m. \tag{1}$$

The recursive sequence $(x_n)_{n\in\mathbb{N}}$ depends on four integer parameters:

- the **module** m where $m \geq 2$,
- the **multiplier** $a \in [0 \dots m-1]$,
- the **increment** $b \in [0 \dots m-1]$,
- the **initial value** $x_0 \in [0 \dots m-1]$.

We call this recursive formula a **linear congruential generator**, in the case $b = 0$ also a **multiplicative generator**, in the case $b \neq 0$, a **mixed congruential generator**. Furthermore we call

$$s : \mathbb{Z}/m\mathbb{Z} \longrightarrow \mathbb{Z}/m\mathbb{Z}, \quad s(x) = ax + b \bmod m.$$

the **generating function** of the generator. Formula (1) then becomes

$$x_n = s(x_{n-1}).$$

Programming a linear congruential pseudorandom generator is extremely easy, even in assembler languages; for Sage see Sage sample 1.1. The algorithm works very fast. Moreover the pseudorandom numbers are statistically good *if the parameters m, a, b are suitably chosen.* In contrast the choice of the initial value is unrestricted. This freedom allows a reasonable variation of the generated pseudorandom numbers.

Use of the pseudorandom sequence as a bitstream for XOR encryption requires at least that we consider the initial value x_0, or the complete parameter set (m, a, b, x_0), as effective key, and keep it secret, cf. Figure 1.5.

Remarks and Examples

1. Since x_n may assume only m different values the sequence is periodic with a period length $\leq m$; including a possible preperiod.

2. Choosing $a = 0$ obviously doesn't make sense. Also for $a = 1$ we get a useless sequence, namely $x_0, x_0 + b, x_0 + 2b, x_0 + 3b, \dots$, that also $\bmod\, m$ contains several regular subsequences.

Sage Example 1.1 Generating pseudorandom numbers by a linear congruential random generator

```
def lcg(m,a,b,s,n):
  x = s
  outlist = []
  for i in range (0,n):
    y = (a*x + b) % m
    outlist.append(y)
    x = y
  return outlist
```

3. For $m = 13$, $a = 6$, $b = 0$, $x_0 = 1$ we get the sequence

   $$6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1$$

   of period length 12 that looks like a fairly random permutation of the integers 1 to 12, despite the small module.

4. Choosing the multiplier $a = 7$ instead of 6 we get a much less sympathic sequence:

   $$7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1.$$

5. If a and m are coprime, then the sequence is purely periodic (no preperiod). For $a \bmod m$ is invertible, hence $ac \equiv 1 \pmod m$ for some c. Thus always $x_{n-1} = cx_n - cb \bmod m$. If $x_{\mu+\lambda} = x_\mu$ with $\mu \geq 1$, then also $x_{\mu+\lambda-1} = x_{\mu-1}$ etc., finally $x_\lambda = x_0$.

6. By induction we immediately get

   $$x_k = a^k x_0 + (1 + a + \cdots + a^{k-1}) \cdot b \bmod m \tag{2}$$

   for all k—a definite warning about the poor randomness of the sequence: Formula (2) allows direct access to any element of the sequence. Note that the coefficient of b is $(a^k - 1)/(a - 1)$ where the division is $\bmod m$.

7. Let $m = 2^e$ and a be even. Then

   $$x_k = (1 + a + \cdots + a^{e-1}) \cdot b \bmod m$$

   for all $k \geq e$, hence, after a certain preperiod, the period has length 1. More generally common divisors of a and m reduce the period. We want to avoid this effect.

8. Let d be a divisor of m. Then the sequence $y_n = x_n \bmod d$ is the analogous congruential sequence for the module d, generated by the formula $y_n = ay_{n-1} + b \bmod d$. Hence the sequence (x_n), if considered mod d, has a period $\leq d$ that might be very short.

9. This effect is especially inconvenient in the case of a power $m = 2^e$: Then the least significant bit of x_n has a period of length at most 2, hence alternates between 0 and 1, or is constant. And the k least significant bits together have a period of at most 2^k.

10. The innocuously looking example $m = 2^{32}$, $a = 4095 = 2^{12} - 1$, $b = 12794$ exhibits an extremely bad choice of parameters: From $x_0 = 253$ we get $x_1 = 1048829$ and $x_2 = 253 = x_0$.

Preferred modules are

- $m = 2^{32}$ that exhausts the 32 bit range and moreover is computationally efficient,
- $m = 2^{31} - 1$ that is the maximum 32 bit integer, and computationally almost as efficient as a power of 2. Another advantage: This number is prime (claimed by MERSENNE in 1644, proved by EULER in 1772), and this enhances the quality of the pseudorandom sequence. More generally these arguments apply to FERMAT primes $2^k + 1$ and MERSENNE primes $2^k - 1$. The next prime of this kind is $2^{61} - 1$.

Table 1.1 shows the first 100 members of a sequence that is generated with the module $m = 2^{31} - 1 = 2147483647$, the multiplier $a = 397204094$, the increment $b = 0$, and the initial value $x_0 = 58854338$, Sage code sample 1.2. Figure 1.6 gives a visual impression of this information. We see that the sequence doesn't follow any obvious rules. However it is clear that such a visual impression is not a sufficient criterion for the quality of a pseudorandom sequence.

Sage Example 1.2 Using a linear congruential random generator

```
sage: mm = 2**31 - 1
sage: aa = 397204094
sage: bb = 0
sage: seed = 58854338
sage: seq = lcg(mm,aa,bb,seed,100); seq
```

Table 1.1: 100 members of a linear congruential sequence

1292048469	319941267	173739233	1992841820
345565651	2011011872	31344917	592918912
1827933824	1691830787	857231706	1416540893
1184833417	145217588	589958351	1776690121
1330128247	558009026	1479515830	1197548384
1627901332	929586843	19840670	1268974074
1682548197	760357405	666131673	1642023821
787305132	1314353697	167412640	1377012759
963849348	971229179	247170576	1250747100
703109068	1791051358	1978610456	1746992541
177131972	1844679385	1328403386	1811091691
1586500120	1175539757	74957396	753264023
468643347	821920620	1269873360	963348259
1698955999	139484430	30476960	1327705603
1266305157	1337811914	1808105128	640050202
37935526	1185470453	2111728842	380228478
808553600	934194915	824017077	881361640
1492263703	414709486	298916786	1883338449
771128019	558671080	1935988732	798347213
120356246	1378842534	37149011	272238278
1190345324	1006355270	1161592162	1079789655
220609946	1918105148	791775291	979447727
1160648370	779600833	1170336930	1271974642
375813045	1089009771	280197098	1144249742
1236647368	1729816359	650188387	1714906064

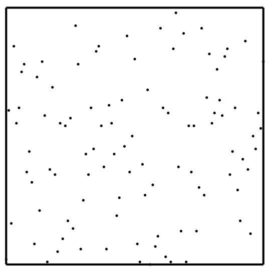

Figure 1.6: A linear congruential sequence. Horizontal axis: counter from 0 to 100, vertical axis: size of the integer from 0 to $2^{31} - 1$.

1.4 The Maximum Period Length

Under what conditions does the period of a linear congruential generator with module m attain the theoretic maximum length m? A multiplicative generator will never attain this period since the output 0 reproduces itself forever. Thus for this question we consider mixed generators with nonzero increment. As the trivial generator with generating function $s(x) = x + 1 \bmod m$ shows the period length m really occurs; on the other hand this example also shows that a period of maximum length is insufficient as a proof of quality for a random generator. Nevertheless maximum period is an important criterion, and the general result is easily stated:

Proposition 1 (Hull/Dobell **1962,** Knuth) *The linear congruential generator with generating function $s(x) = ax + b \bmod m$ has period m if and only if the following three conditions hold:*

(i) *b and m are coprime.*

(ii) *Each prime divisor p of m divides $a - 1$.*

(iii) *If 4 divides m, then 4 divides $a - 1$.*

From the first condition we conclude $b \neq 0$, hence the generator is mixed. Before giving the proof of the proposition we state and prove a lemma. (We'll use two more lemmas from Part III, Appendix A.1, that we state here without proofs.)

Lemma 1 *Let $m = m_1m_2$ with coprime natural numbers m_1 and m_2. Let λ, λ_1, and λ_2 be the periods of the congruential generators $x_n = s(x_{n-1}) \bmod m$, $\bmod\, m_1$, $\bmod\, m_2$ with initial value x_0 in each case. Then λ is the least common multiple of λ_1 and λ_2.*

Proof. Let $x_n^{(1)}$ and $x_n^{(2)}$ be the corresponding outputs for m_1 and m_2. Then $x_n^{(i)} = x_n \bmod m_i$. Since $x_{n+\lambda} = x_n$ for all sufficiently large n we immediately see that λ is a multiple of λ_1 and λ_2. On the other hand from $m \mid t \iff m_1, m_2 \mid t$ we get

$$x_n = x_k \iff x_n^{(i)} = x_k^{(i)} \quad \text{for } i = 1 \text{ and } 2.$$

Hence λ is not larger than the least common multiple of λ_1 and λ_2. $\diamond$

The two lemmas without proofs:

Lemma 2 *Let $n = 2^e$ with $e \geq 2$.*

(i) *If a is odd, then*

$$a^{2^s} \equiv 1 \pmod{2^{s+2}} \quad \textit{for all } s \geq 1.$$

(ii) *If* $a \equiv 3 \pmod 4$, *then* $n \mid 1 + a + \cdots + a^{n/2-1}$.

Lemma 3 *Let* p *be prime, and* e, *a natural number with* $p^e \geq 3$. *Assume* p^e *is the largest power of* p *that divides* $x - 1$. *Then* p^{e+1} *is the largest power of* p *that divides* $x^p - 1$.

Proof of the proposition For both directions we may assume $m = p^e$ where p is prime by Lemma 1.

"$\Longrightarrow$": Each residue class in $[0 \ldots m-1]$ occurs exactly once during a full period. Hence we may assume $x_0 = 0$. Then

$$x_n = (1 + a + \cdots + a^{n-1}) \cdot b \bmod m \quad \text{for all } n.$$

Since x_n assumes the value 1 for some n we conclude that b is invertible $\bmod m$, or that b and m are coprime.

Let $p \mid m$. From $x_m = 0$ we now get $m \mid 1 + a + \cdots + a^{m-1}$, hence

$$p \mid m \mid a^m - 1 = (a-1)(1 + a + \cdots + a^{m-1}).$$

FERMAT's little theorem gives $a^p \equiv a \pmod p$, hence

$$a^m = a^{p^e} \equiv a^{p^{e-1}} \equiv \ldots \equiv a \pmod p,$$

hence $p \mid a - 1$. This proves (ii).

Statement (iii) corresponds to the case $p = 2$ with $e \geq 2$. From (ii) we get that a is even. The assumption $a \equiv 3 \pmod 4$ would result in the contradiction $x_{m/2} = 0$ by Lemma 2. Hence $a \equiv 1 \pmod 4$.

"$\Longleftarrow$": Again we may assume $x_0 = 0$. Then

$$x_n = 0 \Longleftrightarrow m \mid 1 + a + \cdots + a^{n-1}.$$

In particular the case $a = 1$ is trivial. Hence assume $a \geq 2$. Then

$$x_n = 0 \Longleftrightarrow m \mid \frac{a^n - 1}{a - 1}.$$

We have to show:

- $m \mid \frac{a^m-1}{a-1}$—then $\lambda | m$;
- m doesn't divide $\frac{a^{m/p}-1}{a-1}$—then $\lambda \geq m$ since m is a power of p.

Let p^h be the maximum power that divides $a - 1$. By Lemma 3 we conclude

$$a^p \equiv 1 \pmod{p^{h+1}}, \quad a^p \not\equiv 1 \pmod{p^{h+2}}$$

and successively

$$a^{p^k} \equiv 1 \pmod{p^{h+k}}, \quad a^{p^k} \not\equiv 1 \pmod{p^{h+k+1}}$$

for all k. In particular $p^{h+e} \mid a^m - 1$. Since no larger power than p^h divides $a-1$ we conclude that $m = p^e \mid \frac{a^m-1}{a-1}$. The assumption $p^e \mid \frac{a^{m/p}-1}{a-1}$ leads to the contradiction $p^{e+h} \mid a^{p^{e-1}} - 1$. $\diamondsuit$

The main application of Proposition 1 is for modules that are powers of 2:

Corollary 1 (Greenberger 1961) *For the module $m = 2^e$ with $e \geq 2$ the period m is attained if and only if:*

(i) *b is odd.*

(ii) *$a \equiv 1 \pmod 4$.*

For prime modules Proposition 1 is useless, as the following corollary shows.

Corollary 2 *For a prime module m the period m is attained if and only if b is coprime with m and $a = 1$.*

This (lousy) result admits an immediate generalization to squarefree modules m:

Corollary 3 *For a squarefree module m the period m is attained if and only if b is coprime with m and $a = 1$.*

In summary Proposition 1 shows how to get the maximum possible period, and Corollary 1 provides a class of half-decent useful examples.

1.5 The Maximum Period of a Multiplicative Generator

A multiplicative generator $x_n = ax_{n-1} \bmod m$ never has period m since the output 0 reproduces itself. So what is the largest possible period? In the following proposition λ is the CARMICHAEL function, and this is exactly the context where it occurred for the first time.

Proposition 2 (CARMICHAEL **1910)** *The maximum period of a multiplicative generator with generating function $s(x) = ax \bmod m$ is $\lambda(m)$. A sufficient condition for the period $\lambda(m)$ is:*

(i) *a is primitive* $\bmod m$.

(ii) *x_0 is relatively prime to m.*

Proof. We have $x_n = a^n x_0 \bmod m$. If $k = \mathrm{ord}_m a$ is the order of a in the multiplicative group of $\mathbb{Z}/m\mathbb{Z}$, then $x_k = x_0$. Thus the period is $\leq k \leq \lambda(m)$. Now assume a is primitive mod m, hence $1, a, \ldots, a^{\lambda(m)-1} \bmod m$ are distinct, and let x_0 be relatively prime to m. Then the x_n are distinct for $n = 0, \ldots, \lambda(m) - 1$, and the period is $\lambda(m)$. $\diamond$

Corollary 1 *Let $m = p$ prime. Then the generator has the maximum period $\lambda(p) = p - 1$ if and only if:*
(i) *a is primitive* $\bmod p$.
(ii) $x_0 \neq 0$.

Thus for prime modules we are in a comfortable situation: The period misses the maximum value for one-step recursive generators only by 1, and any initial value is good except 0.

Section 1.9 will broadly generalize this result.

How to find a primitive element is comprehensively discussed in Appendix A of Part III.

1.6 Feedback Shift Registers

Feedback shift registers (FSR) are a classical and popular method of generating pseudorandom sequences. The method goes back to GOLOMB in 1955 [2], but is often named after TAUSWORTHE who picked up the idea in a 1965 paper. FSRs are especially convenient for hardware implementation.

An FSR of length l is specified by a Boolean function $f\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$, the "feedback function". Figure 1.7 shows the mode of operation—representing f by a Boolean circuit yields an explicit construction plan. The output consists of the rightmost bit u_0, all the other bits are shifted to the right by one position, and the leftmost cell is filled by the bit $u_l = f(u_{l-1}, \ldots, u_0)$. Thus the recursive formula

$$u_n = f(u_{n-1}, \ldots, u_{n-l}) \quad \text{for } n \geq l \tag{3}$$

represents the complete sequence.

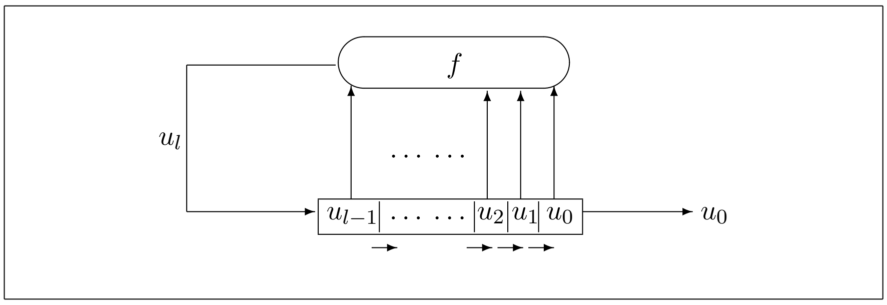

Figure 1.7: A feedback shift register (FSR)

The bits $u_0, \ldots, u_{l-1}$ form the start value. The "key expansion" transforms the short sequence $u = (u_0, \ldots, u_{l-1})$ (the effective key) of length l into a key stream $u_0, u_1, \ldots$ of arbitrary length. In a cryptographic context the bits $u_0, u_1, \ldots$ form the key stream. In other contexts it might be unnecessary to conceal the output bits, but even then hiding the initial state might make sense, starting the output sequence at u_l. Additionally in a cryptographic context treating the internal parameters, that is the feedback function f or some of its coefficients, as components of the key makes sense. Then the effective key length is larger than l.

In this respect the realization in hardware differs from a software implementation: Hardware allows using an adjustable feedback function only by complex additional circuits. Thus in the hardware case we usually assume an unchangeable feedback function, and (at least in the long run) we cannot prevent the attacker from figuring it out. In contrast a software implementation allows a comfortable change of the feedback function at any time such that it may serve as part of the key.

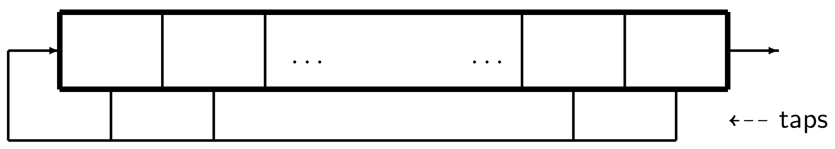

Figure 1.8: Simple graphical representation of an LFSR

For a Sage example we use the procedure `fsr` from Appendix C.1 and the construction of a Boolean function from Appendix E.3 of Part II. (Attach the modules `Bitblock.sage`, `boolF.sage`, `FSR.sage`.)

Sage Example 1.3 Generating a bitsequence by a nonlinear FSR

```
sage: f2 = BoolF([1,1,1,0,1,1,0,0,0,1,0,0,0,1,1,0])
sage: start = [0,1,1,1]
sage: seq = fsr(f2,start,20); seq
[1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

The simplest and best understood instances of FSRs are the linear feedback shift registers (LFSR). Their feedback functions f are linear. From Part II we know that a linear function

$$f\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$$

is simply a partial sum from an l-bit block:

$$f(u_{n-1},\ldots,u_{n-l}) = \sum_{j=1}^{l} a_j u_{n-j}, \tag{4}$$

where the coefficients a_j are 0 or 1. If I is the subset of indices j with $a_j = 1$, then the iteration (3) takes the form

$$u_n = \sum_{j\in I} u_{n-j}. \tag{5}$$

A simple graphical representation of an LFSR is shown in Figure 1.8. Here the subset I defines the contacts ("taps") that feed the respective cell contents into the feedback sum.

In SageMath we implement a special class `LFSR`, see Appendix C.1 whose use is demonstrated in the code sample 1.4.

For a good choice of the parameters, see Section 1.9, the sequence has a period of about 2^l, the number of possible different states of the register, and statistical tests are hardly able to distinguish it from a uniformly distributed true random sequence, see Section 1.10. It is remarkable that such a simple

Sage Example 1.4 Generating a bitsequence by a linear FSR

```
sage: coeff = [0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,1]
sage: reg = LFSR(coeff)
sage: start = [0,1,1,0,1,0,1,1,0,0,0,1,0,0,1,1]
sage: reg.setState(start)
sage: bitlist = reg.nextBits(20); bitlist
[1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1]
```

approach generates pseudorandom sequences of fairly high quality! Of course the initial state $u = (0, \ldots, 0)$ is inappropriate. For an initial state $\neq 0$ the maximum possible period is $2^l - 1$, see Section 1.9.

For using an LFSR for bitstream encryption the secret inner parameters—the coefficients $a_1, \ldots, a_l$—as well as the initial state $u_0, \ldots, u_{l-1}$ together constitute the key. In contrast the length l of the register is assumed as known to the attacker. Beware of Section 2.3!

1.7 Multistep generators

Multistep (linear recursive) generators are a common generalization of linear congruential generators and LFSRs. A convenient framework for their treatment is a finite ring R (commutative with 1); this comprises not only the residue class rings $\mathbb{Z}/m\mathbb{Z}$ but also the finite fields including the prime fields $\mathbb{F}_p$.

An r-step linear recursive generator outputs a sequence (x_n) in R by the rule

$$x_n = a_1x_{n-1} + \cdots + a_rx_{n-r} + b.$$

The parameters of this procedure are

- the **recursion depth** r (assume $a_r \neq 0$),
- the **coefficient tuple** $a = (a_1, \ldots, a_r) \in R^r$,
- the **increment** $b \in R$,
- a **start vector** $(x_0, \ldots, x_{r-1}) \in R^r$.

The linear recursive generator is called **homogeneous** if the increment $b = 0$, **inhomogeneous** otherwise.

Figure 1.9 visualizes the operation of a linear recursive generator in analogy with an LFSR.

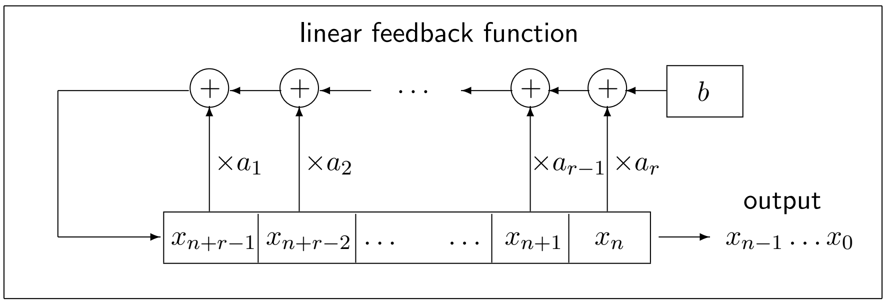

Figure 1.9: A linear recursive generator

Inhomogeneous linear recursive generators easily reduce to homogeneous ones, but only with an additional recursion step: Subtracting the two equations

$$\begin{aligned} x_{n+1} &= a_1x_n + \cdots + a_rx_{n-r+1} + b, \\ x_n &= a_1x_{n-1} + \cdots + a_rx_{n-r} + b, \end{aligned}$$

we get

$$x_{n+1} = (a_1 + 1)x_n + (a_2 - a_1)x_{n-1} \cdots + (-a_r)x_{n-r}.$$

Example In the case $r = 1$, $x_n = ax_{n-1} + b$, this formula becomes

$$x_n = (a+1)x_{n-1} - ax_{n-2}.$$

In the following we often neglect the inhomogeneous case.

In the homogeneous case we introduce the **state vectors** $x_{(n)} = (x_n, \ldots, x_{n+r-1})^t$ and write

$$x_{(n)} = Ax_{(n-1)} \quad \text{for } n \geq 1,$$

using the **companion matrix**

$$A = \begin{pmatrix} 0 & 1 & \ldots & 0 \\ & \ddots & \ddots & \\ & & & 1 \\ a_r & a_{r-1} & \ldots & a_1 \end{pmatrix}.$$

This suggests the next step of generalization: the **matrix generator** with parameters:

- an $r \times r$-matrix $A \in M_r(R)$,
- a start vector $x_0 \in R^r$.

The output sequence is generated by the formula

$$x_n = Ax_{n-1} \in R^r.$$

1.8 General linear generators

Even more general (and conceptually simpler) is the abstract algebraic version, the **general linear generator**. This is the setting:

- a ring R (commutative with 1),
- an R-module M,
- an R-linear map $A : M \longrightarrow M$,
- a start value $x_0 \in M$.

From this we generate a sequence $(x_n)_{n\in\mathbb{N}}$ by the formula

$$x_n = Ax_{n-1} \quad \text{for } n \geq 1. \tag{6}$$

Examples

1. For a homogeneous linear congruential generator we have
$$R = \mathbb{Z}/m\mathbb{Z}, \quad M = R \quad (r = 1), \quad A = (a).$$
2. For an inhomogeneous linear congruential generator we have
$$R = \mathbb{Z}/m\mathbb{Z}, \quad M = R^2 \quad (r = 2), \quad A = \begin{pmatrix} 0 & 1 \\ -a & a+1 \end{pmatrix}.$$
3. For an LFSR we have
$$R = \mathbb{F}_2, \quad M = \mathbb{F}_2^l \quad (r = l), \quad A = \text{the companion matrix},$$
that contains only 0's and 1's.

In the case of a finite M the recursion (6) can assume only finitely many different values, therefore (after a potential preperiod) must become periodic.

Proposition 3 *Let M be a finite R-module and $A : M \longrightarrow M$ be linear. Then the following statements are equivalent:*

(i) *All sequences generated by the corresponding general linear generator (6) are purely periodic.*

(ii) *A is bijective.*

Proof. "(i) $\Longrightarrow$ (ii)": Assume that A is not bijective. Since M is finite A is not surjective. Hence there is an $x_0 \in M - A(M)$. Then $x_0 = Ax_t$ can never occur, hence the sequence is not purely periodic.

"(ii) $\Longrightarrow$ (i)": Let A be bijective and x_0, an arbitrary start vector. Let t be the first index such that x_t assumes a value that occured before, and let s be the smallest index with $x_t = x_s$. Since $x_s = Ax_{s-1}$ and $x_t = Ax_{t-1}$ the assumption $s \geq 1$ leads to

$$x_{t-1} = A^{-1}x_t = A^{-1}x_s = x_{s-1},$$

contradicting the minimality of t. $\diamond$

Looking at the companion matrix we immediately apply this result to homogeneous multistep congruential generators, and in particular to LFSRs:

Corollary 1 *A homogeneous linear congruential generator of recursion depth r always generates purely periodic sequences if the coefficient a_r is invertible in $\mathbb{Z}/m\mathbb{Z}$.*

This is true also in the inhomogeneous case since the formula

$$x_{n-r} = a_r^{-1}(x_n - a_1x_{n-1} - \cdots - a_{r-1}x_{n-r+1} - b)$$

reproduces the sequence in the reverse direction.

Corollary 2 *An LFSR of length l generates only purely periodic sequences if the rightmost tap is set (that is, $a_l \neq 0$).*

1.9 Matrix generators over finite fields

A matrix generator over a field K is completely specified by an $r \times r$ matrix

$$A \in M_r(K)$$

(except for the choice of the start vector $x_0 \in K^r$). The objective of the present section is the characterization of the sequences with maximum period length.

In the polynomial ring $K[T]$ in one indeterminate T the set

$$\{\rho \in K[T] \mid \rho(A) = 0\}$$

is an ideal. Since $K[T]$ is a principal ring (even Euclidean) this ideal is generated by a unique monic polynomial μ. This polynomial is called the **minimal polynomial** of A. Since the matrix A is a zero of its own characteristic polynomial χ we have $\mu|\chi$. If A is invertible, then the absolute term of μ is $\neq 0$; otherwise μ would have the root 0, and A, the eigenvalue 0.

Lemma 4 *Let K be a field, $A \in GL_r(K)$, a matrix of finite order t, μ, the minimal polynomial of A, $s = \deg \mu$, $R := K[T]/\mu K[T]$, and $a \in R$, the residue class of T. Then:*

$$a^k = 1 \Longleftrightarrow \mu | T^k - 1 \Longleftrightarrow A^k = \mathbf{1}.$$

In particular $a \in R^\times$, t is also the order of a, and $\mu | T^t - 1$.

Proof. R is a K-algebra of dimension s. If $\mu = b_sT^s + \cdots + b_0$ (where $b_s = 1$), then

$$\mu - b_0 = T \cdot (b_sT^{s-1} + \cdots + b_1).$$

Since $b_0 \neq 0$, the residue class $T \bmod \mu$ is invertible, hence $a \in R^\times$. Since a^k is the residue class of T^k all the equivalences follow. $\diamond$

Corollary 1 *If K is a finite field with q elements, then*

$$t \leq \#R^\times \leq q^s - 1 \leq q^r - 1.$$

From now on let K be a finite field with q elements. Then also the group $GL_r(K)$ of invertible $r \times r$-matrices is finite. The vector space K^r consists of q^r vectors. We know already that every sequence from the matrix generator corresponding to $A \in GL_r(K)$ is purely periodic. One full cycle consists of the null vector $0 \in K^r$ alone. The remaining vectors in general distribute over several cycles. If s is the length of such a cycle, and x_0, the corresponding start vector, then $x_0 = x_s = A^s x_0$. Hence A^s has the eigenvalue 1, and consequently, A has as eigenvalue an s-th root of unity.

Maybe all vectors $\neq 0$ are in a single cycle of the maximum possible period length $q^r - 1$. In this case $A^s x = x$ for all vectors $x \in K^r$ if $s = q^r - 1$, but not for a smaller exponent > 0. Hence $t = q^r - 1$ is the order of A. This shows:

Corollary 2 *Let K be finite with q elements. Then:*

(i) *If the matrix generator for A and a start vector $\neq 0$ outputs a sequence of period s, then A has as eigenvalue an s-th root of unity.*

(ii) *If there is an output sequence of period length $q^r - 1$, then $t = q^r - 1$ is the order of A.*

Lemma 5 *Let K be a finite field with q elements, and $\varphi \in K[T]$ be an irreducible polynomial of degree d. Then $\varphi | T^{q^d-1} - 1$.*

Proof. The residue class ring $R = k[T]/\varphi K[T]$ is an extension field of degree $d = \dim_K R$, hence has $h := q^d$ elements, and R contains at least one zero a of φ, namely the residue class of T. Since each $x \in R^\times$ satisfies the equation $x^{h-1} = 1$ we conclude that a is also a zero of $T^{h-1} - 1$. Hence $\mathrm{ggT}(\varphi, T^{h-1} - 1)$ is not a constant. Since φ is irreducible $\varphi | T^{h-1} - 1$. $\diamond$

Definition Let K be a finite field with q elements. A polynomial $\varphi \in K[T]$ of degree d is called **primitive** if φ is irreducible and is not a divisor of $T^k - 1$ for $1 \leq k < q^d - 1$.

Theorem 1 *Let K be a finite field with q elements and $A \in GL_r(K)$. Then the following statements are equivalent:*

(i) *The matrix generator for A generates a sequence of period $q^r - 1$.*

(ii) *The order of A is $q^r - 1$.*

(iii) *The characteristic polynomial χ of A is primitive.*

Proof. "(i) $\Longrightarrow$ (ii)": See Corollary 2 (ii).

"(ii) $\Longrightarrow$ (iii)": In Corollary 1 we now have $t = q^r - 1$. Hence $\#R^\times = q^s - 1$, hence R is a field, and thus μ is irreducible. Moreover $s = r$, hence $\mu = \chi$, and μ is not a divisor of $T^k - 1$ for $1 \leq k < q^r - 1$ by Lemma 4. Therefore μ is primitive.

"(iii) $\Longrightarrow$ (i)": Since χ is irreducible, $\chi = \mu$. The residue class a of T is a zero of μ and has multiplicative order $q^r - 1$ by the definition of "primitive". Since taking the q-th power is an automorphism of the field R that fixes K elementwise all the r powers a^{q^k} for $0 \leq k < r$ are zeroes of μ, and they are all different. Therefore they must represent all the zeroes, and they all have multiplicative order $q^r - 1$. Hence A has no eigenvalue of lower order. By Corollary 2 (i) there is no shorter period. $\diamond$

For an LFSR take A as the companion matrix as in Section 1.7. Hence the characteric polynomial is $T^l - a_1 T^{l-1} - \cdots - a_l$.

Corollary 1 *An LFSR of length l generates a sequence of the maximum possible period length $2^l - 1$ if and only if its characteristic polynomial is primitive, and the start vector is $\neq 0$.*

This result reduces the construction of LFSRs that generate maximum period sequences to the construction of primitive polynomials over the field $\mathbb{F}_2$.

The special case of dimension $r = 1$ describes a multiplicative generator $x_n = ax_{n-1}$ over the finite field K with q elements. The corresponding 1×1 matrix $A = (a)$ is the multiplication by a. Thus a is the only eigenvalue, and $\chi = T - a \in K[T]$ is the characteristic polynomial. It is linear, hence irreducible. Since

$$\chi | T^k - 1 \Longleftrightarrow a \text{ is a zero of } T^k - 1 \Longleftrightarrow a^k = 1,$$

χ is primitive if and only if a is a generating element of the multiplicative group $K^\times$, hence a primitive element. This proves the following slight generalization of the corollary of Proposition 2:

Corollary 2 *The multiplicative generator over K with multiplier a generates a sequence of period $q - 1$ if and only if a is primitive and the start value is $x_0 \neq 0$.*

1.10 Statistical properties of LFSRs

The study of the statistical properties of LFSR sequences of maximum period $2^l - 1$, where l is the length of the LFSR, goes back to Golomb [2].

Here are some results:

1. Each full period contains exactly 2^{l-1} ones and $2^{l-1} - 1$ zeroes.

   Proof Each of the 2^l state vectors $\in \mathbb{F}_2^l$ (except 0) occurs exactly once, corresponding to the integers in the interval $[1 \ldots 2^l - 1]$. Of these integers 2^{l-1} are odd, the remaining ones are even, and their parities yield the exact output sequence of the LFSR.

2. A **run** in a sequence is a constant subsequence of maximum length.

   Example: $\ldots 0111110 \ldots$ is a run of ones of length 5.

   Noting that the pieces of length l of the LFSR sequence are exactly the different state vectors $\neq 0$ we immediately see that a full period contains:

   - no run of length $> l$,
   - exactly one run of 1's and no run of 0's of length l—otherwise the zero state vector would occur, or the all-1 state would occur more often than once,
   - exactly one run of 1's and exactly one run of 0's of length $l - 1$,
   - more generally exactly 2^{k-1} runs of 1's or 0's each of length $l - k$ for $1 \leq k \leq l - 1$,
   - in particular exactly 2^{l-1} runs of length 1, exactly half of them consisting of 0's or 1's.

3. For a periodic sequence $x = (x_n)_{n\in\mathbb{N}}$ in $\mathbb{F}_2$ of period s the **auto-correlation** w. r. t. the shift t is defined as

$$\begin{aligned}\kappa_x(t) &= \frac{1}{s} \cdot [\#\{n \mid x_{n+t} = x_n\} - \#\{n \mid x_{n+t} \neq x_n\}] \\ &= \frac{1}{s} \cdot \sum_{n=0}^{s-1} (-1)^{x_{n+t}+x_n}\end{aligned}$$

   (as in Part II for Boolean functions). Consider a sequence x generated by an LFSR of length l,

$$x_n = a_1 x_{n-1} + \cdots + a_l x_{n-l} \quad \text{for } n \geq l,$$

   and the sequence $y_n = x_{n+t} - x_n$ of its differences. This sequence is obviously generated by the same LFSR. If the start values $y_0, \ldots, y_{l-1}$ are all 0, then the y sequence is constant $= 0$, the t-th state vector $x_{(t)} = x_{(0)}$, hence t is a multiple of the period, and $\kappa_x(t) = 1$. Otherwise—and if x has the maximum possible period $s = 2^l - 1$—a full period of y consists of exactly 2^{l-1} ones and $2^{l-1} - 1$ zeroes by Remark 1. Thus

$$\kappa_x(t) = \begin{cases} 1, & \text{if } s|t, \\ -\frac{1}{s}, & \text{else.} \end{cases}$$

Hence the auto-correlation is uniformly small, except for shifts by a multiple of the period.

Golomb called these statements the three randomness postulates. They tell us that the sequence is very uniformly distributed. Therefore electrical engineers are fond of LFSR sequences of maximum period, and call them PN sequences (= pseudo-noise sequences).

Executing the Sage code sample 1.4 with the parameter 1024 instead of 20 yields the output (without parentheses and commas):

```
11001000110101100011001111000000 00111011100011100000100011101111
01001001111001011011110010111001 00010010110001100111001111010111
11000100011000001110011000010111 01101010101110110001010111011000
11110000010000100010111100011110 10100111000001111000100001011000
01010101000101111110110011011101 11001001110111110001011000100010
11100100101111110011011001010011 00001100100001100110100011100100
11101000100101110110011011001010 11011100100110111001011100000011
00100010111101111000110000010001 01110100001110011111101000100101
00111010001111000100000000110110 10000101110101110001100000010001
11011011011110111001000110101001 10001111110110101010011111100001
11101110111101011001010110001010 00000100001001100110001110100110
00010100101110100000010101100100 10010110101011111110111111011101
11001010010100010010110111111110 10100101001111110110100100010001
10111100011001111001011111010110 01110111010100100010100101101111
01100111011000000111011111010000 11011101111111110000010001000100
10010111111110101011101110111111 01110010110000010001111001100111
```

The visualization in Figure 1.10 shows that the output looks quite random, at least at first sight.

By the way the LFSR of this example generates a sequence of maximum period $2^{16} - 1 = 65535$ since its characteristic polynomial

$$T^{16} + T^{14} + T^{13} + T^{11} + 1 \in \mathbb{F}_2$$

is primitive.

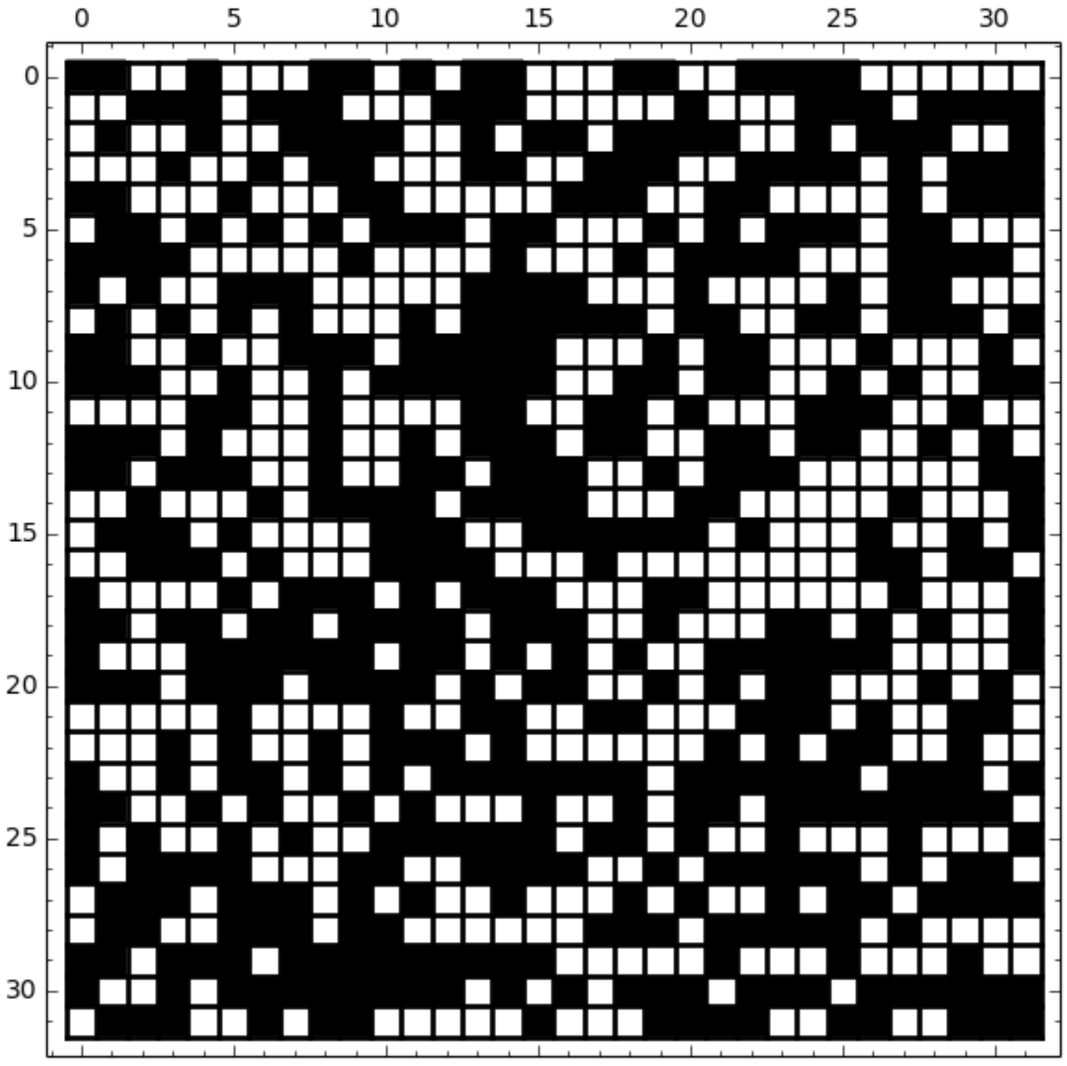

Figure 1.10: An LFSR sequence