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 $\mathrm{O}(\log_2 \log_2 m)$ 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 $\mathrm{O}(n^3)$ 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 $\mathrm{O}(n^3)$.

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).