Chapter 2

Cryptanalysis of Pseudorandom Generators

We slightly enlarge the black box model of a pseudorandom generator, cf. Figure 1.5, to distinguish between secret and public parameters:

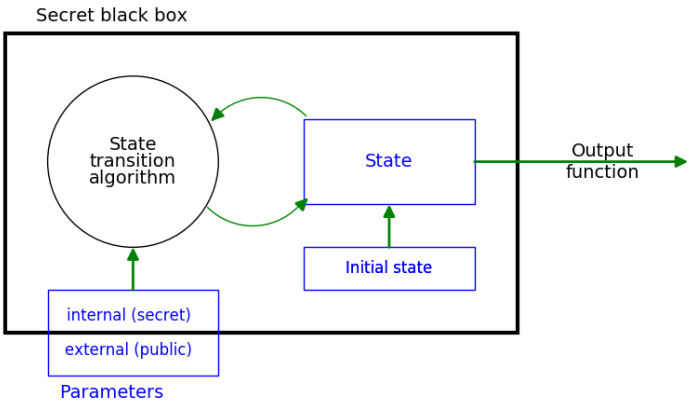

Figure 2.1: Pseudorandom generator (one element per state to be used as pseudorandom sequence

The black box hides an inner state that changes with each step by a given algorithm. This algorithm is controlled by parameters some of which are "public", but some of which are secret and serve as components of the key. The initial state (= start value) is a true random value and likewise secret. With each step the pseudorandom generator outputs a value, depending on its current inner state, until an exterior intervention stops it.

Cryptanalysis of pseudorandom generators assumes a known-plaintext attack. Thus the attacker is supposed to observe (or correctly guess) some elements of the output sequence. Her potential targets are the following

data:

- the secret internal parameters,
- the initial state,
- further elements of the output ("prediction problem").

2.1 The General Linear Generator

Remember that a general linear generator is characterized by

- a ring R and an R-module M as external parameters,
- a linear map $A\colon M \longrightarrow M$ as internal parameter,
- a sequence of vectors $x_n \in M$ as states and output elements,
- a vector $x_0 \in M$ as initial state,
- a recursive formula $x_n = Ax_{n-1}$ for $n \geq 1$ as state transition.

Remark (the trivial case): If A is known, then from each member x_r of the output sequence we may predict all of the following members $(x_n)_{n>r}$. Therefore this case lacks cryptological relevance. Note that calculating the sequence backwards, that is x_n for $0 \leq n < r$, is uniquely possible only if A is injective. But this effect doesn't rescue the cryptologic value of the generator. For simplicity in the following we usually treat forwards prediction only, assuming that an initial chunk $x_0, \ldots, x_{k-1}$ of the output sequence is known. However we should bear in mind that also backwards "prediction" might be an issue.

Assumption for the following considerations: R and M are known, A is unknown, and an initial segment $x_0, \ldots, x_{k-1}$ is given. To avoid trivialities we assume $x_0 \neq 0$. The *prediction problem* for this scenario is: Can the attacker determine $x_k, x_{k+1}, \ldots$?

Yes she can, provided she somehow finds a linear combination

$$x_k = c_1 x_{k-1} + \cdots + c_k x_0$$

with known coefficients $c_1, \ldots, c_k$. For then

$$\begin{aligned} x_{k+1} &= Ax_k = c_1 Ax_{k-1} + \cdots + c_k Ax_0 \\ &= c_1 x_k + \cdots + c_k x_1 \\ &\vdots \\ x_n &= c_1 x_{n-1} + \cdots + c_k x_{n-k} \quad \text{for all } n \geq k, \end{aligned}$$

and by this formula she gets the complete remaining sequence—without determining A (!). But how to find such a linear combination?

A simple example is periodicity: $x_n = x_{n-k}$ for all $n \geq k$. Linear algebra provides a more general solution. In the present abstract framework it is natural to assume M as Noetherian (usually the "proper" generalization of a finite-dimensional vector space). Then the ascending chain of submodules

$$Rx_0 \subseteq Rx_0 + Rx_1 \subseteq \ldots \subseteq M$$

is stationary: there is an r with $x_r \in Rx_0 + \cdots + Rx_{r-1}$. And this yields the linear relation we need; of course it is useful only when we succeed with explicitly determining the involved coefficients. Note that a finite module M—that we usually consider for random generation—is trivially Noetherian.

By this consideration we have shown:

Proposition 4 (Noetherian principle for linear generators) *Let R be a ring, M, an R-module, $A\colon M \longrightarrow M$ linear, and $(x_n)_{n\in\mathbb{N}}$ a sequence in M with $x_n = Ax_{n-1}$ for $n \geq 1$. Then for $r \geq 1$ the following statements are equivalent:*

(i) $x_r \in Rx_0 + \cdots + Rx_{r-1}$.

(ii) *There exist $c_1, \ldots, c_k \in R$ such that $x_n = c_1x_{n-1} + \cdots + c_rx_{n-k}$ for all $r \geq k$.*

If M is Noetherian, then an r with (i) *and* (ii) *exists.*

But how to explicitly determine the index k and the coefficients $c_1, \ldots, c_k$? Of course this can work only for rings R and modules M that admit explicit arithmetic operations.

In the following our main examples are: $R = K$ a finite field, or $R = \mathbb{Z}/m\mathbb{Z}$ a residue class ring of integers. In both cases we have a-priori knowledge on the number of true increments in the chain of submodules; that is, an explicit bound for r. If for example R is a field, then the number of proper steps is bounded by the vector space dimension $\dim M$. In the general case we have:

Proposition 5 (Krawczyk) *Let M be an R-module, and $0 \subset M_1 \subset \ldots \subset M_l \subseteq M$ be a properly increasing chain of submodules. Then $2^l \leq \#M$.*

This result is useful only for a finite module M. However this is the case we are mainly interested in when treating congruential generators. Then we may express it also as $l \leq \log_2(\#M)$. This is not too bad compared with the case field/vector space, both finite: $l \leq \mathrm{Dim}(M) \leq \log_2(\#M)/\log_2(\#R)$.

Proof. Let $b_i \in M_i - M_{i-1}$ for $i = 1, \ldots, l$ (where $M_0 = 0$). Then the subset

$$U = \{c_1b_1 + \cdots + c_lb_l \mid \text{all } c_i = 0 \text{ or } 1\} \subseteq M$$

consists of 2^l distinct elements. For if two of these expressions would represent the same element, their difference would have the form

$$e_1b_1 + \cdots + e_tb_t = 0 \quad \text{with } e_i \in \{0, \pm 1\},\ e_t \neq 0,$$

for some t with $1 \leq t \leq l$. From $e_t = \pm 1 \in R^\times$ we would derive the contradiction $b_t = -e_t^{-1}(e_1b_1 + \cdots + e_{t-1}b_{t-1}) \in M_{t-1}$. Hence $\#M \geq \#U = 2^l$. $\diamond$

2.2 Linear Generators over Fields

In this section we consider the special case where $R = K$ is a field and M a finite dimensional vector space over K (hence a Noetherian K-module).

Then we have to find the minimal k with

$$\dim(Kx_0 + \cdots + Kx_k) = \dim(Kx_0 + \cdots + Kx_{k-1})$$

and then to find the linear combination

$$x_k = c_1 x_{k-1} + \cdots + c_k x_0.$$

This is a standard exercise in linear algebra.

For a concrete calculation we chose a fixed basis $(e_1, \ldots, e_m)$ of M. Let

$$x_n = \sum_{i=1}^{m} x_{in} e_i$$

denote the corresponding basis representation. Since $\operatorname{rank}(x_0, \ldots, x_{k-1}) = k$, there is a set $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$ of indices with $\#I = k$ such that the matrix

$$X = (x_{ij})_{i \in I, 0 \le j < k} = \begin{pmatrix} x_{i_1 0} & \ldots & x_{i_1 k-1} \\ \vdots & & \vdots \\ x_{i_k 0} & \ldots & x_{i_k k-1} \end{pmatrix}$$

is invertible. The coefficients c_j in the relation

$$x_k = \sum_{j=0}^{k-1} c_j x_j,$$

are not yet known, we get them by substituting

$$\sum_{i=1}^{m} x_{ik} e_i = \sum_{j=0}^{k-1} \sum_{i=1}^{m} c_j x_{ij} e_i,$$

hence by the uniqueness of basis coefficients

$$x_{ik} = \sum_{j=0}^{k-1} x_{ij} c_j \quad \text{for all } i \in I,$$

or, in matrix notation,

$$\bar{x} = (x_{ik})_{i \in I} = X \cdot c.$$

The solution for the coefficients c_j is

$$c = X^{-1} \cdot \bar{x}.$$

This proves the first two statements of the following proposition that extends Proposition 4:

Proposition 6 *Under the assumptions of Proposition 4 let $R = K$ be a field and M be finite dimensional of dimension m. Then:*

(i) *The minimal k that fulfils the statements on r in Proposition 4 is the smallest index with $\dim(Kx_0 + \cdots + Kx_k) = k$, and $k \leq m$.*

(ii) *The coefficients $c_1, \ldots, c_k$ are determined by a system of linear equations with an invertible square coefficient matrix whose entries consist of basis coefficients of $x_0, \ldots, x_{k-1}$.*

(iii) *If $k = m$, then A is uniquely determined by the basis coefficients of $x_0, \ldots, x_k$.*

Proof. (iii) Let

$$X_1 = (x_m, \ldots, x_1),\ X_0 = (x_{m-1}, \ldots, x_0) \in M_m(K).$$

Then $X_1 = AX_0$ in matrix representation for the basis $(e_1, \ldots, e_m)$ of M. Since $\operatorname{rank} X_0 = m$ the matrix X_0 is invertible, and

$$A = X_1 X_0^{-1},$$

as claimed. ◇

If A is invertible, then we can determine the sequence (x_n) also in backwards direction as soon as we have a subsequence $x_t, \ldots, x_{t+m}$ of length $m + 1$ with $\operatorname{rank}(x_t, \ldots, x_{t+m-1}) = m$ at our disposition.

Example

For the special case of an r-step homogeneous linear congruential generator $x_n = a_1 x_{n-1} + \cdots + a_r x_{n-r}$ over $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ where p is prime we use the companion matrix

$$A = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ a_r & \ldots & a_2 & a_1 \end{pmatrix}, \quad \operatorname{Det} A = (-1)^r a_r.$$

In this case A is invertible if and only if $a_r \neq 0$, a condition we may assume without loss of generality—otherwise the recursion depth would be $< r$.

For predicting the sequence we need at most $r + 1$ state vectors, or $2r$ elements of the sequence:

Corollary 1 *An r-step homogeneous linear congruential generator with known prime module is predictable given the $2r$ elements $x_0, \ldots, x_{2r-1}$ of the output sequence.*

Corollary 2 *An LFSR of length l is predictable from the first $2l$ output bits.*

Corollary 3 *A homogeneous linear congruential generator with known prime module is predictable from x_0, x_1, an inhomogeneous one, from x_0, x_1, x_2, x_3.*

In the Section 2.4 we'll see that even x_0, x_1, x_2 suffice.

These results knock off LFSRs as sources of key bits for cryptological applications. Keeping the length secret is useless since the attacker may easily determine it by trial and error, putting up with a slight complication of the attack.

For linear congruential generators we might hope that keeping the module m secret (and maybe not choosing a prime) might erect a serious obstacle. However we'll also put this hope at rest in Section 2.5.

2.3 Cracking an LFSR Stream XOR Encryption

Let us break down the abstract setting of Section 2.2 to an explicit procedure for cracking an XOR cipher that uses an LFSR sequence as keystream. (This section doesn't depend on 2.1 or 2.2 but contains a direct approach.)

Consider a key bitstream $u_0, u_1, \ldots$ generated by an LFSR by the formula $u_n = s_1 u_{n-1} + \cdots + s_l u_{n-l}$. Assume a plaintext a is XOR encrypted using this key stream, resulting in the ciphertext c, where $c_i = a_i + u_i$ for $i = 0, 1, \ldots$ What are the prospects of an attacker who knows a chunk of the plaintext?

Well, assume she knows the first $l+1$ bits $a_0, \ldots, a_l$ of the plaintext. She immediately derives the corresponding bits $u_0, \ldots, u_l$ of the key stream, in particular the initial state of the LFSR. For the yet unknown coefficients s_i she knows a linear relation:

$$s_1 u_{l-1} + \cdots + s_l u_0 = u_l.$$

Each additional known plaintext bit yields one more relation, and having l relations, from $2l$ bits of known plaintext, the easy linear algebra over the field $\mathbb{F}_2$ finds a unique solution (in non-degenerate cases).

So assume we know the first $2l$ bits $u_0, \ldots, u_{2l-1}$ from an LFSR of length l. The state vector

$$u_{(i)} = (u_i, \ldots, u_{i+l-1}) \quad \text{for } i = 0, 1, \ldots$$

is the register content for step i (in reversed order compared with Figure 1.7). Thus the analysis focusses on the states, not directly on the output. The recursion in matrix form (for $n \geq l$) is

$$\begin{pmatrix} u_{n-l+1} \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \\ s_l & s_{l-1} & \ldots & s_1 \end{pmatrix} \begin{pmatrix} u_{n-l} \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix}$$

or more parsimoniously (the indices being substituted by $m = n - l + 1$)

$$u_{(m)} = S \cdot u_{(m-1)} \quad \text{for } m \geq 1$$

where S is the companion matrix. As a further step we collect l consecutive state vectors $u_{(i)}, \ldots, u_{(i+l-1)}$ in a state matrix

$$U_{(i)} = \begin{pmatrix} u_i & u_{i+1} & \ldots & u_{i+l-1} \\ u_{i+1} & u_{i+2} & \ldots & u_{i+l} \\ \vdots & \vdots & \ddots & \vdots \\ u_{i+l-1} & u_{i+l} & \ldots & u_{2l-2} \end{pmatrix}$$

and set $U = U_{(0)}$, $V = U_{(1)}$. This yields the formula

$$V = S \cdot U$$

that expresses the unknown coefficients $s_1, \ldots, s_l$ by the known plaintext bits $u_0, \ldots, u_{2l-1}$. Most notably it allows us to write down the solution immediately—provided that the matrix U is invertible:

$$S = V \cdot U^{-1}.$$

The matrix S explicitly displays the coefficients $s_1, \ldots, s_l$. We'll discuss the invertibility later on.

Example

Assume we are given a ciphertext:

```
10011100 10100100 01010110 10100110 01011101 10101110
01100101 10000000 00111011 10000010 11011001 11010111
00110010 11111110 01010011 10000010 10101100 00010010
11000110 01010101 00001011 11010011 01111011 10110000
10011111 00100100 00001111 01010011 11111101
```

We suspect that the cipher is XOR with a key stream from an LFSR of length $l = 16$. The context suggest that the text is in German and begins with the word "Treffpunkt" (meeting point). To solve the cryptogram we need 32 bits of plaintext, that is the first four letters only, presupposed that the theory applies. This gives 32 bits of the key stream:

```
01010100 01110010 01100101 01100110 = T r e f
10011100 10100100 01010110 10100110   cipher bits
-------- -------- -------- --------
11001000 11010110 00110011 11000000   key bits
```

Sage sample 2.1 determines the coefficient matrix. Its last row tells us that all $s_i = 0$ except $s_{16} = s_5 = s_3 = s_2 = 1$.

Now we know the LFSR and the initial state, and can reconstruct the complete key stream—yes, it is the same as in Section 1.10—and write down the plaintext (that by the way begins a bit differently from our guess).

We have shown that the cofficients are uniquely determined assuming the state matrix $U = U_{(0)}$ is invertible. As a consequence in this case the LFSR is completely known, and all output bits are predictable. We have yet to discuss the case where the matrix U is singular.

If one of the first l state vectors (= rows of the matrix U) is zero, then all following state vectors are zero too, and prediction is trivial.

Thus we may assume that none of these vectors are zero, but that they are linearly dependent (reinventing the Noetherian principle for this special

Sage Example 2.1 Determining a coefficient matrix

```
sage: l = 16
sage: kbits =
      [1,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0]
sage: ulist = []
sage: for i in range(0,l):
        state = kbits[i:(l+i)]
        ulist.append(state)
sage: U = matrix(GF(2),ulist)
sage: det(U)
1
sage: W = U.inverse()
sage: vlist = []
sage: for i in range(1,l+1):
        state = kbits[i:(l+i)]
        vlist.append(state)
sage: V = matrix(GF(2),vlist)
sage: S = V*W
sage: S
[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
[1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0]
```

scenario). Then there is a smallest index $k \geq 1$ such that $u_{(k)}$ is contained in the subspace spanned by $u_{(0)}, \ldots, u_{(k-1)}$, and we find coefficients $t_1, \ldots, t_k \in \mathbb{F}_2$ such that

$$u_{(k)} = t_1 u_{(k-1)} + \cdots + t_k u_{(0)}.$$

Then also $u_{(k+1)} = S \cdot u_{(k)} = t_1 S \cdot u_{(k-1)} + \cdots + t_k S \cdot u_{(0)} = t_1 u_{(k)} + \cdots + t_k u_{(1)}$,

and by induction we get

$$u_{(n)} = t_1 u_{(n-1)} + \cdots + t_k u_{(n-k)} \quad \text{for all } n \geq k.$$

This formula predicts all the following bits.

Discussion

- For a singular state matrix this consideration yields a shorter LFSR (of length $k < l$) that generates exactly the same sequence. Then our method doesn't determine the coefficients of the original register but nevertheless correctly predicts the sequence.
- If the bits the attacker knows aren't just the first ones but $2l$ contiguous ones at a later position, then the theorem yields only the prediction of the following bits. In the main case of an invertible state matrix U the LFSR is completely known and may be run backwards to get the previous bits. For a singular state matrix we achieve the same effect using the shorter LFSR constructed above.
- The situation where $2l$ bits of the key stream are known but at non-contiguous positions is slightly more involved. We get linear relations that contain additional (unknown) intermediate bits. If m is the number of these then we get $l + m$ linear equations for $l + m$ unknown bits.
- What if the length l of the LFSR is unknown? Exhaustively trying all values $l = 1, 2, 3, \ldots$ is nasty but feasible. A better approach is provided by the Berlekamp-Massey algorithm, see Section 3.3 that is efficient also without knowledge of l.

2.4 Linear Congruential Generators with Known Module

This section uses elementary methods only and is independent of the general theory from the preceding sections of Chapter 2.

Assume the parameters a and b of the linear congruential generator $x_n = ax_{n-1} + b \bmod m$ are unknown, whereas the module m is known.

We'll show that for predicting the complete output sequence we only need 3 successive elements x_0, x_1, x_2 of the sequence, even for a composite module m. Starting with the relation

$$x_2 - x_1 \equiv a(x_1 - x_0) \pmod{m}$$

we immediately get (assuming for the moment that $x_1 - x_0$ and m are coprime)

$$a = \frac{x_2 - x_1}{x_1 - x_0} \bmod m,$$

where the division is $\bmod m$ (using the extended Euclidean algorithm). The increment b is given by

$$b = x_1 - ax_0 \bmod m.$$

So we found the defining formula and may predict the complete sequence.

A typical tool for this simple case was the **sequence of differences**

$$y_i = x_i - x_{i-1} \quad \text{for } i \geq 1.$$

It follows the rule

$$y_{i+1} \equiv ay_i \pmod{m}.$$

Note that the y_i may be negative lying between the bounds $-m < y_i < m$. Since m is known we might replace them by $y_i \bmod m$, but this was irrelevant in the example, and for an unknown m—to be considered later on—it is not an option.

Lemma 6 (on the sequence of differences) *Assume the sequence (x_i) is generated by the linear congruential generator with module m, multiplier a, and increment b. Let (y_i) be the sequence of differences, $c = \gcd(m, a)$, and $d = \gcd(m, y_1)$. Then:*

(i) *The following statements are equivalent:*
  (a) *The sequence (x_i) is constant.*
  (b) $y_1 = 0$.
  (c) $y_i = 0$ *for all i.*

(ii) $\gcd(m, y_i) \mid \gcd(m, y_{i+1})$ *for all i.*

(iii) $d \mid y_i$ *for all i.*

(iv) *If* $\gcd(y_1, \ldots, y_t) = 1$ *for some* $t \geq 1$, *then* $d = 1$.
(v) $c|y_i$ *for all* $i \geq 2$.
(vi) *If* $\gcd(y_2, \ldots, y_t) = 1$ *for some* $t \geq 2$, *then* $c = 1$.
(vii) $m|y_i y_{i+2} - y_{i+1}^2$ *for all* i.
(viii) *If* $\tilde{a}, \tilde{m}$ *are integers,* $\tilde{m} \geq 1$, *with* $y_i \equiv \tilde{a} y_{i-1} \pmod{\tilde{m}}$ *for* $i = 2, \ldots, r$, *then* $x_i = \tilde{a} x_{i-1} + \tilde{b} \bmod \tilde{m}$ *for all* $i = 1, \ldots, r$ *with* $\tilde{b} = x_1 - \tilde{a} x_0 \bmod \tilde{m}$.

Proof. (i) Note that $y_i = 0$ implies that all following elements are 0.
(ii) If e divides y_i and m, then it also divides $y_{i+1} = a y_i + k_i m$.
(iii) is a special case of (ii).
(iv) follows from $d|\gcd(y_1, \ldots, y_t)$, and this, from (iii).
(v) Let $m = c\tilde{m}$ and $a = c\tilde{a}$. Then $y_{i+1} = c\tilde{a} y_i + k_i c\tilde{m}$, hence $c|y_{i+1}$ for $i \geq 1$.
(vi) follows from $c|\gcd(y_2, \ldots, y_t)$ and this, from (v).
(vii) $y_i y_{i+2} - y_{i+1}^2 \equiv a^2 y_i - a^2 y_i \pmod{m}$.
(viii) by induction: For $i = 1$ the assertion is the definition of $\tilde{b}$. For $i \geq 2$ we have

$$x_i - \tilde{a} x_{i-1} - \tilde{b} \equiv x_i - \tilde{a} x_{i-1} - x_{i-1} + \tilde{a} x_{i-2} \equiv y_i - \tilde{a} y_{i-1} \equiv 0 \pmod{\tilde{m}},$$

as claimed. ◇

The trivial case of a constant sequence merits no further care. However it shows that in general the parameters of a linear congruential generator are not uniquely determined by the output sequence. For the constant sequence may be generated with an arbitrary module m and an arbitrary multiplier a if only the increment is set to $b = -(a-1)x_0 \bmod m$. Even if m is fixed a is not uniquely determined, not even $a \bmod m$.

Previously we considered the case where y_1 and m are coprime, yielding $a = y_2/y_1 \bmod m$. In the general case it might happen that division $\bmod m$ is not unique. This happens if and only if m and y_1 have a non-trivial common divisor, hence $d = \gcd(m, y_1) > 1$. The **sequence of reduced differences** $\bar{y}_i = y_i/d$ (see (iii) in Lemma 6) then follows the recursive formula

$$\bar{y}_{i+1} \equiv \bar{a} \bar{y}_i \pmod{\bar{m}}$$

with the reduced module $\bar{m} = m/d$ and reduced multiplier $\bar{a} = a \bmod \bar{m}$, from which we get a unique $\bar{a} = \bar{y}_2/\bar{y}_1$. Setting $\tilde{a} = \bar{a} + k\bar{m}$ with an arbitrary integer k and $\tilde{b} = x_1 - \tilde{a} x_0 \bmod m$, from Lemma 6 (viii) we also get $x_i = \tilde{a} x_{i-1} + \tilde{b} \bmod m$ for all $i \geq 1$. This proves:

Proposition 7 *Assume the sequence* (x_i) *is generated by a linear congruential generator with known module* m, *but unknown multiplier* a *and increment* b. *Then the complete output sequence is predictable from its first three*

elements x_0, x_1, x_2. If the sequence (x_i) is not constant, then the multiplier a is uniquely determined up to a multiple of the reduced module $\bar{m}$.

Thus also in this situation we sometimes have to content ourselves with predicting the sequence without revealing the parameters used for its generation. Here is a simple concrete example: For $m = 24$, $a = 2k + 1$ with $k \in [0 \dots 11]$, $b = 12 - 2k \bmod 24$, and initial value $x_0 = 1$ we always get the sequence $(1, 13, 1, 13, \dots)$.

2.5 Linear Congruential Generators with Unknown Module

The attack on linear congruential generators surely becomes harder if the module m is kept secret and cannot be guessed in an obvious way. We assume that the attacker has a (short) subsequence $x_0, x_1, \ldots$ of the output sequence at her disposal.

Surprisingly it is easier to attack the multiplier first. The following proposition yields a "surrogate" value a' in a few steps. Note the Noetherian approach via the (implicit) formula $y_{t+1} \in \mathbb{Z}y_1 + \cdots + \mathbb{Z}y_t$, the principal ideal generated by the integer $\gcd(y_1, \ldots, y_t)$.

Proposition 8 (Plumstead-Boyar) *Let (y_i) be the sequence of differences of the linear congruential generator with generating function $s(x) = ax + b \bmod m$, $m \geq 2$, and initial value x_0. Let $y_1 \neq 0$ and t be the smallest index such that $e = \gcd(y_1, \ldots, y_t) \mid y_{t+1}$. Then:*

(i) $t < 1 + \log_2 m$.

(ii) *If $e = c_1 y_1 + \cdots + c_t y_t$ with $c_i \in \mathbb{Z}$ and $a' = (c_1 y_2 + \cdots + c_t y_{t+1})/e$, then $a' \in \mathbb{Z}$ and*

$$y_{i+1} \equiv a' y_i \pmod{m} \quad \text{for all } i.$$

(iii) *If $b' = x_1 - a' x_0$, then*

$$x_i = a' x_{i-1} + b' \bmod m \quad \text{for all } i.$$

Proof. (i) If $e_j = \gcd(y_1, \ldots, y_j)$ doesn't divide y_{j+1}, then $e_{j+1} \leq e_j/2$. Since $e_1 = |y_1| < m$ we conclude $e = e_t < m/2^{t-1}$, hence $t - 1 < \log_2 m$.

(ii) We have

$$ae = c_1 a y_1 + \cdots + c_t a y_t \equiv c_1 y_2 + \cdots + c_t y_{t+1} = a' e \pmod{m}.$$

The greatest common divisor d of m and y_1 divides e by Lemma 6, hence also $d = \gcd(m, e)$. We divide the congruence first by d:

$$a \frac{e}{d} \equiv a' \frac{e}{d} \pmod{\bar{m}}$$

with the reduced module $\bar{m} = m/d$. Since e/d and $\bar{m}$ are coprime we may divide by e/d:

$$a \equiv a' \pmod{\bar{m}}, \quad a = a' + k\bar{m}.$$

Hence $y_{i+1} \equiv a y_i = a' y_i + k y_i \bar{m} \pmod{m}$. From $d \mid y_i$ follows $y_i \bar{m} \equiv 0$, hence $y_{i+1} \equiv a' y_i \pmod{m}$.

(iii) is an immediate consequence of Lemma 6 (viii). ◇

Examples

1. Let $m = 8397$, $a = 4381$, $b = 7364$ [Reeds 1977]. Generate

$$\begin{array}{lll} x_0 = 2134 & & \\ x_1 = 2160 & y_1 = 26 & e_1 = 26 \\ x_2 = 6905 & y_2 = 4745 & e_2 = 13 \\ x_3 = 3778 & y_3 = -3127 & e_3 = 1 \\ x_4 = 8295 & y_4 = 4517 & \end{array}$$

   We get $c_1 = 87542$, $c_2 = -481$, $c_3 = -1$, and $a' = 416881843$.

2. Let $m = 2^q + 1$, $a = 2^{q-1}$, $b = 2^q$, and $x_0 = 0$. By the corollary of the following Lemma 7 we have $y_i = (-1)^{i-1} \cdot 2^{q-i+1}$ for $i = 1, \ldots, q+1$, and thus $e_i = 2^{q-i+1}$. Hence $t = q + 1$. Thus the upper bound for t in Proposition 8 is sharp, and indeed we need the $q + 3$ elements x_0 to x_{q+2} of the output sequence to determine the surrogate multiplier a'.

Lemma 7 *Let the sequence (c_i) in $\mathbb{Z}$ be defined by $c_0 = 0$, $c_i = 2^{i-1} - c_{i-1}$ for $i \geq 1$. Then*

(i) $c_i = \frac{1}{3} \cdot [2^i - (-1)^i]$ *for all i,*

(ii) $c_i - 2c_{i-1} = (-1)^{i-1}$ *for all $i \geq 1$.*

Proof. (i) follows by induction, (ii) by a direct calculation. ◇

Corollary 1 *Let (x_i) be the output sequence of the linear congruential generator with module $m = 2^q + 1$, multiplier $a = 2^{q-1}$, increment $b = 2^q$, and initial value $x_0 = 0$. Let (y_i) be the sequence of differences. Then*

(i) $x_i = c_i \cdot 2^{q-i+1}$ *for $i = 0, \ldots, q+1$,*

(ii) $y_i = (-1)^{i-1} \cdot 2^{q-i+1}$ *for $i = 1, \ldots, q+1$.*

Proposition 8 provides a surrogate multiplier in an efficient way. Now we need a procedure for determining the module m. We close in on it by "successive correcting". In step j we determine a new surrogate module m_j and a new surrogate multiplier a_j as follows:

- In the first step set $m_1 = \infty$ and $a_1 = a'$. [Calculating mod ∞ simply means calculating with integers, and $\gcd(c, \infty) = c$ for $c \neq 0$, but $= \infty$ for $c = 0$.]
- In step j, $j \geq 2$, let $y'_j := a_{j-1} y_{j-1} \bmod m_{j-1}$. Then set $m_j = \gcd(m_{j-1}, y'_j - y_j)$ and $a_j = a_{j-1} \bmod m_j$.

Thus in iteration step j we use the current surrogate values m_{j-1} and a_{j-1} for m and a and predict a value y'_j for y_j that we compare with the real (known) value y_j. If these two numbers differ, then their difference is a multiple of m. In this case we correct the surrogate values. We always have $m \mid m_j$. The corrected values don't invalidate the former calculations since $y_i \equiv a_j y_{i-1} \pmod{m_j}$ for $i = 2, \ldots, j$, and also $y_i \equiv a_j y_{i-1} \pmod{m}$ for all $i \geq 2$. Also the true sequence (x_i) always fulfils $x_i \equiv a_j x_{i-1} + b_j \pmod{m_j}$ for $i = 1, \ldots, j$ with $b_j = x_1 - a_j x_0$ by Lemma 6 (viii).

In Example 1 above we have

$$\begin{array}{lll} & m_1 = \infty & a_1 = 416881843 \\ y'_2 = 10838927918 & m_2 = 10838923173 & a_2 = 416881843 \\ y'_3 = 5420327549 & m_3 = 8397 & a_3 = 4381 \end{array}$$

The calculation for m_3 is

$$\gcd(10838923173, 5420330676) = 8397.$$

Since $m_3 \leq 2x_2$ we conclude that necessarily $m = m_3$, $a = a_3$, and $b = x_1 - ax_0 \bmod m = 7364$. Thus we found the true values after two correction steps, and we didn't need any further elements of the output sequence than the five we used for determining a'. Note the large intermediate results that suggest that in general the procedure relies on multi-precision integer arithmetic.

Does the procedure always terminate? At the latest when we reach the period of the sequence, that is after at most m steps, the complete sequence is predictable. However this bound is practically useless. Unfortunately it is tight: For arbitrary m let $a = 1$, $b = 1$, and $x_0 = 0$. Then $x_i = i$ and $y_i = 1$ for $i = 0, \ldots, m-1$. The initial value for the surrogate multiplier is $a' = 1$. The first false prediction is $y'_m = 1$ instead of the correct value $y_m = 1 - m$. The end is reached only after evaluating x_m. Although this worst case is easily recognized and might be treated separately it nevertheless hints at the difficulty of finding good general results. And indeed we don't know of any.

From a slightly different point of view we count the number of necessary correction steps where the surrogate module changes. For if $m_j \neq m_{j-1}$, then $m_j \leq m_{j-1}/2$. Let $m^{(0)} = \infty > m^{(1)} > \ldots$ be the sequence of *distinct* surrogate modules. Then

$$m^{(1)} = m_{j_1} = |y'_{j_1} - y_{j_1}| < a'|y_{j_1-1}| + m < m(a'+1),$$

$$m \leq m^{(j)} < \frac{m(a'+1)}{2^{j-1}},$$

hence always $j < 1 + \log_2(a'+1)$. This gives an upper bound for the number of necessary corrections. Joan Plumstead-Boyar described a variant of the

algorithm that results in a potentially smaller value of a', and eventually in the upper bound $2 + \log_2 m$ for the number of correction steps. However in general the algorithm doesn't involve that many corrections making this bound obsolete as a terminating criterion.

It seems that the search for theoretical results is a worthwile task. Could we exclude a (maybe small) class of (maybe bad anyway) linear congruential generators such that the majority of the remaining (interesting) generators obey a practically useful terminating criterion? I would expect such a result. Is there a way to control the distribution of the number of steps? Or at least the mean value?

Anyway the known results suffice to disqualify linear congruential generators for direct cryptographic application.

For an implementation of this algorithm in C see `https://www.staff.uni-mainz.de/pommeren/Cryptology/Bitstream/2_Analysis/LCGcrack.html`.

2.6 A General Prediction Method

The method of Boyar (née Plumstead) admits a broad generalization by the **BK algorithm** (named after Boyar and Krawczyk): It applies to recursive formulas that have an expression in terms of (unknown) linear combinations of known functions. A suitable language for its description is commutative algebra, that is, rings and modules.

So let R be a commutative ring (with $1 \neq 0$), and X, Z be R-modules. Let

$$\Phi^{(i)} : X^i \longrightarrow Z \quad \text{for } i \geq h$$

be a family of maps that we consider as known, and

$$\alpha : Z \longrightarrow X$$

be a linear map considered as secret. From these data we generate a sequence $(x_n)_{n\in\mathbb{N}}$ in X by the following algorithm, see Figure 2.2:

- Set $x_0, \ldots, x_{h-1} \in X$ as initial values.
- After generating $x_0, \ldots, x_{n-1}$ for some $n \geq h$ let

$$\begin{aligned} z_n &:= \Phi^{(n)}(x_0, \ldots, x_{n-1}) \in Z, \\ x_n &:= \alpha(z_n) \in X. \end{aligned}$$

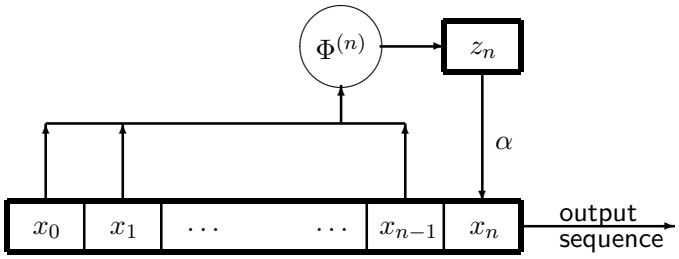

Figure 2.2: A very general generator

Here, in greater generality as before, we allow that each element of the sequence depends on *all* of its predecessors, that is, on the complete "past". A reasonable use for pseudorandom generation of course supposes that the $\Phi^{(i)}$ are efficiently computable. In the sample case $R = \mathbb{Z}/m\mathbb{Z}$, $X = R^k$, the cost should grow at most polynomially with $\log(m)$, h, and k.

Examples

1. The linear congruential generator: $R = \mathbb{Z}/m\mathbb{Z} = X$, $Z = R^2$, $h = 1$, $x_n = ax_{n-1} + b$,

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} x_{i-1} \\ 1 \end{pmatrix},$$

$$\alpha \begin{pmatrix} s \\ t \end{pmatrix} = as + bt.$$

2. The linear-inversive congruential generator: R, X, Z, h, α as above, $x_n = ax_{n-1}^{-1} + b$,

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} x_{i-1}^{-1} \bmod m \\ 1 \end{pmatrix}.$$

   (Set the first component to 0 if x_{i-1} is not invertible mod m.)

3. Congruential generators of higher degree: $R = \mathbb{Z}/m\mathbb{Z} = X$, $Z = R^{d+1}$, $h = 1$, $x_n = a_d x_{n-1}^d + \cdots + a_0$,

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} x_{i-1}^d \\ \vdots \\ x_{i-1} \\ 1 \end{pmatrix},$$

$$\alpha \begin{pmatrix} t_0 \\ \vdots \\ t_d \end{pmatrix} = a_d t_0 + \cdots + a_0 t_d.$$

4. Arbitrary congruential generators: $R = \mathbb{Z}/m\mathbb{Z}$, $x_n = s(x_{n-1})$, $h = 1$. If m is prime, then each function $s\colon R \longrightarrow R$ has an expression as a polynomial of degree $< m$, as in Example 3. For a more general module m we may use the basis $\{e_0, \ldots, e_{m-1}\}$ with $e_i(j) = \delta_{ij}$ of R^R. The basis representation is $s = \sum_{i=0}^{m-1} s(i)e_i$. Thus we set $X = R$, $Z = R^m$, and

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} e_0(x_{i-1}) \\ \vdots \\ e_{m-1}(x_{i-1}) \end{pmatrix},$$

$$\alpha \begin{pmatrix} t_0 \\ \vdots \\ t_{m-1} \end{pmatrix} = s(0)t_0 + \cdots + s(m-1)t_{m-1}.$$

5. For multistep congruential generators set h equal the recursion depth.

6. For nonlinear feedback shift registers see the next section 2.7.

For cryptanalysis we assume that the $\Phi^{(i)}$ are known, but α is unknown. (Later on, in the case $R = \mathbb{Z}/m\mathbb{Z}$, we'll also treat m as unknown.) The question is: Given an initial segment $x_0, \ldots, x_{n-1}$ $(n \geq h)$ of the output sequence, is there a method to predict the next element x_n?

To this end we consider the ascending chain $Z_h \subseteq Z_{h+1} \subseteq \ldots \subseteq Z$ of submodules with

$$Z_n = Rz_h + \cdots + Rz_n.$$

If $Z_n = Z_{n-1}$, then $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$ with $t_h, \ldots, t_{n-1} \in R$, and applying α we get the formula

$$x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$$

that predicts x_n from $x_0, \ldots, x_{n-1}$ without using knowledge of α.

If Z is a Noetherian R-module, then we encounter a stationary situation after finitely many steps: $Z_n = Z_l$ for $n \geq l$. Beginning with this index the complete sequence x_n is predictable by the following "algorithm":

1. Calculate $z_n = \Phi^{(n)}(x_0, \ldots, x_{n-1})$.
2. Find a linear combination $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$.
3. Set $x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$.

The Noetherian principle allows the prediction by a linear relation (that however might change from step to step).

To transform the "algorithm" into a true algorithm we need a procedure that explicitly finds a linear combination in step 2, solving a system of linear equations in Z.

For our standard example of a congruential generator with module $m = 8397$ (here assumed to be known), $x_0 = 2134$, $x_1 = 2160$, $x_2 = 6905$, we calculate

$$z_1 = \begin{pmatrix} 2134 \\ 1 \end{pmatrix}, \quad z_2 = \begin{pmatrix} 2160 \\ 1 \end{pmatrix}, \quad z_3 = \begin{pmatrix} 6905 \\ 1 \end{pmatrix}.$$

Trying to write z_3 as a linear combination $t_1 z_1 + t_2 z_2$ we get the system

$$\begin{aligned} 2134 t_1 + 2160 t_2 &= 6905 \\ t_1 + t_2 &= 1 \end{aligned} \tag{1}$$

of linear equations in $R = \mathbb{Z}/m\mathbb{Z}$. By elimination we find

$$26 t_1 = -4745 = 3652.$$

The inverse of 26 mod 8397 is 323, and thus we get $t_1 = 4016$, $t_2 = 4382$. This result correctly predicts $x_3 = 3778$.

Proceeding in this way we correctly predict the complete output sequence. The reason is that $Z_2 = Z$:

$$z_2 - z_1 = \begin{pmatrix} 26 \\ 0 \end{pmatrix}, \quad e_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \in Z_2, \quad e_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = z_1 - 2134 \cdot e_1 \in Z_2.$$

This example contains a partial answer to the question of when the chain of submodules Z_n becomes stationary: At least when $Z_l = Z$. But in a more general case this might never happen. Note also that from $Z_l = Z_{l+1}$ we can't conclude that the chain is stationary at Z_l—later on it could ascend again. For a bound on the number of proper increments see Proposition 5.

In each single loop of the prediction algorithm there are two possible alternative events:

- $z_n \notin Z_{n-1}$. Then predicting x_n is impossible, and Z_{n-1} properly extends to $Z_n = Z_{n-1} + Rz_n$.
- $z_n \in Z_{n-1}$. Then the algorithm correctly predicts x_n.

By Proposition 5 the first of these two events may happen at most $\log_2(\#Z)$ times (or $\operatorname{Dim} Z$ times if R is a field). For each of these events we need access to the next element x_n of the output sequence to get ahead. On first sight this looks disappointing, but some thought brings to mind that it is a realistic situtation for cryptanalysis: In the process of breaking a cipher the cryptanalyst works with a supposed key until she gets nonsense "plaintext". Then she tries to guess the following plaintext characters by context knowledge, corrects the supposed key and goes on with deciphering. Remember that we already encountered this effect in the last section. And note that the present algorithm is fairly simple but contents itself with predicting elements instead of determining the unknown parameters of the random generator.

2.7 Nonlinear Feedback Shift Registers

As another example of the general prediction method we consider arbitrary, not necessarily linear, feedback shift registers as illustrated in Figure 2.3.

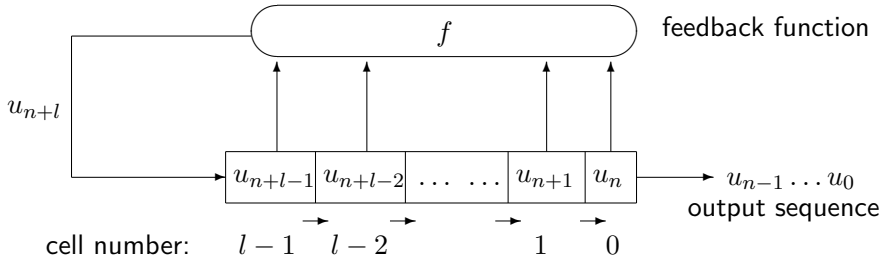

Figure 2.3: A feedback shift register (FSR) of length l

Here the feedback function is an arbitrary Boolean function $f\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$ whose algebraic normal form is a polynomial

$$f(y_1, \ldots, y_l) = \sum_{I \subseteq \{1,\ldots,l\}} a_I y^I \qquad \text{with } y^I = \prod_{j \in I} y_j.$$

We want to apply the prediction method with $R = X = \mathbb{F}_2$, $h = l$, $Z = \mathbb{F}_2^{2^l}$. For $i \geq l$

$$\Phi^{(i)}\colon \mathbb{F}_2^i \longrightarrow Z$$

is given by

$$z_i := \Phi^{(i)}(x_1, \ldots, x_i) = (y^I)_{I \subseteq \{1,\ldots,l\}} \qquad \text{with } y = (x_{i-l+1}, \ldots, x_i).$$

And finally we set

$$\alpha\colon Z \longrightarrow X, \quad \alpha((t_I)_{I \subseteq \{1,\ldots,l\}}) = \sum a_I t_I.$$

First we treat two concrete examples:

Examples

1. $l = 2$, $f = T_1 T_2 + T_2$. From the initial values $u_0 = 1$, $u_1 = 0$ we generate the sequence (manually or by Sage example 2.2)

   $$u_0 = 1,\ u_1 = 0,\ u_2 = 1,\ u_3 = 0,\ \ldots$$

   (that evidently has period 2). We have

   $$Z = \mathbb{F}_2^4, \qquad z_n = \begin{pmatrix} u_{n-1}u_{n-2} \\ u_{n-1} \\ u_{n-2} \\ 1 \end{pmatrix},$$

$$z_2 = \begin{pmatrix}0\\0\\1\\1\end{pmatrix}, \quad z_3 = \begin{pmatrix}0\\1\\0\\1\end{pmatrix}, \quad z_4 = \begin{pmatrix}0\\0\\1\\1\end{pmatrix} = z_2, \quad \ldots$$

From this the cryptanalyst recognizes the linear recursion

$$z_n = z_{n-2} = 0 \cdot z_{n-1} + 1 \cdot z_{n-2} \quad \text{for } n \geq 4.$$

She even recognizes the period, and correctly predicts

$$u_n = 0 \cdot u_{n-1} + 1 \cdot u_{n-2} = u_{n-2} \quad \text{for } n \geq 4.$$

Note that the very same sequence can be generated by a *linear* FSR of length 2. The analysis used the elements u_0, u_1, u_2, u_3.

2. $l = 3$, $f = T_1T_3 + T_2$. From the initial values $u_0 = 0$, $u_1 = 1$, $u_2 = 1$ we generate the elements (manually or by Sage example 2.3)

$$u_3 = 1,\ u_4 = 0,\ u_5 = 1,\ u_6 = 1,\ u_7 = 1,\ u_8 = 0,\ u_9 = 1,\ \ldots$$

of the output sequence. We have

$$Z = \mathbb{F}_2^8, \qquad z_n = \begin{pmatrix} u_{n-1}u_{n-2}u_{n-3}\\ u_{n-1}u_{n-2}\\ u_{n-1}u_{n-3}\\ u_{n-2}u_{n-3}\\ u_{n-1}\\ u_{n-2}\\ u_{n-3}\\ 1 \end{pmatrix},$$

$$z_3 = \begin{pmatrix}0\\1\\0\\0\\1\\1\\0\\1\end{pmatrix}, \quad z_4 = \begin{pmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{pmatrix}, \quad z_5 = \begin{pmatrix}0\\0\\0\\1\\0\\1\\1\\1\end{pmatrix}, \quad z_6 = \begin{pmatrix}0\\0\\1\\0\\1\\0\\1\\1\end{pmatrix}, \quad z_7 = \begin{pmatrix}0\\1\\0\\0\\1\\1\\0\\1\end{pmatrix} = z_3,$$

and so on. Hence the supposed linear recursion is

$$z_n = z_{n-4} \quad \text{for } n \geq 4,$$

again it reflects the periodicity. We get the correct prediction formula

$$u_n = u_{n-4} \quad \text{for } n \geq 4.$$

We needed the elements from u_0 to u_6; and again we found an "equivalent" LFSR, this time of length 4.

Sage Example 2.2 $f_1 = T_1T_2 + T_2$—monomials with exponent pairs `[1,1]` $\hat{=} 3$ and `[0,1]` $\hat{=} 1$, hence ANF bitblock `[0,1,0,1]`

```
f1 = BoolF([0,1,0,1],method="ANF")
y = f1.getTT(); y
[0, 1, 0, 0]
start = [0,1]
seq = fsr(f1,start,10); seq
[1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

Sage Example 2.3 $f_2 = T_1T_3 + T_2$—monomials with exponent triples `[1,0,1]` $\hat{=} 5$ and `[0,1,0]` $\hat{=} 2$, hence ANF bitblock `[0,0,1,0,0,1,0,0]`

```
f2 = BoolF([0,0,1,0,0,1,0,0],method="ANF")
y = f2.getTT(); y
[0, 1, 0, 0]
start = [1,1,0]
seq = fsr(f2,start,10); seq
[0, 1, 1, 1, 0, 1, 1, 1, 0, 1]
```

Since the dimension of Z grows exponentially with the register length the prediction algorithm reaches its limits soon. In the worst case the stationary state of the ascending chain of subspaces—and the needed linear relation—occurs only after 2^l steps. This observation would make shift registers up to a length of about 32 predictable with manageable cost using linear algebra in a binary vector space of dimension 2^{32}.

However in the examples we observed that the linear relation we found is nothing other than the formula for the final periodic repetition. This was not a fortunate coincidence but is a general phenomen that has an easy proof. For details see the paper [6]. Hence instead of solving large systems of linear equations we can apply an algorithm for period search that needs significantly less resources. This approach enables a realistic attack on shift registers of lengths up to about 80.

From a general point of view there is another objection against using arbitrary FSRs: The feedback function f depends on 2^l parameters. To have f efficiently computable and to deal with a manageable key space we have to restrict the choice of f, say by forcing "almost all" coefficients a_I in the ANF of the "admissible" feedback functions f to 0. Thus we specify a "small" set $\mathcal{M} \subseteq \mathcal{P}(\{1, \dots l\})$ a priori, and use only functions f whose ANF

$$f(x_1, \dots, x_l) = \sum_{I \in \mathcal{P}(\{1,\dots l\})} a_I x^I$$

has coefficients $a_I = 0$ for $I \notin \mathcal{M}$. Then the key space has size $2^{\#\mathcal{M}}$. However the coice of $\mathcal{M}$ is part of the encryption algorithm—in particular for a hardware FSR—, not a part of the key. KERCKHOFFS' principle warns us that the enemy will learn about $\mathcal{M}$ sooner or later. In the model of Figure 2.1 we treat the a priori "monomial supply" $\mathcal{M}$ as public parameter, and the concrete "monomial selection" I as secret parameter.

The necessity of choosing an efficiently computable feedback function and a manageable key space enforces restrictions that make the prediction method efficient too. Expressed in a somewhat sloppy way:

Proposition 9 *Each bit sequence that is generated by an FSR with efficiently computable feedback function is efficiently predictable.*

Our treatment of this problem was quite coarse. To derive mathematically correct statements there are two approaches:

1. Directly estimate the circuit complexity of the prediction algorithm by the circuit complexity of the feedback function.

2. Consider families of Boolean functions—that define families of FSRs—whose complexity grows polynomially with the register length, and show that the costs of the corresponding prediction procedures also grow at most polynomially.

For a comprehensive treatment see the cited paper [6].

We conclude that FSRs, no matter whether linear or nonlinear, are unsuited for generating pseudorandom sequences of cryptographic value—at least if naively applied. The method of BOYAR/KRAWCZYK breaks also nonlinear FSRs in realistic scenarios. And the result of BETH/DAI in Section 3.6 will open another promising way of predicting an FSR using the BERLEKAMP/MASSEY algorithm, see Section 3.3.

2.8 A General Congruential Generator

The prediction procedure becomes somewhat more involved when the module of a congruential generator is unknown. We abandon the general setting of commutative algebra and use special properties of the rings $\mathbb{Z}$ and $\mathbb{Z}/m\mathbb{Z}$, in particular the "canonical" representation of the residue classes of $\mathbb{Z}/m\mathbb{Z}$ by the subset $\{0, \ldots, m-1\} \subseteq \mathbb{Z}$.

Let $X = \mathbb{Z}^r$, $\bar{X} = (\mathbb{Z}/m\mathbb{Z})^r$, $Z = \mathbb{Z}^k$, $\bar{Z} = (\mathbb{Z}/m\mathbb{Z})^k$. The generator uses maps

$$\Phi^{(i)} : X^i \longrightarrow Z \quad \text{for } i \geq h,$$

$$\alpha : \bar{Z} \longrightarrow \bar{X} \quad \text{linear},$$

where α and m are unknown to the cryptanalyst. Identifying the residue classes with their canonical representants we consider $\bar{X}$ as the subset $\{0, \ldots, m-1\}^r$ of X. Then we generate a sequence by the same algorithm as in the previous Section 2.6, and call this procedure a **general congruential generator**, if the evaluation of the maps $\Phi^{(i)}$ is efficient with costs that depend at most polynomially on r, k, and $\log(m)$. In particular there is a bound M for the values of the $\Phi^{(i)}$ on $\{0, \ldots, m-1\}^{ri}$ that is at most polynomial in r, k, and $\log(m)$.

The cryptanalysis proceeds in two phases. In phase one we work over the ring $\mathbb{Z}$ and its quotient field $\mathbb{Q}$, and we determine a multiple $\hat{m}$ of the module m. In phase two we work over the ring $\mathbb{Z}/\hat{m}\mathbb{Z}$. Predicting x_n in this situation can trigger three different events:

- $z_n \notin Z_{n-1}$. Then the module Z_{n-1} (over $\mathbb{Q}$ or $\mathbb{Z}/\hat{m}\mathbb{Z}$) must be enlarged to Z_n, and no prediction is possible for x_n. The cryptanalyst needs some more plaintext.
- The prediction of x_n is correct.
- The prediction of x_n is false. Then the module $\hat{m}$ has to be adjusted.

In phase one Z_{n-1} is the vector space over $\mathbb{Q}$ that is spanned by $z_h, \ldots, z_{n-1}$ (omitting redundant z_i's).

Case 1: $z_n \notin Z_{n-1}$. Then set $Z_n = Z_{n-1} + \mathbb{Q}z_n$. This case can occur at most k times.

Case 2: [Linear relation] $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$. Then predict $x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ (as element of $\mathbb{Q}^r$).

Case 3: We have an analogous linear relation, but $\hat{x}_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ differs from x_n. Let $d \in \mathbb{N}$ be the common denominator of $t_h, \ldots, t_{n-1}$. Then

$$d\hat{x}_n = \alpha(dt_h z_h + \cdots + dt_{n-1} z_{n-1}) = \alpha(dz_n) = dx_n$$

in $\bar{X}$, that is mod m. This shows:

Lemma 8 *(*BOYAR*) The greatest common divisor $\hat{m}$ of the components of $d\hat{x}_n - dx_n$ in case 3 is a multiple of the module m.*

The result of phase one is a multiple $\hat{m} \neq 0$ of the true module m. The expense is:

- at most $k+1$ trials of solving a system of linear equations for up to k unknowns over $\mathbb{Q}$,
- one determination of the greatest common divisor of r integers.

Along the way the procedure correctly predicts a certain number of elements x_n, each time solving a system of linear equations of the same type.

How large can $\hat{m}$ be? For an estimate we need an upper bound M for all components of all $\Phi^{(i)}$ on $\{0, \ldots, m-1\}^{ri} \subseteq X^i$. We use HADAMARD's inequality: For arbitrary vectors $x_1, \ldots, x_k \in \mathbb{R}^k$ we have

$$|\operatorname{Det}(x_1, \ldots, x_k)| \leq \|x_1\|_2 \cdots \|x_k\|_2$$

where $\| \bullet \|_2$ is the Euclidean norm.

Lemma 9 *$\hat{m} \leq (k+1) \cdot m \cdot \sqrt{k^k} \cdot M^k$. In particular $\log(\hat{m})$ is bounded by a polynomial in k, $\log(m)$, $\log(M)$.*

Proof. The coefficient vector t is the solution of a system of at most k linear equations for the same number of unknowns. The coefficients z_i of this system are bounded by M. By HADAMARD's inequality and CRAMER's rule the numerators dt_i and denominators d of the solution are bounded by

$$\prod_{i=1}^{k} \sqrt{\sum_{j=1}^{k} M^2} = \prod_{i=1}^{k} \sqrt{kM^2} = \sqrt{k^k} \cdot M^k.$$

Hence the components of $d\hat{x}_n$ are bounded by

$$\|d\hat{x}_n\|_\infty = \|\sum dt_i x_i\|_\infty \leq \sqrt{k^k} \cdot M^k \cdot \sum \|x_i\|_\infty \leq km \cdot \sqrt{k^k} \cdot M^k$$

because m bounds the components of the x_i. We conclude

$$\|d\hat{x}_n - dx_n\|_\infty \leq km \cdot \sqrt{k^k} \cdot M^k + \sqrt{k^k} \cdot M^k \cdot m = (k+1) \cdot m \cdot \sqrt{k^k} \cdot M^k,$$

as claimed. $\diamond$

How does this procedure look in the example of an ordinary linear congruential generator? Here we have

$$z_1 = \begin{pmatrix} x_0 \\ 1 \end{pmatrix}, z_2 = \begin{pmatrix} x_1 \\ 1 \end{pmatrix}, z_3 = \begin{pmatrix} x_2 \\ 1 \end{pmatrix}, \ldots$$

If $x_1 = x_0$, then we have the trivial case of a constant sequence. Otherwise z_3 is a rational linear combination $t_1 z_1 + t_2 z_2$. Solving the system

$$\begin{aligned} x_0 t_1 + x_1 t_2 &= x_2, \\ t_1 + t_2 &= 1 \end{aligned}$$

yields

$$t = \frac{1}{d} \cdot \begin{pmatrix} -x_2 + x_1 \\ x_2 - x_0 \end{pmatrix} \quad \text{with } d = x_1 - x_0.$$

From this we derive the prediction

$$\hat{x}_3 = t_1 x_1 + t_2 x_2 = \frac{-x_2 x_1 + x_1^2 + x_2^2 - x_2 x_0}{x_1 - x_0} = \frac{(x_2 - x_1)^2}{x_1 - x_0} + x_2.$$

Hence $d(\hat{x}_3 - x_3) = (x_2 - x_1)^2 - (x_1 - x_0)(x_3 - x_2) = y_2^2 - y_1 y_3$ where (y_i) is the sequence of differences. If $\hat{x}_3 = x_3$, then we must continue this way. Otherwise we get, see Lemma 6,

$$m | \hat{m} = |y_1 y_3 - y_2^2|.$$

For our concrete standard example, where $x_0 = 2134$, $x_1 = 2160$, $x_2 = 6905$, $x_3 = 3778$, $y_1 = 26$, $y_2 = 4745$, $y_3 = -3127$, this general approach gives

$$\hat{m} = 4745^2 + 26 \cdot 3127 = 22596327.$$

A closer look, using Lemma 8 directly, even yields

$$t_1 = -\frac{365}{2},\ t_2 = \frac{367}{2},\ \hat{x}_3 = \frac{1745735}{2},\ \hat{m} = 2 \cdot (\hat{x}_3 - x_3) = 1738179.$$

In phase two of the algorithm we execute the same procedure but over the ring $\hat{R} = \mathbb{Z}/\hat{m}\mathbb{Z}$. However we can't simply reduce mod $\hat{m}$ the rational numbers from phase one. Hence we restart at z_h. Again we distinguish three cases for each single step:

Case 1: $z_n \notin \hat{Z}_{n-1} = \hat{R} z_h + \cdots + \hat{R} z_{n-1}$. Then set $\hat{Z}_n = \hat{Z}_{n-1} + \hat{R} z_n$ (and represent this $\hat{R}$-module by a non-redundant system $\{z_{j_1}, \ldots, z_{j_l}\}$ of generators where $z_{j_l} = z_n$). We can't predict x_n (but have to get it from somewhere else).

Case 2: $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$. Then predict $x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ (as an element of $\hat{X} = (\mathbb{Z}/\hat{m}\mathbb{Z})^r$). The prediction turns out to be correct.

Case 3: The same, but now the predicted value $\hat{x}_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ differs from x_n in $\hat{X}$. Then considering $\hat{x}_n - x_n$ as an element of $\mathbb{Z}^r$ we show:

Lemma 10 *In case 3 the greatest common divisor d of the coefficients of $\hat{x}_n - x_n$ is a multiple of m, but not a multiple of $\hat{m}$.*

Proof. It is a multiple of m since $\hat{x}_n \bmod m = x_n$. It is not a multiple of $\hat{m}$ since otherwise $\hat{x}_n = x_n$ in $\hat{X}$. $\diamond$

In case 3 we replace $\hat{m}$ by the greatest common divisor of d and $\hat{m}$ and reduce mod $\hat{m}$ all the former z_j. The lemma tells us that the new $\hat{m}$ is properly smaller than the old one.

By Lemma 9 case 3 can't occur too often, the number of occurences is polynomially in k, $\log(m)$, and $\log(M)$. If we already hit the true m this case can't occur any more. Case 1 may occur at most $\log_2(\#(\mathbb{Z}/\hat{m}\mathbb{Z})^k) = k \cdot \log_2(\hat{m})$ times in phase 2 by Proposition 5, and this bound is polynomial in k, $\log(m)$, and $\log(M)$.

> **Note.** There is a common aspect of phases one and two: In both cases we use the full quotient ring. The full quotient ring of $\mathbb{Z}$ is the quotient field $\mathbb{Q}$. In a residue class ring $\mathbb{Z}/m\mathbb{Z}$ the non-zero-divisors are exactly the elements that are coprime with m, hence the units. Thus $\mathbb{Z}/m\mathbb{Z}$ is its own full quotient ring.

For the concrete standard example we had $\hat{m} = 1738179$ after phase one, and now have to solve mod $\hat{m}$ the system (1) of linear equations. Since the determinant -26 is coprime with $\hat{m}$ we already have $Z_2 = \hat{R}^2$, and know that case 1 will never occur. The inverse of -26 is 66853 (in $\mathbb{Z}/\hat{m}\mathbb{Z}$), so from $-26\,t_1 = 4745$ we get $t_1 = 868907$. Hence $t_2 = 1 - t_1 = 869273$, and $\hat{x}_3 = 1_1 x_1 + t_2 x_2 = 3778$ is a correct prediction.

In the next step we calculate new coefficients t_1 and t_2 for the linear combination $z_4 = t_1 z_1 + t_2 z_2$. We solve (in $\mathbb{Z}/\hat{m}\mathbb{Z}$)

$$\begin{aligned} 2134\,t_1 + 2160\,t_2 &= 3778, \\ t_1 + t_2 &= 1. \end{aligned}$$

Eliminating t_2 yields $-26\,t_1 = 1618$, hence $t_1 = 401056$, and thus $t_2 = 1337124$, as well as $\hat{x}_4 = 1_1 x_1 + t_2 x_2 = 302190$. Since $x_4 = 8295$ we are in case 3 and must adjust $\hat{m}$:

$$\gcd(\hat{x}_4 - x_4, \hat{m}) = \gcd(293895, 1738179) = 8397.$$

Now $\hat{m} < 2x_2$. Thus from now on only case 2 will occur. This means that we'll predict all subsequent elements correctly.

A **prediction method** for a general congruential generator is an algorithm that gets the initial values $x_0, \ldots, x_{h-1}$ as input, then successively produces predictions of $x_h, x_{h+1}, \ldots$, and compares them with the true values; in the case of a mistake it adjusts the parameters using the respective true value.

A prediction method is **efficient** if

1. the cost of predicting each single x_n is polynomial in r, k, and $\log(m)$,

2. the number of false predictions is bounded by a polynomial in r, k, and $\log(m)$, as is the cost of adjusting the parameters in the case of a mistake.

The Boyar/Krawczyk algorithm that we considered in this section fulfils requirement 2. It also fulfils requirement 1 since solving systems of linear equations over residue class rings $\mathbb{Z}/m\mathbb{Z}$ is efficient (as shown in Section 9.2 of Part I). Thus we have shown:

Theorem 2 *For an arbitrary (efficient) general congruential generator the* Boyar/Krawczyk *algorithm is an efficient prediction method.*

A simple concrete example shows the application to a non-linear congruential generator. Suppose a quadratic generator of the form

$$x_n = ax_{n-1}^2 + bx_{n-1} + c \bmod m$$

outputs the sequence

$$x_0 = 63,\ x_1 = 96,\ x_2 = 17,\ x_3 = 32,\ x_4 = 37,\ x_5 = 72.$$

We set $X = \mathbb{Z}$, $Z = \mathbb{Z}^3$, $h = 1$. In phase one the vectors

$$z_1 = \begin{pmatrix} 3969 \\ 63 \\ 1 \end{pmatrix} z_2 = \begin{pmatrix} 9216 \\ 96 \\ 1 \end{pmatrix} z_3 = \begin{pmatrix} 289 \\ 17 \\ 1 \end{pmatrix}$$

span $\mathbb{Q}^3$ since the coefficient matrix is the Vandermonde matrix with determinant 119922. Solving

$$z_4 = \begin{pmatrix} 1024 \\ 32 \\ 1 \end{pmatrix} = t_1 z_1 + t_2 z_2 + t_3 z_3$$

yields

$$t_1 = \frac{160}{253}, \quad t_2 = -\frac{155}{869}, \quad t_3 = \frac{992}{1817},$$

with common denominator $d = 11 \cdot 23 \cdot 79 = 19987$. The algorithm predicts

$$\hat{x}_4 = \frac{1502019}{19987} \neq x_4.$$

Hence the first guessed module is

$$\hat{m} = d\hat{x}_4 - dx_4 = 762500,$$

and phase one is completed. Now we have to solve the same system of linear equations over $\mathbb{Z}/\hat{m}\mathbb{Z}$. Here the determinant is a zero divisor. We get two solutions, one of them being

$$t_1 = 156720, \quad t_2 = 719505, \quad t_3 = 648776.$$

Thus we predict the correct value

$$\hat{x}_4 = 156720 \cdot 96 + 719505 \cdot 17 + 648776 \cdot 32 \bmod 763500 = 37.$$

We are in case 2, and continue with predicting x_5: The system

$$z_5 = \begin{pmatrix} 1369 \\ 37 \\ 1 \end{pmatrix} = t_1 z_1 + t_2 z_2 + t_3 z_3$$

has two solutions, one of them being

$$t_1 = 2010\,, \quad t_2 = 558640\,, \quad t_3 = 201851\,,$$

hence

$$\hat{x}_5 = 136572\,, \quad \hat{x}_5 - x_5 = 136500\,.$$

We are in case 3 and adjust $\hat{m}$ to

$$\gcd(762500, 136500) = 500.$$

This exhausts the known values. Because all z_i are elements of $\hat{Z}_3 = \hat{R}z_1 + \hat{R}z_2 + \hat{R}z_3 \neq \hat{R}^3$ case 1 remains a possibility for the following steps. Since $x_0, \ldots, x_5$ are smaller than half the current module $\hat{m}$ also case 3 remains possible. In particular maybe we have to adjust the module furthermore.

Trying to predict x_6 we get $(\bmod 500)$

$$t_1 = 240\,, \quad t_2 = 285\,, \quad t_3 = 476\,, \quad x_6 = 117\,.$$

Exercise. What happens in the concrete standard example if after phase 1 we continue with the value $\hat{m} = 22596327$?

2.9 Analysis of Congruential Generators with Truncated Output

Cryptanalysis is significantly harder for pseudorandom generators that don't output all bits of the generated numbers. Then the sequence of differences is known at most approximately, greatest common divisors cannot be determined, and the algorithms of Plumstead-Boyar or Boyar/Krawczyk break down.

If the parameters of the pseudorandom generator are known, the cryptanalyst may try an exhaustion. The following consideration lacks mathematical strength. It doesn't presuppose that the pseudorandom generator is linear.

Suppose the generator produces n-bit integers but outputs only q bits (from fixed known positions) and suppresses $n - q$ bits. Then for each q-bit fragment of the output there exist 2^{n-q} possible complete values. In other words, a pseudorandom n-bit integer has the given bits at the given positions with probability $1/2^q$.

To continue we assume for simplicity that the q output bits are the most significant bits. So we decompose the integer x into $x = x_0 2^{n-q} + x_1$ where $0 \le x_1 < 2^{n-q}$. The value x_0, the first q bits, is known. The cryptanalyst runs through the 2^{n-q} different possibilities for x_1. For each choice of x_1 she forms $x = x_0 2^{n-q} + x_1$ and sets $y = s(x)$ with the generating function s of the pseudorandom generator. She compares y with the next q bits of the output that she knows. If the pseudorandom generator is statistically good, then the probability of a hit is $1/2^q$. Thus from the 2^{n-q} test values of x_0 there survive about 2^{n-2q} ones. In the case $q \ge \frac{n}{2}$ she expects exactly one hit. Otherwise she proceeds. After using k substrings of q bits the expected number of hits is about 2^{n-kq}. The expected necessary number of q-bit substrings exceeds k only if $kq \le n$, or $q \ge \frac{n}{k}$. For $q = \frac{1}{4}$ (as in the example $n = 32$, $q = 8$, that is an output of 8 bits of a 32-bit integer) four q-bit fragments suffice (where the exhaustion runs through 2^{24} integers). This trial-and-error procedure is manageable for small modules m. But note that the expense grows exponentially with m (assume the ratio $r = \frac{q}{n}$ of output bits is bounded away from 1).

For linear congruential generators with unknown module Frieze/Kennan/Lagarias, Håstad/Shamir, and J. Stern developed a better (probabilistic) procedure whose first step—finding the module—is summarized in the statement: *The cryptanalyst finds m with high probability if the generator outputs more than $2/5$ of the leading bits.* (without proof).

In the second step the cryptanalyst finds the multiplier a under the assumption that the module m is already known. In the third step she determines the complete integers x_i, or the differences y_i. Also with these

tasks she succeeds except for a negligeable subset of multipliers, and for the "good" multipliers she needs slightly more than one third of the leading bits of x_0, x_1, x_2, and x_3, to derive the complete integers. This enables her to predict all further output of the generator. A similar, somewhat weaker result by J. Stern holds for the case where instead of leading bits the generator outputs "inner bits" of the generated integers.

Thus the cryptanalysis of linear congruential generators reveals fundamental weaknesses, independently of the quality of their statistical properties.

Nevertheless linear congruential generators are useful for statistical applications. It is extremely unlikely that an application procedure "by accident" contains the steps that break a linear congruential generator and reveal its determinism. On the other hand linear congruential generators are disqualified for cryptographic applications once and for all, even with truncated output. However it is an open problem whether the objections also hold for a truncation strategy that outputs "very few" bits, say a quarter (note $\frac{1}{4} < \frac{2}{5}$), or only $\log\log(m)$ bits.

References

- J. Stern: Secret linear congruential generators are not cryptographically secure. FOCS 28 (1987), 421–426.
- Frieze/Håstad/Kannan/Lagarias/Shamir: Reconstructing truncated integer variables satisfying linear congruences. SIAM J. Comput. 17 (1988), 262–280.
- J. Boyar: Inferring sequences produced by a linear congruential generator missing low-order bits. J. Cryptology 1 (1989), 177–184.

2.10 Summary

In Sections 2.1 to 2.8 we developed a prediction method whose overall workflow is depicted in Figure 2.4.

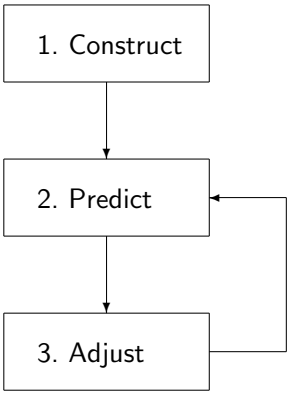

Figure 2.4: Workflow for prediction

1. By guessing plaintext the cryptanalyst finds subsequences of the key stream until she succeeds in constructing a linear relation for the state vectors (Noetherian principle).
2. Using this linear relation she predicts some more key bits.
3. If the predicted key bits are false (the plaintext ceases from making sense), then the cryptanalyst has to guess some more plaintext and to use it to adjust the parameters. Then she continues predicting bits.

This procedure is efficient for the "classical" pseudorandom generators, in particular for congruential generators—even with unknown module—and for feedback shift registers—even nonlinear ones. "Efficient" means that the computational cost is low, and also implies that the needed amount of known or correctly guessed plaintext is small.

One lesson learnt from these results is that for cryptographically secure pseudorandom generation we never should directly use the state of the generator as output. Rather we should insert a transformation between state and output that conceals the state—the output function of Figure 2.1. Section 2.9 illustrates that simply suppressing some bits —"truncating" or "decimating" the output—might be to weak as an output transformation. In the following section we'll learn about better output transformations.

There is a large twilight zone between pseudorandom generators that promise advantage to the cryptanalyst, and pseudorandom generators that

put the cipher designer's mind at ease. In any case we should prefer pseudorandom generators for which both of the procedures

- state transition,
- output function,

are nonlinear. The twilight zone where we don't know useful results on security contains (among others) quadratic congruential generators with slightly truncated output.

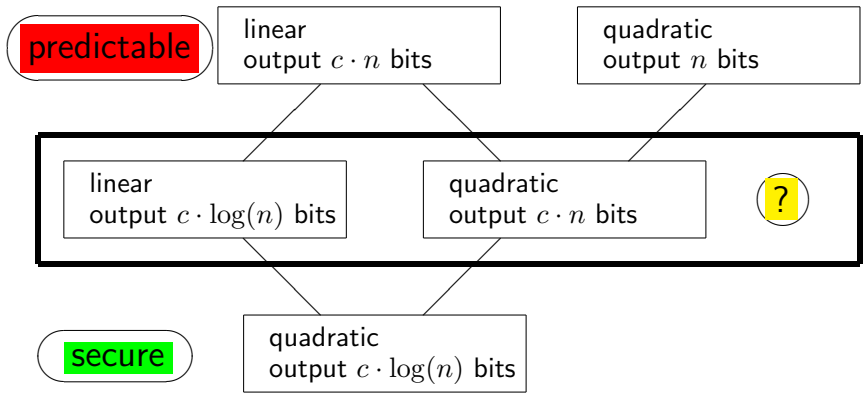

Figure 2.5: Predictable and secure congruential pseudorandom generators for n-bit integers (c a constant factor)

The following chapters present two approaches that are believed to lead to secure pseudorandom generators:

- combinations of LFSRs with a nonlinear output transformation (Chapter 3),
- nonlinear congruential generators with substantially truncated output (Chapter 4).