Chapter 4

Perfect Pseudorandom Generators

As we saw the essential cryptologic criterion for pseudorandom generators is unpredictability. In the 1980s cryptographers, guided by an analogy with asymmetric cryptography, found a way of modelling this property in terms of complexity theory: Prediction should boil down to a known "hard" algorithmic problem such as factoring integers or discrete logarithm. This idea established a new quality standard for pseudorandom generators, much stronger than statistical tests, but eventually building on unproven mathematical hypotheses. Thus the situation with respect to the security of pseudorandom generators is comparable to asymmetric encryption.

As an interesting twist it soon turned out that in a certain sense unpredictability is a universal property: For an unpredictable sequence there is *no efficient algorithm at all* that distinguishes it from a true random sequence, a seemingly much stronger requirement. See Theorem 4 (Yao's theorem). This universality justifies the denomination "perfect" for the corresponding pseudorandom generators. In particular there is no efficient statistical test that is able to distinguish the output of a perfect pseudorandom generator from a true random sequence. Thus, on the theoretical side, we have a very appropriate model for pseudorandom generators that are absolutely strong from a statistical viewpoint, and invulnerable from a cryptological viewpoint. In other words:

> *Perfect pseudorandom generators are cryptographically secure and statistically undistinguishable from true random sources—and are fit for* any *efficient application that needs random input.*
>
> *Presumably perfect pseudorandom generators exist, but there is no complete mathematical proof ot their existence.*

The first concrete approaches to the construction of perfect pseudorandom generators yielded algorithms that were too slow for most practical uses

(given the then current CPUs), the best known being the BBS generator (for Lenore Blum, Manuel Blum, Michael Shub). But modified approaches soon provided pseudorandom generators that are passably fast und nevertheless (presumably) cryptographically secure.

Looking back at Section 3.6 we might stumble over the apparent contradiction with the general "rule": "If a sequence has a short description (as the BBS sequence obviously has!), then it can't be random and even has a short description by a linear feedback shift register." In particular this would yield an efficient algorithm that distinguishes it from a random sequence. However as already stated in 3.6 this rule leaves a small loophole—small in relative terms but maybe wide enough in absolute terms. The notion of pseudorandomness tries to slip through this loophole, see Figure 4.1: Pseudorandom sequences

- are not random because they have a short description,
- can't nevertheless be efficiently predicted, or distinguished from random sequences.

For some more theoretical framework see the book [1].

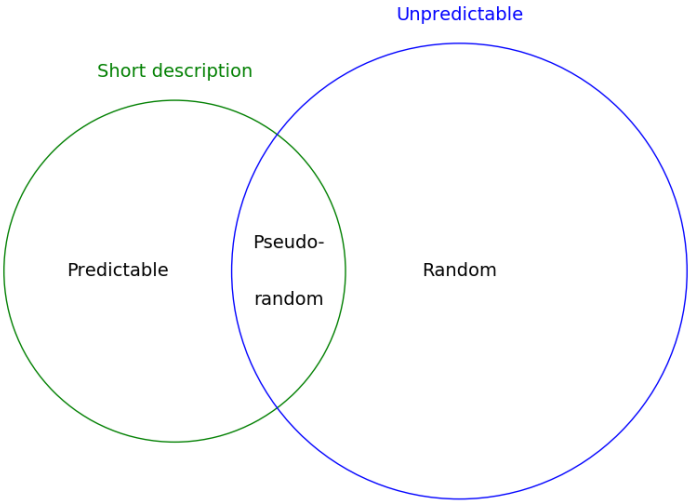

Figure 4.1: "Pseudorandom" as intersection of "Short description" and "Unpredictable"—the size of the areas is by far not to scale.

In the literature we find many tests for randomness:

- Marsaglia's diehard test suite, a collection of statistical tests that check the fitness of a sequence for statistical, but not cryptological applications,
- Golomb's postulates, see Section 1.10 above,

- the linear complexity profile, see Chapter 3 above,
- Maurer's universal test, see [5, 5.4.5],
- the LIL test, see [9].

By definition a perfect pseudorandom generator will pass all these tests. However in practice, since perfectness is an "asymptotic property" only, applying these tests can't harm.

4.1 The BBS Generator

As with the RSA cipher we consider an integer module n that is a product of two large prime numbers. For the BBS generator we choose a Blum **integer**, preferably—but not necessarily—a special or even superspecial one.

> Choosing n superspecial ensures that the sequence of states has a huge period. See the discussion in the Appendices 12–14 of Part III. However the following security proof doesn't depend on this property.

The BBS generator works in the following way that easily fits into the general framework of Figure 2.1: As a first step choose two large random Blum primes p and q, and form their product $n = pq$. The factors p and q are internal (secret) parameters, the product n may be treated as internal or external (public) parameter. As a second step choose a random integer, the "seed", s with $1 \leq s \leq n-1$, and coprime with n.

> The coprimality is efficiently tested with the Euclidean algorithm. If we catch an s not coprime with n, we have factorized n by hazard. This might happen, but is extremely unlikely, and can easily be captured at initialization time.

Then we proceed with generating a pseudorandom sequence: Take $x_0 = s \in \mathbb{M}_n$ as initial state, and form the sequence of inner states of the pseudorandom generator: $x_i = x_{i-1}^2 \bmod n$ for $i = 1, 2, 3, \ldots$ In each step output the last significant bit of the binary representation, that is $u_i = x_i \bmod 2$ for $i = 1, 2, 3 \ldots$, or in other words, the parity of x_i.

> If $x_i < \sqrt{n}$, then $x_i^2 \bmod n = x_i^2$, the integer square, so x_{i+1}^2 has the same parity as x_i. In order to avoid a constant segment at the beginning of the output, often the boundary areas $s < \sqrt{n}$, as well as $s > n - \sqrt{n}$, are excluded. However if we really choose s as a true random value, the probability for s falling into these boundary areas is extremely low. But to be on the safe side we may require $\sqrt{n} \leq s \leq n - \sqrt{n}$.
>
> If the seed s happens to be a quadratic non-residue, the sequence of inner states (the BBS sequence) has a preperiod of length 1.

Example

Of course an example with small numbers is practically irrelevant, but it illustrates the algorithm: Take $p = 7$, $q = 11$, $n = 77$, $s = 53$. Then $s^2 = 2809$, hence $x_1 = 37$, and $u_1 = 1$ since x_1 is odd. The following table shows the beginning of the sequence of states:

i	1	2	3	4	...
x_i	37	60	58	53	...
u_i	1	0	0	1	...

Since $x_4 = 53 = s$ the seed s happens to be a quadratic residue, and the BBS sequence has period 4. Therefore the output "pseudorandom" sequence (u_i) a forteriori has period 4.

Treating the primes p and q as secret is essential for the security of the BBS generator. They serve for forming n only, afterwards they may even be destroyed—in contrast with RSA there is no further use for them (except when you use SageMath, see below). Likewise all the non-output bits of the inner states x_i must be secret. Moreover there is no reason to reveal the product $n = pq$ even if the following security proof doesn't depend on the nondisclosure of n.

SageMath has an implementation of the BBS generator via the methods `random_blum_prime()` and `blum_blum_shub()`. The code sample 4.1 shows how to use them.

Sage Example 4.1 Generating a pseudorandom bit sequence by the BBS generator

```
sage: from sage.crypto.util import random_blum_prime
sage: from sage.crypto.stream import blum_blum_shub
sage: p = random_blum_prime(2^511, 2^512)
sage: q = random_blum_prime(2^511, 2^512)
sage: s = 11.powermod(248,p*q)  # a (not so random) example
sage: prseq = blum_blum_shub(1024,s,p,q)
```

Table 4.1 shows a Blum integer with 309 decimal places (or 1024 bits) that was an intermediate result of this program. Considering the progress of factoring algorithms we better should use Blum integers of at least 2048 bits.

```
 4506 15286 74466 50249 26225 14044 26383 22616 74480 10227
69340 10344 80414 96318 08671 21639 63710 30387 17602 25696
53909 02080 09976 45161 76261 91025 59480 62175 49124 86394
40823 70452 14981 62658 94574 67753 74945 83135 16199 61782
07594 51105 16833 44889 30109 66289 10763 64987 90309 41852
27681 66632 02722 32988 57145 85172 07427 89442 30004 31819
83739 34537
```

Table 4.1: A 1024 bit Blum integer

Table 4.2 shows the resulting bitsequence. Be warned that the SageMath

output is of type `StringMonoidElement`. For further use in a stream cipher it might be necessary to convert it to a bitblock or bitstring.

```
1000 1111 1001 0101 1001 0111 0011 0100 0010 1000 1100 0001
1010 0101 1110 1001 1010 1001 0110 0010 1010 1010 0111 0111
1000 1010 1000 1101 1111 1101 1010 1100 1100 0001 0101 1001
0111 1111 0001 0100 1010 0000 1100 1010 0101 1000 1110 0000
0001 1011 0100 0100 1010 0010 1010 1010 0110 1001 0111 1100
1011 0010 0011 0100 1101 1001 0101 0100 0111 0100 0010 0111
1101 1000 0010 0111 1000 0110 1110 0111 1110 1101 0110 1000
0001 0011 1111 0011 0011 0101 0001 0001 1010 0110 0101 1000
1010 1100 1011 0011 1111 1000 1001 0100 0001 1110 1111 1111
1001 0000 0010 0000 0111 0111 1001 0001 1111 0100 1010 0011
1000 0111 1100 0000 1011 0110 1011 1010 0111 0100 1110 1001
1001 0101 0011 1000 0010 0011 1010 1001 1100 0010 1111 1001
1010 1001 0110 0011 1001 0100 1000 1111 1001 1001 0010 1000
0111 0110 1101 0011 0110 0010 1110 0010 0000 1100 1011 1111
0011 0010 0110 1110 1000 1000 1110 1110 0011 0010 0100 0100
1101 1000 0011 0010 1000 1110 1000 1101 1010 0001 0011 1100
1001 0110 1010 0000 0000 0000 1011 0111 1010 0010 1100 1010
0100 0010 0010 0010 0010 1011 0100 0000 1100 1010 1101 0000
1101 1111 0011 0001 1000 0000 0111 0111 1110 1111 0011 1011
1111 0001 0010 1000 0110 1011 0111 0011 1111 1011 0101 0100
0110 1111 1111 0011 1011 0000 1010 0010 1100 0010 1001 0101
1110 1001 1001 1001
```

Table 4.2: 1024 "perfect" pseudorandom bits. Note that generating 1024 pseudorandom bits from a 1024-bit random integer isn't worth the effort. However we could continue this sequence much further and generate, say, 2^{30} pseudorandom bits.

Figure 4.2 gives an optical impression of the randomness of this sequence, and Figure 4.3 of its linearity profile.

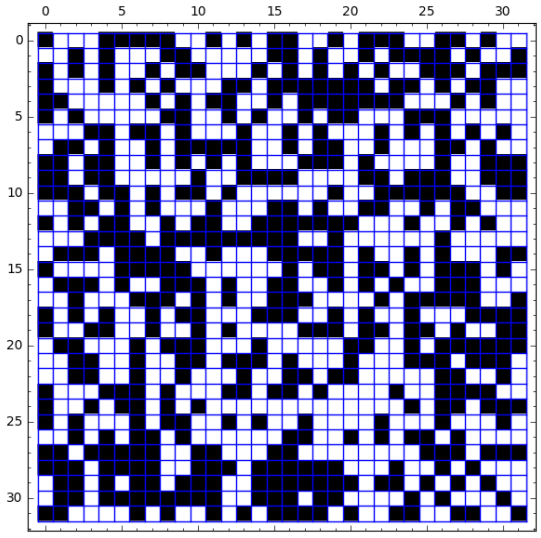

Figure 4.2: Visualization of a “perfect” pseudorandom sequence

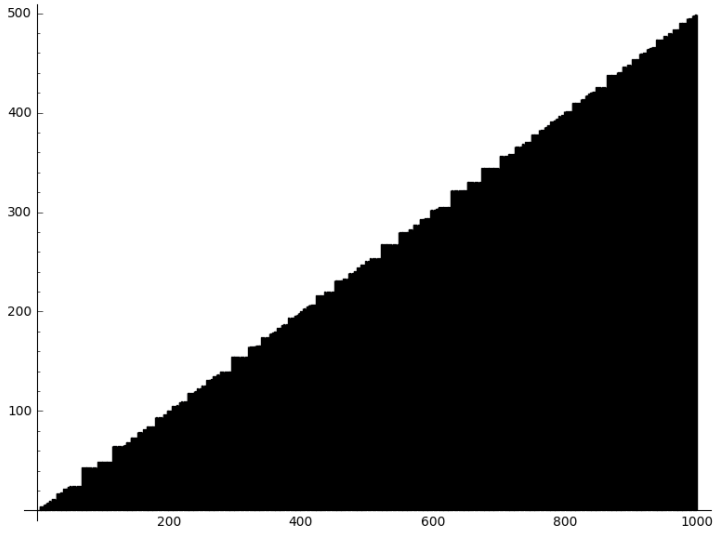

Figure 4.3: Linearity profile of a “perfect” pseudorandom sequence

4.2 The BBS Generator and Quadratic Residuosity

Given a seed $s \in \mathbb{M}_n^+$ the BBS generator outputs a bit sequence $(b_1(s), \ldots, b_r(s))$—by the way the same sequence as the seed $s' = \sqrt{s^2} \bmod n$ that is a quadratic residue. A probabilistic circuit (see Appendix B of Part III)

$$C \colon \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$$

has an ε-advantage for **BBS extrapolation** with respect to n if

$$P(\{(s,\omega) \in \mathbb{M}_n \times \Omega \mid C(b_1(s), \ldots, b_r(s), \omega) = \mathrm{lsb}(\sqrt{s^2} \bmod n)\}) \geq \frac{1}{2} + \varepsilon.$$

In other words: The algorithm implemented by C "predicts" (or extrapolates) the bit preceding a given subsequence with ε-advantage.

> If we seed the generator with a quadratic residue s, then C outputs the parity of s (with ε-advantage). If fed with a later segment $(b_{i+1}, \ldots, b_{i+r})$ (with $i \geq 1$) of a BBS output C extrapolates the preceding bit b_i.

In the following lemmas and proposition let τ_t be the maximum expense of the operation $xy \bmod n$ where n is a t-bit integer and $0 \leq x, y < n$. We know that $\tau_t = \mathrm{O}(t^2)$ (and even know an exact upper bound for the circuit size).

Lemma 16 *Let n be a* Blum *integer $< 2^t$. Assume the probabilistic circuit $C \colon \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$ has an ε-advantage for BBS extrapolation with respect to n. Then there is a probabilistic circuit $C' : \mathbb{F}_2^t \times \Omega \longrightarrow \mathbb{F}_2$ of size $\#C' \leq \#C + r\tau_t + 4$ that has an ε-advantage for deciding quadratic residuosity for $x \in \mathbb{M}_n^+$.*

Proof. First we compute the BBS sequence $(b_1, \ldots, b_r)$ for the seed $s \in \mathbb{M}_n^+$ at an expense of $r\tau_t$. Then C computes the bit $\mathrm{lsb}(\sqrt{s^2} \bmod n)$ with advantage ε. Therefore setting

$$C'(s,\omega) := \begin{cases} 1 & \text{if } C(b_1, \ldots, b_r, \omega) = \mathrm{lsb}(s), \\ 0 & \text{otherwise,} \end{cases}$$

we decide the quadratic residuosity of s with ε-advantage by the corollary of Proposition 24 in Appendix A.11 of Part III. The additional costs for comparing bits are at most 4 additional nodes in the circuit. $\diamond$

Now let $C \colon \mathbb{F}_2^t \times \Omega \longrightarrow \mathbb{F}_2$ be an arbitrary probabilistic circuit. Then for $m \geq 1$ we define the **m-fold circuit** by

$$C^{(m)} \colon \mathbb{F}_2^t \times \Omega^m \longrightarrow \mathbb{F}_2,$$

$$C^{(m)}(s, \omega_1, \ldots, \omega_m) := \begin{cases} 1 & \text{if } \#\{i \mid C(s, \omega_i) = 1\} \geq \frac{m}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

So this circuit represents the "majority decision". Its implementation consists of m parallel copies of C, one integer addition of m bits, and one comparision of $\lceil^2 \log m\rceil$-bit integers, hence by Appendix B.3 of Part III its size is

$$\#C^{(m)} \leq r \cdot \#C + 2m^2.$$

Lemma 17 (Amplification of advantage) *Let $A \subseteq \mathbb{F}_2^t$, and let C be a circuit that computes the Boolean function $f : A \longrightarrow \mathbb{F}_2$ with an ε-advantage. Let $m = 2h + 1$ be odd.*

Then $C^{(m)}$ computes the function f with an error probability of

$$\leq \frac{(1 - 4\varepsilon^2)^h}{2}.$$

For each $\delta > 0$ there is an

$$m \leq 3 + \frac{1}{2\delta\varepsilon^2}$$

such that $C^{(m)}$ computes the function f with an error probability δ.

Proof. The probability that C gives a correct answer is

$$p := P(\{(s, \omega) \in A \times \Omega \mid C(s, \omega) = f(s)\}) \geq \frac{1}{2} + \varepsilon.$$

Since enlarging ε tightens the assertion we may assume that $p = \frac{1}{2} + \varepsilon$. The complementary value $q := 1 - p = \frac{1}{2} - \varepsilon$ equals the probability that C gives a wrong answer. Hence the probability of getting exactly k correct answers from m independent invocations of C is $\binom{m}{k} p^k q^{m-k}$. Thus the error probability we search is

$$P(\{(s, \omega_1, \ldots, \omega_m) \in A \times \Omega^m \mid C^{(m)}(s, \omega_1, \ldots, \omega_m) = f(s)\})$$

$$\begin{aligned}
&= \sum_{k=0}^{h} \binom{m}{k} (\tfrac{1}{2} + \varepsilon)^k (\tfrac{1}{2} - \varepsilon)^{m-k} \\
&= (\tfrac{1}{2} + \varepsilon)^h (\tfrac{1}{2} - \varepsilon)^{h+1} \cdot \sum_{k=0}^{h} \binom{m}{k} (\tfrac{1}{2} + \varepsilon)^{k-h} (\tfrac{1}{2} - \varepsilon)^{h-k} \\
&= (\tfrac{1}{4} - \varepsilon^2)^h \cdot (\tfrac{1}{2} - \varepsilon) \cdot \underbrace{\sum_{k=0}^{h} \binom{m}{k} \underbrace{\left(\frac{\frac{1}{2} - \varepsilon}{\frac{1}{2} + \varepsilon}\right)^{h-k}}_{\leq 1}}_{\leq 2^{m-1} = 4^h} \\
&\leq (1 - 4\varepsilon^2)^h
\end{aligned}$$

which proves the first statement.

For an error probability δ a sufficient condition is:

$$\begin{aligned}(1-4\varepsilon^2)^h &\leq 2\delta,\\ h\cdot\ln(1-4\varepsilon^2) &\leq \ln 2+\ln\delta,\\ h &\geq \frac{\ln 2+\ln\delta}{\ln(1-4\varepsilon^2)}.\end{aligned}$$

Therefore we choose

$$h := \left\lceil \frac{\ln 2+\ln\delta}{\ln(1-4\varepsilon^2)} \right\rceil. \tag{1}$$

Then the error probability of $C^{(m)}$ is at most δ, and

$$\begin{aligned}h &\leq 1+\frac{\ln 2+\ln\delta}{\ln(1-4\varepsilon^2)} = 1+\frac{\ln\frac{1}{\delta}-\ln 2}{\ln\frac{1}{1-4\varepsilon^2}}\\ &\leq 1+\frac{\frac{1}{\delta}-1-\ln 2}{4\varepsilon^2} \leq 1+\frac{1}{4\delta\varepsilon^2},\end{aligned}$$

proving the second statement. ◇

By the way the size of $C^{(m)}$ is

$$\#C^{(m)} \leq \left[3+\frac{1}{2\delta\varepsilon^2}\right]\cdot\#C + 2\cdot\left[3+\frac{1}{2\delta\varepsilon^2}\right]^2.$$

Merging the two lemmas we get:

Proposition 13 *Let n be a* Blum *integer $< 2^t$. Assume the probabilistic circuit $C : \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$ has an ε-advantage for BBS extrapolation with respect to n. Then for each $\delta > 0$ there is a probabilistic circuit $C' : \mathbb{F}_2^t \times \Omega' \longrightarrow \mathbb{F}_2$ that decides quadratic residuosity in $\mathbb{M}_n^+$ with error probability δ and has size*

$$\#C' \leq \left[3+\frac{1}{2\delta\varepsilon^2}\right]\cdot[\#C + r\tau_t + 4] + 2\cdot\left[3+\frac{1}{2\delta\varepsilon^2}\right]^2.$$

Note that the size of C' is polynomial in r, $\#C$, $\frac{1}{\delta}$, $\frac{1}{\varepsilon}$, and t, and we even could make this polynomial explicit. Thus:

From an efficient probabilistic BBS extrapolation algorithm for the module n with ε-advantage we can construct an efficient probabilistic decision algorithm for quadratic residuosity for n with arbitrary small error probability.

This complexity bound becomes even more perspicuous, when we specify dependencies from the input complexity, measured by the bit size t. Thus we choose

- $r \leq f(t)$ with a polynomial $f \in \mathbb{Q}[T]$ (that is we generate only "polynomially many" pseudorandom bits),
- $\frac{1}{\delta} \leq g(t)$ (or $\delta \geq 1/g(t)$) with a polynomial $g \in \mathbb{Q}[T]$ (that is we don't choose δ "too small", not like an ambitious $\delta < 1/2^t$),
- $\frac{1}{\varepsilon} \leq h(t)$ (or $\varepsilon \geq 1/h(t)$) with a polynomial $h \in \mathbb{Q}[T]$ (that is ε is reasonably small, not only like a modest $\varepsilon \approx 1/\log(t)$).

Then

$$\begin{aligned} \#C' &\leq \left[3 + \frac{1}{2}\, g(t)\, h(t)^2\right] \cdot [\#C + f(t)\, \tau_t + 4] + 2 \cdot \left[3 + \frac{1}{2}\, g(t)\, h(t)^2\right]^2 \\ &\leq \Phi(t) \cdot \#C + \Psi(t) \end{aligned}$$

with polynomials Φ, $\Psi \in \mathbb{Q}[t]$. In the following section we'll see how this statement makes BBS a "perfect" pseudorandom generator.

The hypothetical decision algorithm for $s \in \mathbb{M}_n^+$ from Proposition 13 runs like this (assuming that n is a public parameter):

1. Construct the BBS-sequence $b_1(s), \ldots, b_r(s)$ (using the public parameter n).
2. Choose the desired error probability δ.
3. Choose $m = 2h + 1$ with h as in Equation 1.
4. Choose random elements $\omega_1, \ldots, \omega_m \in \Omega$ and determine $b_i = C(s, \omega_i) \in \mathbb{F}_2$ for $i = 1, \ldots, r$.
5. Count $z = \#\{i \mid b_i = \mathrm{lsb}(s)\}$.
6. If $z \geq m/2$ output 1 ("quadratic residue"), else output 0 ("quadratic nonresidue").

4.3 Perfect Pseudorandom Generators

A sound definition of the concept "pseudorandom generator" is overdue. Informally we define it as an efficient algorithm that takes a "short" bitstring $s \in \mathbb{F}_2^n$ and converts it into a "long" bitstring $s \in \mathbb{F}_2^r$, compare Appendix A.2.

The terminology of complexity theory as in Appendix B of Part III allows us to give a mathematically exact (but not completey satisfying from a practical point of view) definition by considering parameter-dependent families of Boolean maps (or circuits) $G_n \colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^{r(n)}$, and analyzing their behaviour when the parameter n grows to infinity. Such an algorithm—represented by the family $(G_n)_{n\in\mathbb{N}}$ of Boolean circuits—can be efficient only if the "expanding function" $r \colon \mathbb{N} \longrightarrow \mathbb{N}$ grows at most polynomially with the parameter n, otherwise even writing down the output sequence in an efficient way is impossible. We measure the complexity in a meaningful way by the size of the circuit (or by counting the number of needed bit operations) that likewise must grow at most polynomially with n.

To make this idea more precise we consider an infinite parameter set $M \subseteq \mathbb{N}$. We assume that an instance of the generator is defined for each $m \in M$. As an example think of M as a set of Blum integers. Let $M_n = M \cap [2^{n-1} \ldots 2^n[$ be the set of n-bit integers in M.

A **pseudorandom generator with parameter set M and expansion function** r is a family $G = (G_m)_{m\in M}$ of Boolean circuits

$$G_m \colon A_m \longrightarrow \mathbb{F}_2^{r(n)} \quad \text{with } A_m \subseteq \mathbb{F}_2^n,$$

where n is the bitlength of m, such that there exists a (deterministic) polynomial family of circuits $\tilde{G} = (\tilde{G}_n)_{n\in\mathbb{N}}$, where $\tilde{G}_n$ has $2n$ deterministic input nodes, with $\tilde{G}_n(m, x) = G_m(x)$. (In other words: The pseudorandom bits are efficiently computable. In particular the function r is bounded by a polynomial in n.) A_m is called the set of seeds for the parameter m. Thus each G_m expands an n-bit sequence $x \in A_m$ to a $r(n)$-bit sequence $G_m(x) \in \mathbb{F}_2^{r(n)}$.

To see how the BBS generator fits into this definition let M be the set of Blum integers or an infinite subset of it, $A_m = \mathbb{M}_m$, and $G_m(x) = (b_1(x), \ldots, b_{r(n)}(x))$ be the corresponding BBS sequence, $b_i(x) = \mathrm{lsb}(x_i)$ where $x_0 = x$, $x_i = x_{i-1}^2 \bmod m$, for $m \in M$.

A **polynomial test** for the pseudorandom generator G is a polynomial family of (probabilistic) circuits $C = (C_n)_{n\in\mathbb{N}}$,

$$C_n : \mathbb{F}_2^n \times \mathbb{F}_2^{r(n)} \times \Omega_n \longrightarrow \mathbb{F}_2$$

over a probability space $\Omega_n \subseteq \mathbb{F}_2^{s(n)}$ where $s(n)$ is the number of probabilistic inputs of C_n. Thus the test C_n may depend on the parameter m. The probability that the test computes the value 1 for a sequence generated by G is

$$p(G, C, m) = P\{(x, \omega) \in A_m \times \Omega_n \mid C_n(m, G_m(x), \omega) = 1\}.$$

The probability that the test computes the value 1 for an arbitrary ("true random") sequence of the same length is

$$\bar{p}(C, m) = P\{(u, \omega) \in \mathbb{F}_2^{r(n)} \times \Omega_n \mid C_n(m, u, \omega) = 1\}.$$

Ideally (for a "good" generator) these two probabilities should agree approximately: the test should not be an ε-distinguisher for reasonable values of ε and for almost all parameters m. We say the pseudorandom generator G **passes the test** C if for all non-constant polynomials $h \in \mathbb{N}[X]$ the set A of $m \in M$ with

$$|p(G, C, m) - \bar{p}(C, m)| \geq \frac{1}{h(n)}$$

is sparse in M (the set of parameters m for which C is a $1/h(n)$-distinguisher).

Recall from Appendix B.7 of Part III that this means that

$$\frac{\#(A \cap M_n)}{\#M_n} \leq \frac{1}{\eta(n)} \quad \text{for almost all } n \in \mathbb{N}$$

for each non-constant polynomial $\eta \in \mathbb{N}[X]$.

The pseudorandom generator G is called **perfect** if it passes all polynomial tests. In sloppy words:

No efficient statistical test (or algorithm) is able to distinguish a bit sequence generated by G from a "true random" bit sequence.

4.4 Yao's Criterion

At first sight trying to prove the perfectness of a pseudorandom generator G seems hopeless. How to manage "all polynomial tests"? But surprisingly a seemingly much weaker test is sufficient. Let $G_m(x) = (b_1^{(m)}(x), \ldots, b_{r(n)}^{(m)}(x))$ be the bit sequence generated by G_m from the seed x. Let $C = (C_n)_{n\in\mathbb{N}}$ be a polynomial family of circuits,

$$C_n : \mathbb{F}_2^n \times \mathbb{F}_2^{i_n} \times \Omega_n \longrightarrow \mathbb{F}_2$$

with $0 \leq i_n \leq r(n) - 1$, and let $h \in \mathbb{N}[X]$ be a non-constant polynomial. Then we say that C has a $\frac{1}{h}$-advantage for extrapolating G if the set of parameters $m \in M$ with

$$P\{(x,\omega) \in A_m \times \Omega_n \mid C_n(m, b_{j_m+1}^{(m)}(x), \ldots, b_{j_m+i_n}^{(m)}(x), \omega) = b_{j_m}^{(m)}(x)\}$$

$$\geq \frac{1}{2} + \frac{1}{h(n)} \tag{2}$$

for an index j_m, $1 \leq j_m \leq r(n) - i_n$, is not sparse in M. In other words given a subsequence C extrapolates the preceding bit with a small advantage in sufficiently many cases. We say that G passes the **extrapolation test** if there exists no such polynomial family of circuits with a $\frac{1}{h}$-advantage for extrapolating G for any polynomial $h \in \mathbb{N}[X]$.

For instance the linear congruential generator fails the extrapolation test, as does a linear feedback shift register.

Theorem 4 [Yao's criterion] *The following statements are equivalent for a pseudorandom generator G:*

(i) *G is perfect.*

(ii) *G passes the extrapolation test.*

Proof. "(i) $\Longrightarrow$ (ii)": Assume G fails the extrapolation test. Then there is a polynomial family C of circuits that has a $\frac{1}{h}$-advantage for extrapolating G. Let $A \subseteq M$ be the non-sparse set of parameters for which the inequality (2) holds. We construct a polynomial test $C' = (C'_n)_{n\in\mathbb{N}}$:

$$C'_n(m, u, \omega) = C_n(m, u_{j_m+1}, \ldots, u_{j_m+i_n}, \omega) + u_{j_m} + 1$$

where for $m \in \mathbb{F}_2^n - A$ we set $j_m = 1$ (this value doesn't matter anyway). Hence

$$C'_n(m, u, \omega) = 1 \Longleftrightarrow C_n(m, u_{j_m+1}, \ldots, u_{j_m+i_n}, \omega) = u_{j_m}.$$

For $m \in A$ we get

$$p(G, C', m) = P\{C_n(m, b^{(m)}_{j_m+1}(x), \ldots, b^{(m)}_{j_m+i_n}(x), \omega) = b^{(m)}_{j_m}(x)\} \;\geq\; \frac{1}{2} + \frac{1}{h(n)}$$

and have to compare this value with

$$\bar{p}(C', m) = P\{C_n(m, u_{j_m+1}, \ldots, u_{j_m+i_n}, \omega) = u_{j_m}\}$$

$$= P\{C_n(\ldots) = 0 \text{ and } u_{j_m} = 0\} + P\{C_n(\ldots) = 1 \text{ and } u_{j_m} = 1\}.$$

(The sum corresponds to a decomposition into two disjoint subsets.) Each summand denotes the probability that two independent events occur simultaneously. Thus

$$\bar{p}(C', m) = \frac{1}{2}P\{C_n(\ldots) = 0\} + \frac{1}{2}P\{C_n(\ldots) = 1\} = \frac{1}{2}.$$

Hence for $m \in A$

$$p(G, C', m) - \bar{p}(C', m) \geq \frac{1}{h(n)}.$$

We conclude that G fails the test C', and therefore is not perfect.

"(ii) $\Longrightarrow$ (i)": Assume G is not perfect. Then there is a polynomial test C failed by G. Hence there is a non-constant polynomial $h \in \mathbb{N}[X]$ and a $t \in \mathbb{N}$ with

$$|p(G, C, m) - \bar{p}(C, m)| \geq \frac{1}{h(n)}$$

for m from a non-sparse subset $A \subseteq M$ with $\#A_n \geq \#M_n/n^t$ for infinitely many $n \in I$. For at least half of all $m \in A_n$ we have $p(G, C, m) > \bar{p}(C, m)$ or the inverse inequality. First we treat the first of these two cases (for fixed n).

For $k = 0, \ldots, r(n)$ let

$$p^k_m = P\{C_n(m, t_1, \ldots, t_k, b^{(m)}_{k+1}(x), \ldots, b^{(m)}_{r(n)}(x), \omega) = 1\}$$

where $t_1, \ldots, t_k \in \mathbb{F}_2$ are random bits. So we consider the probability in $A_m \times (\mathbb{F}_2^k \times \Omega_n)$. We have

$$p^0_m = p(G, C, m), \quad p^{r(n)}_m = \bar{p}(C, m),$$

$$\frac{1}{h(n)} \leq p^0_m - p^{r(n)}_m = \sum_{k=1}^{r(n)} (p^{k-1}_m - p^k_m)$$

for the $m \in A_n$ under consideration. Thus there is an r_m with $1 \leq r_m \leq r(n)$ such that

$$p^{r_m-1}_m - p^{r_m}_m \geq \frac{1}{r(n)h(n)}.$$

One of these values r_m occurs at least $(\#M_n/2n^t r(n))$ times, denote it by k_n.

Let $\Omega'_n = \mathbb{F}_2^{k_n} \times \Omega_n$. The polynomial family C' of circuits whose deterministic inputs are fed from $A_n \times \mathbb{F}_2^{r(n)-k_n}$, and whose probabilistic inputs from Ω'_n, is defined for this n by

$$C'_n(m, u_1, \ldots, u_{r(n)-k_n}, t_1, \ldots, t_{k_n}, \omega) = C_n(m, t, u, \omega) + t_{k_n} + 1.$$

Hence

$$C'_n(m, u, t, \omega) = t_{k_n} \iff C_n(m, t, u, \omega) = 1.$$

Now

$$C'_n(m, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{r(n)}(x), t, \omega) = b^{(m)}_{k_n}(x)$$

$$\iff \begin{cases} C_n(m, t, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{r(n)}(x), \omega) = 1 \quad \text{and} \quad t_{k_n} = b^{(m)}_{k_n}(x) \\ \text{or} \\ C_n(m, t, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{r(n)}(x), \omega) = 0 \quad \text{and} \quad t_{k_n} \neq b^{(m)}_{k_n}(x) \end{cases}$$

Both cases describe the occurence of two independent events. Therefore the probability of the second one is $\frac{1}{2}(1 - p_m^{k_n})$. The first one is equivalent with

$$C_n(m, t_1, \ldots, t_{k_n-1}, b^{(m)}_{k_n}(x), \ldots, b^{(m)}_{r(n)}(x), \omega) = 1 \quad \text{and} \quad t_{k_n} = b^{(m)}_{k_n}(x)\,.$$

Its probability is $p_m^{k_n-1}/2$. Together this gives

$$P\{C'_n(m, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{r(n)}(x), t, \omega) = b^{(m)}_{k_n}(x)\}$$

$$= \frac{1}{2} + \frac{1}{2}(p_m^{k_n-1} - p_m^{k_n}) \geq \frac{1}{2} + \frac{1}{2r(n)h(n)}$$

for at least $\#M_n/2n^t r(n)$ of the parameters $m \in M_n$. With $u = t + \deg(r) + 1$ this is $\geq \#M_n/n^u$ for infinitely many $n \in I$.

In the case where $p(G, C, m) < \bar{p}(C, m)$ for at least half of all $m \in A_n$ we analoguously set

$$C'_n(m, u, t, \omega) = C_n(m, t, u, \omega) + t_{k_n}.$$

Then the derivation runs along the same lines.

Therefore G fails the extrapolation test (with $i_n = r(n) - k_n$ and $j_m = k_n$). $\diamond$

By the way the proof made use of the non-uniformity of the computational model: C'_n depends on k_n, and we didn't give an algorithm that determines k_n.

4.5 The Prediction Test

The extrapolation test looks somewhat strange since it extrapolates the bit sequence in reverse direction, a clear contrast with the usual cryptanalytic procedures that try to predict *forthcoming* bits. We'll immediately remedy this quaint effect:

Let $C = (C_n)_{n\in\mathbb{N}}$ be a polynomial family of circuits,

$$C_n : \mathbb{F}_2^n \times \mathbb{F}_2^{i_n} \times \Omega_n \longrightarrow \mathbb{F}_2$$

with $0 \leq i_n \leq r(n)-1$, and let $h \in \mathbb{N}[X]$ be a non-constant polynomial. Then C has a $\frac{1}{h}$-advantage for predicting G if the subset of parameters $m \in M$ with

$$P\{(x,\omega) \mid C_n(m, b_1^{(m)}(x), \ldots, b_{i_n}^{(m)}(x), \omega) = b_{i_n+1}^{(m)}(x)\} \geq \frac{1}{2} + \frac{1}{h(n)}$$

is not sparse in M. The pseudorandom generator G passes the **prediction test** if no polynomial family of circuits has an advantage for predicting G. The proof of "(i) $\Longrightarrow$ (ii)" in Theorem 4 directly adapts to this situation yielding:

Corollary 1 *Every perfect pseudorandom generator passes the prediction test.*

Corollary 2 *If the quadratic residuosity conjecture is true, then the BBS generator is perfect, in particular passes the prediction test.*

Proof. Otherwise from Proposition 13 we could construct a polynomial family of circuits that decides quadratic residuosity for a non-sparse subset of Blum integers. ◇

The paper

> U. V. Vazirani, V. V. Vazirani: Efficient and secure pseudo-random number generation, Crypto 84, 193–202

contains a stronger result: *If the factoring conjecture is true, i. e. if factoring large integers is hard, then the BBS generator is perfect.*

4.6 Examples and Practical Considerations

We saw that the BBS generator is perfect under a plausible but unproven assumption, the quadratic residuosity hypothesis. However we don't know relevant concrete details, for example what parameters might be inappropriate. We know that certain initial states generate output sequences with short periods. Some examples of this effect are known, but we are far from a complete answer except for superspecial Blum modules. However the security proof (depending on the quadratic residuosity hypothesis) doesn't require additional assumptions. Therefore we may confidently use the BBS generator with a pragmatic attitude: randomly choosing the parameters (primes and initial state) the probability of hitting "bad" values is extremely low, much lower then finding a needle in a haystack, or even in the universe.

Nevertheless some questions are crucial for getting good pseudorandom sequences from the BBS generator in an efficient way:

- How large should we choose the module m?
- How many bits can we use for a fixed module and initial state without compromising the security?

The provable results—relative to the quadratic residuosity hypothesis—are qualitative only, not quantitative. The recommendation to choose a module that escapes the known factorization methods also rests on heuristic considerations only, and doesn't seem absolutely mandatory for a module that itself is kept secret. The real quality of the pseudorandom bit sequence, be it for statistical or for cryptographic applications, can only be assessed by empirical criteria for the time being. We are confident that the danger of generating a "bad" pseudorandom sequence is extremely small, in any case negligeable, for modules that escape the presently known factorization algorithms, say at least of a length of 2048 bits, and for a true random choice of the module and the initial state.

> Émile Borel proposed an informal ranking of negligeability of extremely small probabilities: $\leq 10^{-6}$ from a human view; $\leq 10^{-15}$ from a terrestrial view; $\leq 10^{-45}$ from a cosmic view. By choosing a sufficiently large module m for RSA or BBS we easily undercut Borel's bounds by far.

For the length of the useable output sequence we only know the qualitative criterion "at most polynomially many" that is useless in a concrete application. But even if we only use "quadratically many" bits we wouldn't hesitate to take 4 millions bits from the generator with a ≥ 2000 bit module. Should we need substantially more bits we would restart the generator with new parameters after every few millions of bits.

An additional question suggests itself: Are we allowed to output more then a single bit of the inner state in each iteration step to enhance the practical benefit of the generator? At least 2 bits?

Vazirani and Vazirani, and independently Alexi, Chor, Goldreich, and Schnorr gave a partial answer to this question, unfortunately also a qualitative one only: at least $O(\log_2 \log_2 m)$ of the least significant bits are "safe". Depending on the constants that hide in the "O" we need to choose a sufficiently large module, and trust empirical experience. A common recommendation is using $\lfloor \log_2 \log_2 m \rfloor$ bits per step. Then for a module m of 2048 bits, or roughly 600 decimal places, we can use 11 bits per step. Calculating $x^2 \bmod m$ for a n bit number m takes $(\frac{n}{64})^2$ multiplications of 64-bit integers and subsequently the same number of divisions of the type "128 bits by 64 bits". For $n = 2048$ this makes a total of $2 \cdot (2^5)^2 = 2048$ multiplicative operations to generate 11 bits, or about 200 operations per bit. A well-established rule of thumb says that a modern CPU executes one multiplicative operation per clock cycle. (Special CPUs that use pipelines and parallelism are significantly faster.) Thus on a 2-GHz CPU with 64-bit architecture we may expect roughly $2 \cdot 10^9/200 \approx 10$ million bits per second, provided the algorithm is implemented in an optimized way. This consideration shows that the BBS generator is almost competitive with a software implementation of a sufficiently secure nonlinear combiner of LFSRs, and is fast enough for many purposes if executed on a present day CPU.

The cryptographic literature offers several pseudorandom generators that follow similar principles as BBS:

The RSA generator (Shamir). Choose a random module m of n bits as a product of two large primes p, q, and an exponent d that is coprime with $(p-1)(q-1)$, furthermore a random initial state $x = x_0$. The state transition is $x \mapsto x^d \bmod m$. Thus we calculate $x_i = x_{i-1}^d \bmod m$, and output the least significant bit, or the $\lfloor \log_2 \log_2 m \rfloor$ least significant bits. If the RSA generator is not perfect, then there exists an efficient algorithm that breaks the RSA cipher. Since calculating d-th powers is more expensive by a factor n than squaring the cost is higher then for BBS: for a random d the algorithm needs $O(n^3)$ cycles per bit.

The index generator (Blum/Micali). As module choose a random large prime p of n bits, and find a primitive root a for p. Furthermore choose a random initial state $x = x_0$, coprime with $p - 1$. Then calculate $x_i = a^{x_{i-1}} \bmod p$, and and ouput the most significant bit of x_i, or the $\lfloor \log_2 \log_2 p \rfloor$ most significant bits. The perfectness of the index generator relies on the hypothesis that calculating discrete logarithms mod$p$ is hard. The cost per bit also is $O(n^3)$.

The elliptic index generator (Kaliski). It works like the index generator, but replacing the group of invertible elements of the field $\mathbb{F}_p$ by

an elliptic curve over $\mathbb{F}_p$ (such a curve is a finite group in a canonical way).

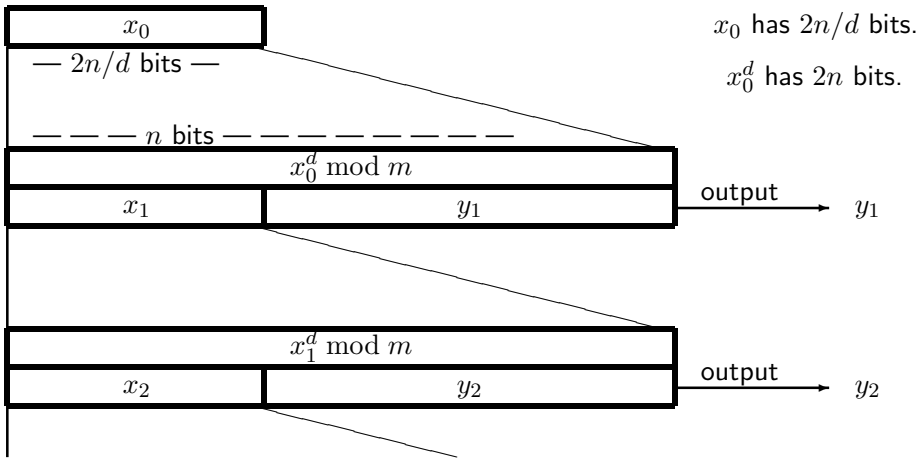

Figure 4.4: Micali-Schnorr generator

4.7 The Micali-Schnorr Generator

Micali and Schnorr proposed a pseudorandom generator that is a descendent of the RSA generator. Fix an odd number $d \geq 3$. The parameter set is the set of all products m of two primes p and q whose bit lengths differ by at most 1, and such that d is coprime with $(p-1)(q-1)$. For an n-bit number m let $h(n)$ be an integer $\approx \frac{2n}{d}$. Then the d-th power of an $h(n)$-bit number is (approximately) a $2n$-bit number.

In the i-th step calculate $z_i = x_{i-1}^d \bmod m$. Take the first $h(n)$ bits as the new state x_i, that is $x_i = \lfloor z_i / 2^{n-h(n)} \rfloor$, and output the remaining bits, that is $y_i = z_i \bmod 2^{n-h(n)}$. Thus the bits of the result z_i are partitioned into two *disjoint* parts: the new state x_i, and the output y_i. Figure 4.4 illustrates this scheme.

But why may we hope that this pseudorandom generator is perfect? This depends on the hypothesis: There is no efficient test that distinguishes the uniform distribution on $\{1, \ldots, m-1\}$ from the distribution of $x^d \bmod m$ for uniformly distributed $x \in \{1, \ldots, 2^{h(n)}\}$. If this hypothesis is true, then the Micali-Schnorr generator is perfect. This argument seems tautologic, but heuristic considerations show a relation with the security of RSA and with factorization. Anyway we have to concede that this "proof of security" seems considerably more airy then that for BBS.

How fast do the pseudorandom bits tumble out of the machine? As elementary operations we again count the multiplication of two 64-bit numbers, and the division of a 128-bit number by a 64-bit number with 64-bit quotient. We multiply and divide by the classical algorithms. Thus the product of s (64-bit) words and t words costs st elementary operations. The cost of

division is the same as the cost of the product of divisor and quotient.

The concrete recommendation by the inventors is: $d = 7$, $n = 512$. (Today we would choose a larger n.) The output of each step consists of 384 bits, withholding 128 bits as the new state. The binary power algorithm for a 128-bit number x with exponent 7 costs several elementary operations:

- x has 128 bits, hence 2 words.
- x^2 has 256 bits, hence 4 words, and costs $2 \cdot 2 = 4$ elementary operations.
- x^3 has 384 bits, hence 6 words, and costs $2 \cdot 4 = 8$ elementary operations.
- x^4 has 512 bits, hence 8 words, and costs $4 \cdot 4 = 16$ elementary operations.
- x^7 has 896 bits, hence 14 words, and costs $6 \cdot 8 = 48$ elementary operations.
- $x^7 \bmod m$ has ≤ 512 bits, and likewise costs $6 \cdot 8 = 48$ elementary operations.

This makes a total of 124 elementary operations; among them only one reduction $\bmod m$ (for x^7). Our reward consists of 384 pseudorandom bits. Thus we get about 3 bits per elementary operation, or, by the assumptions in Section 4.6, about 6 milliards bits per second. Compared with the BBS generator this amounts to a factor of about 1000.

Parallelization increases the speed virtually without limit: The Micali-Schnorr generator allows complete parallelization. Thus distributing the work among k CPUs brings a profit by the factor k since the CPUs can work indepedently of each other without need of communication.

4.8 The Impagliazzo-Naor Generator

Recall the knapsack problem (or subset sum problem):

Given positive integers $a_1, \ldots, a_n \in \mathbb{N}$ and $T \in \mathbb{N}$.

Wanted a subset $S \subseteq \{1, \ldots, n\}$ with

$$\sum_{i \in S} a_i = T.$$

This problem is believed to be hard. We know it is NP-complete. Building on it Impagliazzo and Naor developed a pseudorandom generator:

Let k and n be (sufficiently large) integers with $n < k < \frac{3n}{2}$. As parameters we choose random $a_1, \ldots, a_n \in [1 \ldots 2^k]$.

Attention: quite a lot of big numbers.

The state space consists of the power set of $\{1, \ldots, n\}$. So the states are subsets $S \subseteq \{1, \ldots, n\}$. We represent them by bit sequences in $\mathbb{F}_2^n$ in the natural way. In each single step we form the sum

$$\sum_{i \in S} a_i \mod 2^k.$$

This is a k-bit integer. Output the first $k - n$ bits, and retain the last n bits as the new state, see Figure 4.5

Thus state transition and output function are:

$$\begin{aligned} T(S) &= \sum_{i \in S} a_i \bmod 2^n \\ &\quad \text{(retain the rightmost } n \text{ bits)} \\ U(S) &= \left\lfloor \frac{\sum_{i \in S} a_i \bmod 2^k}{2^n} \right\rfloor \\ &\quad \text{output the leftmost } k - n \text{ bits} \end{aligned}$$

If this pseudorandom generator is not perfect, then the knapsack problem admits an efficient solution. Here we omit the proof. See

- R. Impagliazzo, M. Naor: Efficient cryptographic schemes provably as secure as subset sum. J. Cryptology 9 (1996), 199–216.

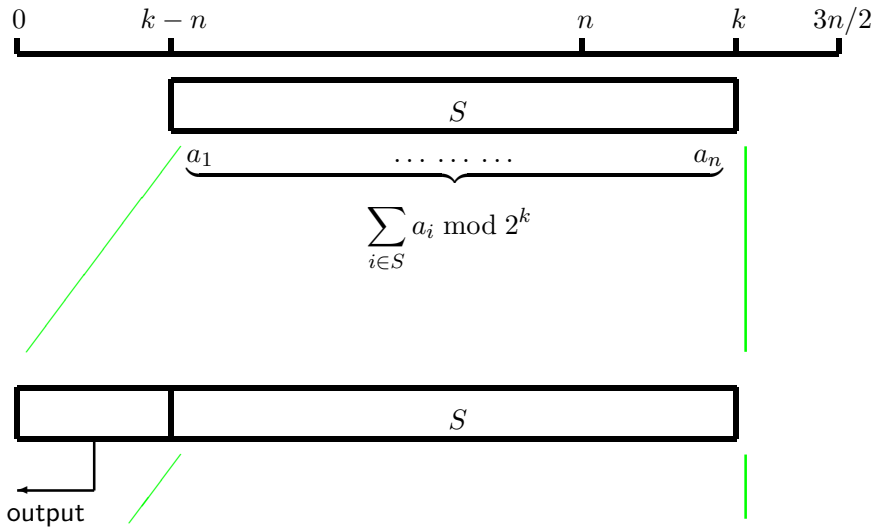

Figure 4.5: The Impagliazzo-Naor generator

Appendix A

Statistical Distiguishers

As usual in these lecture notes we restrict ourselves to finite probability spaces.

A.1 Distingishing Distributions by a Test

Let A be a finite probability space with two probability distributions P_0 and P_1. Accordingly for a real valued function $\Delta\colon A \longrightarrow \mathbb{R}$ we have the mean values (or expectations)

$$\mu_i = \sum_{a \in A} \Delta(a) \cdot P_i(a).$$

For $\varepsilon > 0$ we call Δ an **ε-distinguisher** of P_0 and P_1 if

$$|\mu_1 - \mu_0| \geq \varepsilon.$$

That is, the expectations of Δ with respect to P_0 and P_1 differ considerably.

> Note the analogy with the common statistical test scenario where we decide whether a sample deviates from an assumed distribution by comparing mean values.

This notion has an obvious analogue for bit valued functions (or binary attributes) $\Delta\colon A \longrightarrow \mathbb{F}_2$. Here

$$\mu_i \;=\; \sum_{a \in \Delta^{-1}(1)} P_i(a) \;=\; P_i(\Delta^{-1}(1))$$

is the probability that $\Delta(a) = 1$ for a randomly chosen $a \in A$. Thus

$$\mu_1 - \mu_0 = P_1(\Delta^{-1}(1)) - P_0(\Delta^{-1}(1)).$$

The "test" Δ ε-distinguishes between the distributions P_1 and P_0 if the probabilities for $\Delta(a) = 1$ with respect to these two distributions differ by at least ε.

> Note that the notion "test" just means "function". However in the present context it suggests a role that this function plays. A similar remark also holds for the notion "randomize".

We may "randomize" our test by more generally considering a function

$$\Delta\colon A \times \Omega \longrightarrow \mathbb{F}_2$$

where Ω is a finite probability space from which we take an additional random input ω, and then consider the probabilities μ_i that $\Delta(a, \omega) = 1$,

$$\mu_i \;=\; \frac{1}{\#A \cdot \#\Omega} \cdot \#\{(a, \omega) \in A \times \Omega \mid \Delta(a, \omega) = 1\}.$$

A.2 Testing Bitsequences

A statistical test for bitsequences of length r is simply a Boolean function $\Delta\colon \mathbb{F}_2^r \longrightarrow \mathbb{F}_2$, a probabilistic statistical test is a function

$$\Delta\colon \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$$

where Ω is a finite probability space.

We want to distinguish between random bitsequences $u \in \mathbb{F}_2^r$, and bitsequences that arise from a "generator map"

$$G\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^r$$

that transforms a randomly chosen $x \in \mathbb{F}_2^n$ (called "seed") to a bitsequence $G(x) \in \mathbb{F}_2^r$. This sequence $G(x)$, if it passes our tests, may qualify as a pseudorandom sequence. In this test scenario the reference distribution P_0 is the uniform distribution on $\mathbb{F}_2^r$,

$$P_0(u) = \frac{1}{2^r} \quad \text{for all } u \in \mathbb{F}_2^r\,.$$

We want to compare it with the induced distribution

$$P_1(u) = \frac{1}{2^n} \cdot \#\{x \in \mathbb{F}_2^n \mid G(x) = u\}\,.$$

Or, somewhat more generally, if G is defined on a subset $A \subseteq \mathbb{F}_2^n$ only,

$$P_1(u) = \frac{1}{\#A} \cdot \#\{x \in A \mid G(x) = u\}\,.$$

A probabilistic statistical test $\Delta : \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$ ε-distinguishes between random bitsequences $u \in \mathbb{F}_2^r$ and sequences generated by $G\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^r$ if

$$|\mu_1 - \mu_0| \geq \varepsilon$$

where

$$\mu_0 = \frac{1}{2^r \cdot \#\Omega} \cdot \#\{(u,\omega) \in \mathbb{F}_2^r \times \Omega \mid \Delta(u,\omega) = 1\}$$

is the probability that the test assigns the value 1 to a random bitsequence $u \in \mathbb{F}_2^r$, and

$$\mu_1 = \frac{1}{2^n \cdot \#\Omega} \cdot \#\{(x,\omega) \in A \times \Omega \mid \Delta(G(x),\omega) = 1\}$$

is the probability that the test yields the value 1 for a bitstring generated by a random seed $x \in A$.

Examples

We want to distinguish sequences generated by a map $G\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^r$ from random sequences (by deterministic tests, that is $\#\Omega = 1$).

Example 1

First an extremely simple example with the test function

$$\Delta\colon \mathbb{F}_2^r \longrightarrow \mathbb{F}_2, \quad \Delta(u) = \begin{cases} 1 & \text{if } \#\{i \mid u_i = 1\} \geq \frac{r}{2}, \\ 0 & \text{otherwise,} \end{cases}$$

That is Δ decides on the majority of ones in the sequence u. Then obviously $\mu_0 = \frac{1}{2}$.

Case 1a: Let $n = 1$ and $G\colon \mathbb{F}_2 \longrightarrow \mathbb{F}_2^r$ be defined by

$$\begin{aligned} G(0) &= (0,0,0,\ldots), \\ G(1) &= (1,1,1,\ldots). \end{aligned}$$

Then also $\mu_1 = \frac{1}{2}$, yielding $\mu_1 - \mu_0 = 0$. Thus Δ is not an ε-distinguisher for any $\varepsilon > 0$.

Case 1b: We keep the definition of $G(1)$ but change the definition of $G(0)$ to

$$G(0) = (1,0,1,0,1,\ldots).$$

Then $\Delta(G(0)) = \Delta(G(1)) = 1$, hence $\mu_1 = 1$, yielding $\mu_1 - \mu_0 = \frac{1}{2}$. Thus Δ is an ε-distinguisher for $0 < \varepsilon \leq \frac{1}{2}$.

Example 2

For a serious example we consider sequences generated by a linear feedback shift register $G\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^r$ of length n where $2n < r \leq 2^n - 1$. We know that the output of G is distinguished by a low linear complexity $\lambda(u) \leq n$. Therefore we use

$$\Delta\colon \mathbb{F}_2^r \longrightarrow \mathbb{F}_2, \quad \Delta(u) = \begin{cases} 1 & \text{if } \lambda(u) < \frac{r}{2}, \\ 0 & \text{if } \lambda(u) \geq \frac{r}{2}, \end{cases}$$

as test . Since $n < \frac{r}{2}$ this yields

$$\mu_1 \;=\; \frac{1}{2^n} \cdot \#\{x \in \mathbb{F}_2^n \mid \Delta(G(x)) = 1\} \;=\; 1.$$

For arbitrary sequences $u \in \mathbb{F}_2^r$ we know from Theorem 3 that we may expect $\lambda(u) \approx \frac{r}{2}$. A more precise statement follows from the frequency count in Proposition 11

$$k \;:=\; \#\{u \in \mathbb{F}_2^r \mid \lambda(u)) \leq \frac{r-1}{2}\} \;=\; 1 + \sum_{l=1}^{\lfloor \frac{r-1}{2} \rfloor} 2^{2l-1} \;=\; \frac{1}{2} + \frac{1}{2} \cdot \sum_{l=0}^{\lfloor \frac{r-1}{2} \rfloor} 4^l.$$

Case 2a: Let r be even. Then $\lfloor \frac{r-1}{2} \rfloor = \frac{r}{2} - 1$, and

$$k \;=\; \frac{1}{2} + \frac{1}{2} \cdot \frac{4^{r/2} - 1}{3} \;=\; \frac{1}{2} + \frac{1}{6} \cdot (2^r - 1) \;=\; \frac{1}{3} + \frac{1}{6} \cdot 2^r,$$

$$\mu_0 \;=\; \frac{1}{2^r} \cdot k \;=\; \frac{1}{6} + \frac{1}{3 \cdot 2^r} \;\leq\; \frac{1}{3} \quad \text{for } r \geq 1.$$

Case 2b: Let r be odd. Then $\lfloor \frac{r-1}{2} \rfloor = \frac{r-1}{2}$, and

$$k \;=\; \frac{1}{2} + \frac{1}{2} \cdot \frac{4^{(r+1)/2} - 1}{3} \;=\; \frac{1}{2} + \frac{1}{6} \cdot (2^{r+1} - 1) \;=\; \frac{1}{3} + \frac{1}{3} \cdot 2^r,$$

$$\mu_0 \;=\; \frac{1}{2^r} \cdot k \;=\; \frac{1}{3} + \frac{1}{3 \cdot 2^r} \;\leq\; \frac{1}{2} \quad \text{for } r \geq 1.$$

Hence in any case we have

$$\mu_1 - \mu_0 \;\geq\; \frac{1}{2} \quad \text{for } r \geq 1.$$

Thus Δ is an ε-distinguisher for $0 < \varepsilon \leq \frac{1}{2}$, distinguishing between LFSR sequences and random sequences.