Concrete Security of the Blum-Blum-Shub Pseudorandom Generator

Andrey Sidorenko and Berry Schoenmakers

Eindhoven University of Technology,
P.O. Box 513, 5600MB Eindhoven, The Netherlands
{a.sidorenko, l.a.m.schoenmakers}@tue.nl

Abstract. The asymptotic security of the Blum-Blum-Shub (BBS) pseudorandom generator has been studied by Alexi et al. and Vazirani and Vazirani, who proved independently that $O(\log \log N)$ bits can be extracted on each iteration, where N is the modulus (a Blum integer). The concrete security of this generator has been analyzed previously by Fischlin and Schnorr and by Knuth.

In this paper we continue to analyse the concrete security the BBS generator. We show how to select both the size of the modulus and the number of bits extracted on each iteration such that a desired level of security is reached, while minimizing the computational effort per output bit. We will assume a concrete lower bound on the hardness of integer factoring, which is obtained by extrapolating the best factorization results to date.

While for asymptotic security it suffices to give a polynomial time reduction a successful attack to factoring, we need for concrete security a reduction that is as efficient as possible. Our reduction algorithm relies on the techniques of Fischlin and Schnorr, as well as ideas of Vazirani and Vazirani, but combining these in a novel way for the case that more than one bit is output on each iteration.

1 Introduction

Generally speaking, a pseudorandom generator is a deterministic algorithm that, given a truly random binary sequence of length n, outputs a binary sequence of length $M > n$ that "looks random". The input to the generator is called the seed and the output is called the pseudorandom bit sequence. Security of a pseudorandom generator is a characteristic that shows how hard it is to tell the difference between the pseudorandom sequences and truly random sequences. For the Blum-Blum-Shub (BBS) pseudorandom generator [2] distinguishing these two distributions is as hard as factoring a large composite integer.

Although asymptotic security of the BBS generator is thoroughly analyzed [1,15] it has been uncertain how to select the size of the modulus and the number of bits extracted on each iteration such that a desired level of security is reached, while minimizing the computational effort per output bit. In this paper we answer this question. We construct an efficient reduction of a successful attack on the BBS generator to factoring. Then we assume a concrete lower bound

on the hardness of integer factoring, which is obtained by extrapolating the best factorization results to date. It gives a lower bound for the running time of the successful attack on the BBS generator (Theorem 3). This lower bound is used for selecting the optimal values for the size of the modulus and the number of bits extracted on each iteration.

1.1   Notation

Throughout we use the following notation.

- $s \in_R S$ indicates that s is chosen uniformly at random from set S.
- N is a Blum integer, that is $N = pq$, where p, q are prime, $p \equiv q \equiv 3 \bmod 4$.
- n is the size (in bits) of N.
- $\mathbb{Z}_N(+1)$ is the set of integers of Jacobi symbol $+1$ modulo N.
- $\Lambda_N = \mathbb{Z}_N(+1) \cap (0, \frac{N}{2})$.
- $[y]_N = y \bmod N \in [0, N)$ for $y \in \mathbb{Z}$.
- $y \underline{\bmod} N \in \left(-\frac{N}{2}, \frac{N}{2}\right)$ denotes the smallest absolute residue of y modulo N.
- $\ell_i(y)$ denotes the i-th least significant bit of y, $i = 1, 2, \ldots$.
- $E_N(y) = |y^2 \underline{\bmod} N|$, which is referred to as the absolute Rabin function.

Note that the absolute Rabin function E_N permutes Λ_N [5].

1.2   Security of Pseudorandom Generators

Let G be a pseudorandom generator that produces binary sequences of length M. Let $S \subset \{0, 1\}^M$ be the set of these sequences.

Consider a probabilistic algorithm A that, given a binary sequence $s = s_1 \ldots s_M$, outputs a bit $A(s) \in \{0, 1\}$. We may think of A as a statistical test of randomness.

Definition 1. *A pseudorandom generator G passes statistical test A with tolerance $\varepsilon > 0$ if*

$$|\Pr(A(s) = 1 \mid s \in_R S) - \Pr(A(s) = 1 \mid s \in_R \{0, 1\}^M)| < \varepsilon.$$

Otherwise the pseudorandom generator G fails statistical test A with tolerance ε. The probability is taken over all choices of s, and internal coin flips of A.

We refer to a sequence $s \in_R \{0, 1\}^M$ as a truly random binary sequence. Throughout this paper we often call A an adversary that tries to distinguish pseudorandom sequences from truly random sequences.

Note that the maximum possible value of ε is $\varepsilon = 1 - 2^{n-M}$. It corresponds to the statistical test that with probability 1 outputs 1 if $s \in S$ and outputs 0 if $s \in \{0, 1\}^M \backslash S$.

Definition 2. *A pseudorandom generator is asymptotically secure if it passes all polynomial time statistical tests with tolerance negligible in n.*

The above definition originates from [16]. Asymptotic security guarantees that, as the seed length increases, no polynomial time statistical test can distinguish the pseudorandom sequences from truly random sequences with non-negligible probability. However, this definition says little about the security of the pseudorandom generator in practice for a particular choice of seed length and against adversaries investing a specific amount of computational effort. For practical considerations it is important to focus on *concrete security reductions* which give explicit bounds on running time and success probability of statistical tests. The following definition is due to [9,3,14].

Definition 3. *A pseudorandom generator is (T_A, ε)-secure if it passes all statistical tests with running time at most T_A with tolerance ε.*

We determine the values for T_A and ε such that the BBS pseudorandom generator defined below is (T_A, ε)-secure.

1.3   The BBS Generator

The following definition is due to [5].

Definition 4 (The BBS pseudorandom generator). *Let k, j be positive integers. Let $x_1 \in_R \Lambda_N$ be the seed. Consider a deterministic algorithm that transforms the seed into a binary sequence of length $M = jk$ by repeating the following steps for $i = 1, \ldots, k$.*

1. For $r = 1, \ldots, j$ output $b_{(i-1)j+r} = \ell_{j-r+1}(x_i)$.
2. $x_{i+1} = E_N(x_i)$.

We call this algorithm the BBS pseudorandom generator with j output bits per iteration.

In order to output a pseudorandom sequence (a BBS sequence) of length M the generator iterates the absolute Rabin function k times generating j bits on each iteration.

Remark 1. Strictly speaking, the above definition of the BBS differs from the original one presented in [2]. The original generator iterates the Rabin function $E_N^*(x) = x^2 \bmod N$ and outputs only one bit on each iteration ($j = 1$). We do not introduce a new name for the sake of simplicity.

1.4   Known Results and Our Contributions

Intuitively, the performance of the BBS generator can be improved in two ways. We can either use a modulus of a smaller size or extract more bits per iteration. However, in both cases the security of the algorithm is weakened. What are the optimal values for the parameters given that a certain level of security has to be reached? For instance, what is the optimal value for j?

The security of the BBS generator is proved by reduction. It is shown that if the generator is insecure then there exists an algorithm that factors the modulus. When analyzing asymptotic security the only requirement is that the reduction has to be polynomial time. In case of concrete security the reduction has to be as tight as possible. A tight reduction gives rise to a small modulus, which in turn ensures that the pseudorandom generator is efficient.

The following reductions are known for the BBS generator. The case $j = 1$ has been studied extensively. The tightest reduction is due to Fischlin and Schnorr [5], which gives rise to a rather efficient generator. For the case $j > 1$, the asymptotic security has been analyzed fully by Alexi et al. [1], and independently, by Vazirani and Vazirani [15], who proved that the BBS generator is secure if $j = O(\log \log N)$. However, using their reductions as a basis for the concrete security of the BBS generator would imply that in practical case it does not pay off to extract more than 1 bit on each iteration. Fischlin and Schnorr [5] already suggested ways to tighten the reductions for the case $j > 1$. However, as they point out, the first approach is completely impractical. The second one, though, is similar to our analysis, but they provide no sufficient detail.

Inspired by the ideas of [5] and [15] we construct a new security proof for the BBS generator with j output bits per iteration for $j > 1$. The new reduction is more efficient than all previously known reductions for $j > 1$.

We show how to select both the size of the modulus and the number of bits extracted on each iteration such that a desired level of security is reached, while minimizing the computational effort per output bit. Although the complexity of the reduction grows exponentially in j it does not mean that one should always choose $j = 1$. In Example 7.3 the optimal value is $j = 5$ rather than $j = 1$. We emphasize that the optimal parameter j depends on the length of the output sequence M and on the security parameters T_A, ε.

The rest of the paper is organized as follows. In Section 2 we describe a general idea of the security proof for the BBS generator. The result of [9,16] implies that if the generator is insecure then there exists an algorithm B that, given $E_N(x)$ for some $x \in \Lambda_N$, and $j-1$ least significant bits of x, guesses the j-th least significant bit $\ell_j(x)$. In Section 4 the algorithm B is used for inversion of the absolute Rabin function. Before that, in Section 3, we discuss a simplified inversion algorithm as a stepping stone to the general case. The simplified algorithm is of independent interest since it is almost optimal in terms of the running time. In Section 5 we analyze the success probability of the inversion algorithm of Section 4. We determine the complexity of this algorithm in Section 6. In Section 7 we state our main result about the concrete security of the BBS generator.

2   Security of the BBS Generator

In this section we describe a general idea of the security proof for the BBS generator.

Lemma 1. *Suppose the BBS generator is not (T_A, ε)-secure. Then there exists an algorithm B that, given $E_N(x)$ for some $x \in_R \Lambda_N$, $j-1$ least significant bits*

of x, guesses the j-th least significant bit $\ell_j(x)$ with advantage $M^{-1}\varepsilon$. Here the probability is taken over all choices of $x \in_R \Lambda_N$, and internal coin flips of B. The running time $T_B \leq T_A + O(kn^2)$.

The proof of the above lemma can be found, for instance, in [9].

In Section 4 we show that the algorithm B can be used for the inversion of the absolute Rabin function. Before that, in Section 3, we show how to invert the absolute Rabin function using a "simpler" oracle. Section 3 serves as a stepping stone to the general case.

According to the following lemma inversion of the absolute Rabin function is as hard as factoring Blum integers.

Lemma 2 (Rabin). *Suppose there exists a probabilistic algorithm R that recovers $x \in \Lambda_N$ from $E_N(x)$ in expected time T_R. Then there exists an algorithm F that factors the modulus N in expected time $T_F = 2(T_R + 2\log_2 N)$.*

Since factoring Blum integers is assumed to be a hard problem (in Section 7.1 we will assume a concrete lower bound on the hardness of factoring) "attacking" BBS sequences is also a hard problem.

In practice the terms kn^2 and $\log_2 N$ are small in comparison with T_A and T_R respectively. We omit these terms in the further analysis.

3 The Simplified Inversion Algorithm

To complete our concrete security analysis of the BBS generator, we need to show how to invert the absolute Rabin function E_N, given an oracle of a particular type. However, in this section we will consider the related problem of inverting E_N given a more powerful oracle O_1, see below, and assuming that $2 \in \mathbb{Z}_N(+1)$ (which holds if $N \equiv 1 \bmod 8$, see e.g. [12]). The treatment of this case serves as a stepping stone to the general case, and, additionally, we will point out that in this case the reduction can be shown optimal up to a factor of $\ln n$.

The oracle O_1 is defined as a probabilistic algorithm that *for all $x \in \Lambda_N$*, given $E_N(x)$, guesses bit $\ell_1(x)$ with advantage $\delta > 0$, where the probability is taken over internal coin flips of O_1.

3.1 Binary Division

The main tool of the inversion algorithm is the *binary division* technique [5], which is a means to solve the following problem. The problem is to recover a value α, $0 \leq \alpha < N$, given $\ell_1(\alpha)$, $\ell_1([2^{-1}\alpha]_N)$, ..., $\ell_1([2^{-(n-1)}\alpha]_N)$, where n is the bit length of N.

The solution to this problem is given in terms of rational approximations. For a rational number β, $0 \leq \beta < 1$, we call βN a *rational approximation* of integer α, $0 \leq \alpha < N$, with *error* $|\alpha - \beta N|$. Given a rational approximation βN for α we can get a rational approximation $\beta_1 N$ for $\alpha_1 = [2^{-1}\alpha]_N$ for which the error is reduced by a factor of 2 as follows. If α is even, then $\alpha_1 = \alpha/2$

so put $\beta_1 = \beta/2$; otherwise, $\alpha_1 = (\alpha + N)/2$ so put $\beta_1 = (\beta + 1)/2$. Then we have $|\alpha_1 - \beta_1 N| = \frac{1}{2}|\alpha - \beta N|$. Note that to determine β_1, the only required information on α is its parity.

Given $\ell_1(\alpha), \ell_1([2^{-1}\alpha]_N), \ldots, \ell_1([2^{-(n-1)}\alpha]_N)$, the value of α can be recovered as follows. Put $\beta_0 = 1/2$, then $\beta_0 N$ is a rational approximation of α with error at most $N/2$. Next, we apply the above technique n times to obtain rational approximations $\beta_1 N, \ldots, \beta_n N$ for $[2^{-1}\alpha]_N, \ldots, [2^{-n}\alpha]_N$ respectively, at each step reducing the error by a factor of 2. We thus get a rational approximation $\beta_n N$ to $[2^{-n}\alpha]_N$, for which the error is less than $N/2^{n+1} < 1/2$. The closest integer to $\beta_n N$ is therefore equal to $[2^{-n}\alpha]_N$, and from this value we find α.

3.2   Majority Decision

The bits $\ell_1(\alpha)$, $\ell_1([2^{-1}\alpha]_N)$, $\ldots$, $\ell_1([2^{-(n-1)}\alpha]_N)$ used to recover α by means of the binary division technique will be obtained from the oracle O_1, which essentially outputs $\ell_1(\alpha)$ on input $E_N(\alpha)$ for $\alpha \in \Lambda_N$. However, since the output bit of O_1 is not always correct, we have to run O_1 several times and use some form of majority decision.

Suppose we know $E_N(\alpha)$ and our goal is to determine $\ell_1(\alpha)$ for some $\alpha \in \Lambda_N$. We run O_1 on input $E_N(\alpha)$ m times and assign the majority bit to $\ell_1(\alpha)$. We will show that for $m = \frac{1}{2}(\ln n + \ln p^{-1})\delta^{-2}$, where $0 < p < 1$, the majority decision errs with probability at most p/n.

Let $\tau_1, \ldots, \tau_m$ be the outputs of O_1. Without loss of generality, assume that $\ell_1(\alpha) = 0$. Then the majority decision errs if

$$\frac{1}{m}\sum_{i=1}^{m} \tau_i > \frac{1}{2}. \tag{1}$$

Since for each $\alpha \in \Lambda_N$ the probability that O_1 successfully guesses $\ell_1(\alpha)$ equals $\frac{1}{2} + \delta$ the expected value $\mathrm{E}[\tau_i] = \frac{1}{2} - \delta$, $i = 1, \ldots, m$. (1) implies that

$$\frac{1}{m}\sum_{i=1}^{m} \tau_i - \mathrm{E}[\tau_i] > \delta.$$

Since $\tau_1, \ldots, \tau_m$ are mutually independent Hoeffding's bound [8] gives

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} \tau_i - \mathrm{E}[\tau_i] > \delta\right] \le \exp\left(-2m\delta^2\right).$$

It implies that for $m = \frac{1}{2}(\ln n + \ln p^{-1})\delta^{-2}$ the majority decision errs with probability p/n.

3.3   The Simplified Algorithm

Remark 2. Before describing the simplified inversion algorithm we point out an important fact about oracle O_1. For every $y \in \Lambda_N$ there always exist two different values x_1 and x_2 such that $E_N(x_i) = y$ and $x_i \in \mathbb{Z}_N(+1)$, $i = 1, 2$. Without loss

of generality, let $x_1 < N/2$. Then $x_1 \in \Lambda_N$, $x_2 = N - x_1$. On input y oracle O_1 predicts $\ell_1(x_1)$ rather than $\ell_1(x_2)$. This property will be used on step 3 of the algorithm.

The inversion algorithm, given $E_N(x)$ for some $x \in \Lambda_N$ and parameter p, $0 < p < 1/2$, runs as follows.

1. Pick a random multiplier $a \in_R \mathbb{Z}_N(+1)$. Let $m = \frac{1}{2}(\ln n + \ln p^{-1})\delta^{-2}$.
2. Set $u_0 = 1/2$. $u_0 N$ is a rational approximation of $[ax]_N$ with error at most $N/2$. Set $l_{-1} = 0$.
3. For $t = 0, \ldots, n - 1$ do the following. Compute $E_N([a_t x]_N) = E_N(a_t)E_N(x)$ mod N. Run O_1 on input $E_N([a_t x]_N)$ m times. Let r_t be the majority output bit. Assign $l_t = r_t + l_{t-1} \bmod 2$. Let $a_{t+1} = [2^{-(t+1)}a]_N$. To determine a rational approximation $u_{t+1}N$ for $[a_{t+1}x]_N$, set $u_{t+1} = (u_t + l_t)/2$.
4. Compute $x' = a_n^{-1}\lfloor u_n N + 1/2 \rfloor \bmod N$. If $E_N(x') = E_N(x)$ output x', otherwise repeat the above procedure starting from step 1.

If no error occurs in the above algorithm we have $l_t = \ell_1([a_t x]_N)$ for $t = 0, \ldots, n - 1$. Setting $l_{-1} = 0$ at step 2 means that the algorithm works only if $\ell_1([2ax]_N) = 0$. Since a is chosen at random, $\ell_1([2ax]_N) = 0$ with probability $1/2$.

The goal of step 3 is to determine $\ell_1([a_t x]_N)$. The bit is obtained via the majority decision. Note that on input $E_N([a_t x]_N)$ O_1 predicts either $\ell_1([a_t x]_N)$ (if $\ell_1([a_{t-1}x]_N) = 0$) or $\ell_1(N - [a_t x]_N)$ (if $\ell_1([a_{t-1}x]_N) = 1$). Since $\ell_1(N) = 1$, we have $\ell_1([a_t x]_N) = \ell_1(N - [a_t x]_N) + 1 \bmod 2$. Therefore the majority bit r_t has to be added by l_{t-1} modulo 2 (see also Remark 2).

Since a single majority decision errs with probability at most p/n the probability that $E_N(x') = E_N(x)$ at step 4 is at least $1/2 - p$. Thus the expected running time of the inversion algorithm is at most $(1 - 2p)^{-1}n(\ln n + \ln p^{-1})\delta^{-2}T_{O_1}$, where T_{O_1} is the running time of O_1. For instance, for $p = 1/4$ the running time is essentially $2n(\ln n)\delta^{-2}T_{O_1}$.

Remark 3. The information-theoretic approach of Fischlin and Schnorr [5] implies that inversion of the absolute Rabin function needs to run O_1 at least $(\ln 2/4)n\delta^{-2}$ times. Therefore the running time of the above algorithm is optimal up to a factor of $\ln n$.

4   The Inversion Algorithm

In this section we show how to invert the absolute Rabin function E_N, having access to oracle B that, given $E_N(x)$ for some $x \in_R \Lambda_N$, $j - 1$ least significant bits of x, guesses j-th least significant bit $\ell_j(x)$ with advantage $M^{-1}\varepsilon$.

We build the inversion algorithm for $j \geq 1$ combining the inversion algorithm of [5] for $j = 1$ with the result of [15]. Basic idea of our inversion algorithm is the following. First B is converted into an algorithm O_{xor} that, given $E_N(x)$ for some $x \in \Lambda_N$, guesses the exclusive OR of some subset of first j least significant bits of x with advantage δ, where $\delta = (2^j - 1)^{-1}M^{-1}\varepsilon$. Then $E_N(x)$ is inverted using O_{xor} as an oracle.

4.1   Oracle for Exclusive OR

Let π be a subset of the set of positive integers. For an integer y, $y \geq 0$, let

$$\pi(y) = \sum_{i \in \pi} \ell_i(y) \bmod 2.$$

Note that the subset and the corresponding exclusive OR function are denoted by the same character π.

Let $y \in \Lambda_N$. On input $(\pi, E_N(y))$, where $\pi \subset \{1, \ldots, j\}$ is a nonempty subset, algorithm O_{xor} guesses $\pi(y)$ as follows

1. Select $r_1, \ldots, r_j \in_R \{0, 1\}$. Let r, $0 \leq r < 2^j$, be an integer such that $\ell_k(r) = r_k$, $k = 1, \ldots, j$.
2. Output $\pi(r)$ if $B(E_N(y), r_1, \ldots, r_{j-1}) = r_j$, otherwise output $\pi(r) + 1 \bmod 2$.

The below statement follows explicitly from the Computational XOR Proposition proposed by Goldreich [7].

Lemma 3. *For the above algorithm O_{xor} we have $\Pr[O_{xor}(\pi, E_N(y)) = \pi(y)] = 1/2 + \delta$, $\delta = (2^j - 1)^{-1} M^{-1} \varepsilon$, where the probability is taken over all choices of $y \in_R \Lambda_N$, nonempty subsets $\pi \subseteq \{1, \ldots, j\}$ with uniform probability distribution, and internal coin flips of O_{xor}.*

4.2   Inversion of the Absolute Rabin Function Using O_{xor}

The inversion algorithm described below is based on the same ideas as the simplified inversion algorithm of Section 3. The main difference between these two algorithms is due to the fact that, in comparison with O_1, the advantage of O_{xor} does not have to be the same for all input values. In order to use O_{xor} for the majority decision we have to randomize the input values. For this purpose we use two multipliers $a, b \in_R \mathbb{Z}_N$ and we call O_{xor} on inputs proportional to $E_N(c_{t,i} x)$, where $c_{t,i}$ is a function of a and b such that $c_{t,i}$'s for a fixed t are pairwise independent random variables.

Tightening the Rational Approximations. Suppose $E_N(x)$ is given for $x \in \Lambda_N$. The goal is to recover x.

Let $a, b \in_R \mathbb{Z}_N$. Let $u_0 N$ and vN be rational approximations of $[ax]_N$ and $[bx]_N$. In below algorithm we search through a certain number of quadruples $(u_0 N, vN, l_{a,0}, l_b)$, where $l_{a,0}, l_b \in \{0, 1\}$, so that for at least one of them

$$l_{a,0} = \ell_1([ax]_N), \ l_b = \ell_1([bx]_N),$$
$$|[ax]_N - u_0 N| \leq \eta_a N, \ |[bx]_N - vN| \leq \eta_b N, \tag{2}$$

where $\eta_a = 2^{-j-6} \delta^3, \eta_b = 2^{-j-4} \delta$ (these values result from the analysis of the inversion algorithm, which appears in the extended version of this paper). (2) implies that we have to try at most $\eta_a^{-1} \eta_b^{-1}$ quadruples.

Let $a_t = [2^{-t}a]_N, t = 1, \ldots, n$. By means of the binary division technique we construct rational approximations $u_t N$ for $[a_t x]_N$ so that if (2) holds then

$$|[a_t x]_N - u_t N| \leq \frac{\eta_a N}{2^t}, t = 1, \ldots, n.$$

For $t = n$ we have $|[a_n x]_N - u_n N| < 1/2$, i.e. the closest integer to $u_n N$ is $[a_n x]_N$. Therefore $x = [a_n^{-1} \lfloor u_n N + \frac{1}{2} \rfloor]_N$.

The binary division technique works only if for all $t = 0, \ldots, n-1$ the bits $\ell_1([a_t x]_N)$ are determined. Note that if (2) holds then $\ell_1([ax]_N) = l_{a,0}$. For $t = 1, \ldots, n-1$ we determine the bits $\ell_1([a_t x]_N)$ using oracle O_{xor}.

Finding $\ell_1([a_t x]_N)$ Via Majority Decision. Consider step t of the inversion algorithm for $1 \leq t < n$. At this step we know the rational approximation $u_t N$ for $[a_t x]_N$. The goal is to determine $\ell_1([a_t x]_N)$.

Let i be an integer from a multiset σ_t (we will define the multisets in the end of this section). Using O_{xor} we will determine $\ell_1([a_t x]_N)$ for a fraction of indices $i \in \sigma_t$ with probability slightly higher than $1/2$. Then the majority decision will provide us with a reliable value $\ell_1([a_t x]_N)$. The details follow.

Let $c_{t,i} = a_t(1 + 2i) + b$. Then

$$[c_{t,i} x]_N = [a_t x]_N(1 + 2i) + [bx]_N \bmod N.$$

Let $w_{t,i} = u_t(1 + 2i) + v$, $\tilde{w}_{t,i} = w_{t,i} \bmod 1$. Here $\tilde{w}_{t,i} N$ is an approximation of $[c_{t,i} x]_N$, whereas $w_{t,i} N$ is an approximation of $[a_t x]_N(1 + 2i) + [bx]_N$. Note that if the error of the rational approximation $w_{t,i} N$ is small enough we have

$$[2^j c_{t,i} x]_N = 2^j ([a_t x]_N(1 + 2i) + [bx]_N) - \lfloor 2^j w_{t,i} \rfloor N. \tag{3}$$

We will see that if (3) holds for a certain value of i then the i-th vote in the majority decision is correct with probability $1/2 + \delta$ (this probability cannot be higher since O_{xor} guesses correctly with probability $1/2 + \delta$). In Section 5 we analyze the probability that (3) holds. In the rest of this section we assume that (3) does hold.

It can be shown that if (3) holds then

$$[c_{t,i} x]_N = [a_t x]_N(1 + 2i) + [bx]_N - \lfloor w_{t,i} \rfloor N.$$

Since $\ell_1(N) = 1$, we get

$$\ell_1([c_{t,i} x]_N) = \ell_1([a_t x]_N) + \ell_1([bx]_N) + \lfloor w_{t,i} \rfloor \bmod 2. \tag{4}$$

If (2) holds then $\ell_1([bx]_N) = l_b$ and the only unknown components in (4) are $\ell_1([c_{t,i} x]_N)$ and $\ell_1([a_t x]_N)$. We will determine $\ell_1([c_{t,i} x]_N)$ through O_{xor} and then we will use (4) for the majority decision on $\ell_1([a_t x]_N)$.

Let $\pi_{t,i}$ be a random nonempty subset of $\{1, 2, \ldots, j\}$. Denote $r = \max\{k \mid k \in \pi_{t,i}\}$, $r \leq j$. Each time (for each values of t and i) a new random subset $\pi_{t,i}$ is

selected. The value of r also depends on t and i. We write r instead of $r_{t,i}$ for the sake of simplicity. Denote

$$L_k(y) = y \bmod 2^k$$

for $y \in \mathbb{Z}$, $k = 1, 2, \ldots$. If $y \geq 0$ $L_k(y)$ gives an integer that equals k least significant bits of y.

Lemma 4. *If (3) holds then*

$$\ell_1([c_{t,i}x]_N) = \pi_{t,i}([2^{r-1}c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2.$$

We prove this lemma in Appendix A. Lemma 4 combined with (4) gives

$$\ell_1([a_t x]_N) = \pi_{t,i}([2^{r-1}c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) +$$
$$\ell_1([bx]_N) + \lfloor w_{t,i}\rfloor \bmod 2.$$

If $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$ in the above formula then we can replace $\pi_{t,i}([2^{r-1}c_{t,i}x]_N)$ by $O_{xor}(\pi_{t,i}, E_N([2^{r-1}c_{t,i}x]_N))$. However, since the output bit of O_{xor} is not always correct we have to use some form of majority decision to determine $\ell_1([a_t x]_N)$.

The majority decision on bit $\ell_1([a_t x]_N)$ works as follows. If for majority of indices $i \in \sigma_t$ such that $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$

$$O_{xor}(\pi_{t,i}, E_N([2^{r-1}c_{t,i}x]_N)) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) +$$
$$\ell_1([bx]_N) + \lfloor w_{t,i}\rfloor = 0 \bmod 2, \tag{5}$$

the inversion algorithm decides that $\ell_1([a_t x]_N) = 0$, otherwise it decides that $\ell_1([a_t x]_N) = 1$.

Note that we can check if $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$ as follows. By definition, $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$ if $2^{r-1}c_{t,i} \in \mathbb{Z}_N(+1)$ and $[2^{r-1}c_{t,i}x]_N < \frac{N}{2}$. The first condition can be checked by computing Jacobi symbol of $2^{r-1}c_{t,i}$ modulo N. We check the second condition via the rational approximation of $[2^{r-1}c_{t,i}x]_N$. It can be shown that if (3) holds then for all r, $0 \leq r < j$, $[2^{r-1}c_{t,i}x]_N < \frac{N}{2}$ if and only if $\lfloor 2^r w_{t,i}\rfloor$ is even. If $[2^{r-1}c_{t,i}x]_N \notin \Lambda_N$ we discard the index i. Since $c_{t,i}$ is uniformly distributed in $\mathbb{Z}_N$, $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$ with probability $1/4$ (each of the above conditions is satisfied with probability $1/2$).

Multisets σ_t. In this section we define the multisets σ_t, $t = 1, \ldots, n - 1$. For $t < \log_2 n + 4$ denote $m_t = 4 \cdot 2^t \delta^{-2}$. Let

$$\sigma_t = \{i \mid |1 + 2i| < m_t\}, \ t = 1, \ldots, \log_2 n + 3.$$

As t grows we choose a larger value for m_t. Therefore the majority decisions become more reliable as t grows. We cannot choose large m_t for small t for the following reason. For small t the error $|u_t N - [a_t x]_N|$ is large. If m_t is also large

then $[a_t x]_N (1 + 2i) + [bx]_N$ can differ much from $w_{t,i} = u_t(1 + 2i) + v$ so that
(3) does not hold and (5) cannot be used for the majority decision.

Define $\rho = \{i \mid |1 + 2i| < 2^6 n \delta^{-2}\}$. We randomly select $m = 8 \delta^{-2} \log_2 n$
elements $\sigma = \{i_1, \ldots, i_m\}$ *with repetition* from ρ and let

$$\sigma_t = \sigma, \ m_t = m, \ t = \log_2 n + 4, \ldots, n - 1.$$

For $t = 1, \ldots, n - 1$ $|\sigma_t| = m_t$. For $t \geq \log_2 n + 4$ all the σ_t are the same, the
number of elements (not necessarily different) in this multiset is m.

Note that there exist two basic bounds for error probabilities of major-
ity decisions: Hoeffding's bound and Chebyshev's inequality. Hoeffding's bound
(see also Section 3.2) is asymptotically stronger than Chebyshev's inequality.
However, Hoeffding's bound requires mutual independence of the votes. For
$t = \log_2 n + 4, \ldots, n - 1$ the multisets σ_t are chosen in such a way that Ho-
effding's bound can be used. For $t < \log_2 n + 4$ the votes are just pairwise
independent so only Chebyshev's inequality can be used. However, as mentioned
above, the number of votes cannot be large for small t so in this case we can-
not gain from using Hoeffding's bound rather than Chebyshev's inequality. This
issue is addressed in more details in Section 5.

4.3   The Algorithm

In this section we formally describe the inversion algorithm. Suppose we know
$E_N(x)$ for some $x \in \Lambda_N$. Let O_{xor} be an algorithm that, given $E_N(x)$ for some
$x \in \Lambda_N$ and a subset $\pi \subset \{1, \ldots, j\}$, guesses $\pi(x)$ with advantage δ. The inversion
algorithm that uses O_{xor} as an oracle and outputs x runs as follows.

```
Input E_N(x), N, j and oracle O_xor
---- First part: oracle calls ----
Select random integers a, b ∈ Z_N
For t = 1, ..., n do
    a_t = [2^{-t} a]_N
    For i ∈ σ_t do
        c_{t,i} = a_t(1 + 2i) + b
        If (2^{r-1} c_{t,i} / N) = +1 then
            Select a random nonempty subset π_{t,i} ⊂ {1,...j}
            Set r = max{k | k ∈ π_{t,i}}
            g_{t,i} = O_xor(π_{t,i}, E_N([2^{r-1} c_{t,i} x]_N)), validity bit d_{t,i} = 1
        Else
            d_{t,i} = 0
        End if
    End do
End do
---- Second part: tightening the rational approximations ----
For ũ = 0, ..., ⌊η_a^{-1}/2⌋; ṽ = 0, ..., ⌊η_b^{-1}/2⌋; l_{a,0} = 0, 1; l_b = 0, 1 do
    Reset d_{t,i} with the values calculated in the first part
```

```
Rational u = 2η_a ũ, v = 2η_b ṽ, set u_0 = u
For t = 1, ..., n - 1 do
    Rational u_t = ½(l_{a,t-1} + u_{t-1})
    For i ∈ σ_t such that d_{t,i} = 1 do
        Rational w_{t,i} = u_t(1 + 2i) + v
        If ⌊2^r w_{t,i}⌋ = 0 mod 2 then
            Set r = max{k | k ∈ π_{t,i}}, assign w̃_{t,i} = w_{t,i} mod 1
            e_i = l_b + π_{t,i}(L_r(-⌊2^{r-1}w̃_{t,i}⌋N)) + ⌊w_{t,i}⌋ mod 2
        Else
            d_{t,i} = 0
        End if
    End do
    l_{a,t} = MajorityDecision(g_{t,*} + e_* mod 2, d_{t,*})
End do
x' = [a_n^{-1}⌊u_n N + ½⌋]_N
If (x'/N) = +1 and E_N(x') = E_N(x) then output x'
End do
```

On step t the goal of the algorithm is to determine $\ell_1([a_t x]_N)$. This bit is determined via majority decision. Note that $e_i = l_b + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) + \lfloor w_{t,i}\rfloor \bmod 2$ and $g_{t,i} = O_{xor}(\pi_{t,i}, E_N([2^{r-1}c_{t,i}x]_N))$ (see also (5)). If for a majority of indices $i \in \sigma_t$ such that $d_{t,i} = 1$ we have $g_{t,i} = e_i$ then the majority decision outputs 0, otherwise it outputs 1 (in terms of the above algorithm $d_{t,i} = 1$ if and only if $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$). If the majority decision is correct then $l_{a,t} = \ell_1([a_t x]_N)$.

5   Analysis of the Inversion Algorithm

In this section we determine the success probability of the above inversion algorithm. More formally, we prove the following lemma.

Lemma 5. *The above algorithm, given $E_N(x)$ for $x \in \Lambda_N$, j, and N, outputs x with probability 2/9, where the probability is taken over internal coin flips of the algorithm (which includes the coin flips of O_{xor}.*

Recall that the inversion algorithm works as follows. For $a, b \in_R \mathbb{Z}_N$, we search through a certain number of quadruples $(u_0 N, vN, l_{a,0}, l_b)$ such that for at least one of them (2) holds, i.e. $l_{a,0} = \ell_1([ax]_N)$, $l_b = \ell_1([bx]_N)$; $|[ax]_N - u_0 N| \le \eta_a N$, $|[bx]_N - vN| \le \eta_b N$. Throughout this section we only consider a quadruple for which (2) holds (for the other quadruples we assume that the algorithm outputs x with probability 0).

At each step t, $1 \le t < n$, the goal of the inversion algorithm is to determine $\ell_1([a_t x]_N)$. Using O_{xor} this bit is determined via the majority decision, which depends on a certain number of votes. For $i \in \sigma_t$ such that $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$, the i-th vote is set to 0 if (5) holds, otherwise it is set to 1. The decision on

$\ell_1([a_t x]_N)$ is set to the majority vote. The decision is correct if the majority of the votes is correct.

Consider step t, $1 \leq t < n$. Assume that for all $s < t$ we have determined correctly the bits $\ell_1([a_s x]_N)$. There exist two reasons why for some $i \in \sigma_t$ the i-th vote could be incorrect.

- The error of the rational approximation $w_{t,i} N$ is too large so that (3) does not hold.
- Oracle O_{xor} outputs a wrong bit (recall that it outputs the correct bit only with probability $1/2 + \delta$).

5.1   The Probability That (3) Does Not Hold

Lemma 6. *Assume that (2) holds and for all $s < t$ the bits $\ell_1([a_s x]_N)$ are determined correctly. Then the probability that (3) does not hold for some $i \in \sigma_t$ is at most $\delta/4$. Here the probability is taken over all choices of random multipliers $a, b \in_R \mathbb{Z}_N$.*

Proof. Let us rewrite (3) again:

$$[2^j c_{t,i} x]_N = 2^j \left([a_t x]_N (1 + 2i) + [bx]_N\right) - \lfloor 2^j w_{t,i} \rfloor N.$$

Intuitively, (3) does not hold if there exists a multiple of N between $2^j([a_t x]_N(1 + 2i) + [bx]_N)$ and $2^j w_{t,i} N$. Denote $\Delta_{t,i} = 2^j w_{t,i} N - 2^j ([a_t x]_N (1 + 2i) + [bx]_N)$. Then (3) does not hold if and only if

$$\left| \Delta_{t,i} \right| \geq \left| 2^j ([a_t x]_N (1 + 2i) + [bx]_N) \right|_N = \left| 2^j c_{t,i} x \right|_N,$$

where for $z \in \mathbb{Z}$ $|z|_N = \min([z]_N, N - [z]_N)$ denotes the distance from z to the closest multiple of N.

If (2) holds and for all $s < t$ we have determined correctly the bits $\ell_1([a_s x]_N)$ then

$$\left| [a_t x]_N - u_t N \right| = 2^{-t} \left([ax]_N - u_0 N \right) \leq 2^{-t-j-6} \delta^3 N,$$
$$\left| [bx]_N - v \right| = 2^{-j-4} \delta N.$$

Since $2^{-t} \delta^2 |1 + 2i| \leq 4$ for $i \in \sigma_t$ (see Section 4.2) the triangular inequality gives

$$\left| \Delta_{t,i} \right| = 2^j \left| u_t N(1 + 2i) - [a_t x]_N(1 + 2i) + vN - [bx]_N \right| \leq$$
$$\frac{\delta}{64}(2^{-t}\delta^2|1 + 2i| + 4)N \leq \frac{\delta}{8}N.$$

Thus (3) does not hold only if $\left| 2^j c_{t,i} x \right|_N \leq \delta N/8$. Since $c_{t,i}$ is uniformly distributed in $\mathbb{Z}_N$, the probability that (3) does not hold is at most $\delta/4$. It completes the proof of Lemma 6.

5.2   Error Probability of the Majority Decisions

Throughout this section we will refer to indices i such that $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$ as *valid indices*. The i-th vote in the majority decision on $\ell_1([a_t x]_N)$ is correct if (3) holds and the reply of O_{xor} is correct. Following the notation of [5] we define boolean variables τ_i such that $\tau_i = 1$ only if the i-th vote is incorrect:

$\tau_i = 1$ if and only if (3) does not hold or $O_{xor}([c_{t,i}x]_N, \pi_{t,i}) \neq \pi_{t,i}([c_{t,i}x]_N)$.

It is shown [5] that for any fixed t, $1 \leq t < n$, the multipliers $c_{t,i}$ are pairwise independent. Thus boolean variables τ_i, $i \in \sigma_t$, are also pairwise independent.

   Let μ_t be the number of valid indices $i \in \sigma_t$. The majority decision errs only if

$$\frac{1}{\mu_t} \sum_{\text{valid } i \in \sigma_t} \tau_i > \frac{1}{2}.$$

Due to the different choice of σ_t for $t < \log_2 n + 4$ and for $t \geq \log_2 n + 4$ (see Section 4.2) we divide our analysis into two parts.

Case $t < \log_2 n + 4$. Consider a step $t < \log_2 n + 4$. Since O_{xor} guesses correctly with probability $\frac{1}{2} + \delta$, Lemma 6 implies that the expected value $\mathrm{E}[\tau_i] \leq 1/2 - 3\delta/4$. The majority decision errs only if

$$\frac{1}{\mu_t} \sum_{\text{valid } i \in \sigma_t} \tau_i - \mathrm{E}[\tau_i] \geq \frac{3}{4}\delta.$$

Since the variance of any boolean variable is at most $1/4$, $\mathrm{Var}[\tau_i] \leq 1/4$. Chebyshev's inequality for μ_t pairwise independent random variables τ_i gives

$$\Pr\left[\frac{1}{\mu_t} \sum_{\text{valid } i \in \sigma_t} \tau_i - \mathrm{E}[\tau_i] \geq \frac{3}{4}\delta\right] \leq \left(\frac{3}{4}\delta\right)^{-2} \mathrm{Var}\left[\frac{1}{\mu_t} \sum_{\text{valid } i \in \sigma_t} \tau_i\right] \leq \frac{4}{9\mu_t\delta^2}.$$

Here the probability is taken over all choices of random multipliers $a, b \in_R \mathbb{Z}_N$, and internal coin flips of O_{xor}.

   Since on average $\mu_t = m_t/4 = 2^t\delta^{-2}$, the majority decision for $\ell_1([a_t x]_N)$ errs with probability $\frac{4}{9}2^{-t}$. Thus the probability that at least one of the majority decisions for $t < \log_2 n + 4$ errs is at most $4/9$.

Case $t \geq \log_2 n + 4$. The technique we use for $t \geq \log_2 n + 4$ is called subsample majority decision. It is proposed by Fischlin and Schnorr [5].

   Consider a step $t \geq \log_2 n + 4$. Instead of using indices from a large sample $\rho = \{i \mid |1 + 2i| < 2^6 n\delta^{-2}\}$ we use only indices from a small random subsample $\sigma = \{i_1, \ldots, i_m\} \subset \rho$, where $m = 8\delta^{-2} \log_2 n$ (see also Section 4.2). Although original τ_i, $i \in \rho$, are just pairwise independent $\tau_{i_1}, \ldots, \tau_{i_m}$ are mutually independent. Therefore for these random variables we can use a stronger bound instead of Chebyshev's inequality, namely Hoeffding's bound [8].

Let μ_t be the number of valid indices $i \in \sigma$ (the number of $i \in \sigma$ such that $[2^{r-1}c_{t,i}x]_N \in \Lambda_N$). The majority decision errs only if

$$\frac{1}{\mu_t} \sum_{\text{valid } i_s \in \sigma} \tau_{i_s} - \mathrm{E}[\tau_i] \geq \frac{3}{4}\delta,$$

Let ν_t denote the number of valid indices in ρ. Denote

$$\tau = \frac{1}{\nu_t} \sum_{\text{valid } i \in \rho} \tau_i,$$

where $|\rho| = 2^6 n \delta^{-2}$. The majority decision errs if either $\tau - \mathrm{E}[\tau] \geq \delta/4$ or

$$\frac{1}{\mu_t} \sum_{\text{valid } i_s \in \sigma} \tau_{i_s} - \tau \geq \frac{1}{2}\delta.$$

Chebyshev's inequality for pairwise independent τ_i, $i \in \rho$, gives $\Pr[\tau - \mathrm{E}(\tau) \geq \delta/4] \leq 4/(\nu_t \delta^2)$. Hoeffding's bound [8] implies that for fixed τ_i, $i \in \rho$, and a random subsample $\sigma \subset \rho$

$$\Pr\left[\frac{1}{\mu_t} \sum_{\text{valid } i_s \in \sigma} \tau_{i_s} - \tau \geq \frac{1}{2}\delta\right] \leq \exp\left(-2\mu_t\left(\frac{\delta}{2}\right)^2\right) = \exp\left(-\frac{1}{2}\mu_t\delta^2\right). \quad (6)$$

Since on average $\mu_t = m/4$ and $\nu_t = |\rho|/4$ (on average only $1/4$ of the indices are valid) the majority decision at each step $t \geq \log_2 n + 4$ errs with probability at most $16/(|\rho|\delta^2) + \exp(m\delta^2/8) = 1/(4n) + n^{-1/\ln 2} < 1/(3n)$ for $n > 2^9$. Thus the probability that at least one of the subsample majority decisions for $t \geq \log_2 n + 4$ errs is at most $1/3$.

Therefore the inversion algorithm of Section 4, given $E_N(x)$, j, and N, outputs x with probability at least $1 - (4/9 + 1/3) = 2/9$. It completes the proof of Lemma 5.

6   Complexity of the Inversion Algorithm

In this section we determine the running time of the inversion algorithm. The unit of time we use throughout this paper is a clock cycle.

The first part of the algorithm (oracle calls) consist of n steps $t = 1, \ldots, n$. On average we run the algorithm O_{xor} $m_t/2 = 2 \cdot 2^t \delta^{-2}$ times for $t < \log_2 n + 4$ and $m/2 = 4\delta^{-2} \log_2 n$ for $t \geq \log_2 n + 4$, therefore in total we run O_{xor} $32n\delta^{-2} + 4n(\log_2 n)\delta^{-2} \approx 4n(\log_2 n)\delta^{-2}$ times. Note that the number of oracle calls does not depend on the number of quadruples $(u, v, \ell_1([ax]_N), \ell_1([bx]_N))$.

In the second part (tightening the rational approximations) we do not use O_{xor} but we run a large exhaustive search cycle. The bottleneck of the second part is multiplication $\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor \cdot N$. The size (in bits) of $\tilde{w}_{t,i}$ is $\log_2(\eta_b^{-1}) = \log_2(\delta^{-1}) + j + 4$ and the size of N is n. For instance, for $\varepsilon = 1/2$, $M = 2^{20}$, $j = 5$ we have $\delta = 2^{-26}$ and hence $\log_2(\eta_b^{-1}) = 35$. Therefore we may assume that a single multiplication takes n clock cycles. Hence the complexity of the second part is at most the product of the following factors:

1. Number of quadruples $(u, v, \ell_1([ax]_N), \ell_1([bx]_N))$, that is $\eta_a^{-1}\eta_b^{-1}$;
2. Number of steps t, that is n;
3. Number of votes for the majority decision, that is $m/4 = 2\delta^{-2}\log_2 n$;
4. Complexity of the multiplication $\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor \cdot N$, that is n;

Since $\eta_a = 2^{-j-6}\delta^3, \eta_b = 2^{-j-4}\delta$, the complexity of the second part is $2^{11}\delta^{-6}n^2\log_2 n$ clock cycles. Recall that the running time of O_{xor} is essentially the same as the one of B. Thus the running time of the inversion algorithm is $4n(\log_2 n)\delta^{-2}(T_B + 2^{2j+9}n\delta^{-4})$. Lemma 5 implies that there exists algorithm R that inverts the absolute Rabin function in expected time

$$T_R \leq 18n(\log_2 n)\delta^{-2}(T_B + 2^{2j+9}n\delta^{-4}). \tag{7}$$

Remark 4. The argument of [5] implies that inversion of the absolute Rabin function needs at least $(\ln 2/4)n\delta^{-2}$ runs of B (see also Remark 3). Therefore the number of oracle runs in the above inversion algorithm is optimal up to a factor of $\log_2 n$. However, in (7) we also have a second component that cannot be neglected in practice.

7   Concrete Security of the BBS

In this section we state our main result about the concrete security of the BBS pseudorandom generator. We give a bound for running time T_A and advantage ε such that the BBS generator is (T_A, ε)-secure (Theorem 3).

Theorem 1. *Suppose the BBS pseudorandom generator is not (T_A, ε)-secure. Then there exist an algorithm F that factors the modulus N in expected time*

$$T_F \leq 36n(\log_2 n)\delta^{-2}(T_A + 2^{2j+9}n\delta^{-4}),$$

where $\delta = (2^j - 1)^{-1}M^{-1}\varepsilon$.

Proof. The statement follows from (7), Lemma 1, and Lemma 2.

Therefore a statistical test that distinguishes the BBS sequences from random sequences can be used to factor the modulus N. However, we observe that the reduction is not tight in the sense that for a practical choice of parameters $T_F \gg T_A$. Furthermore, Remark 4 implies that the reduction for the BBS generator cannot be significantly tighter. There is a large gap between security of this pseudorandom generator and the factoring problem.

In order to complete the concrete security analysis of the BBS generator we will assume a concrete lower bound on the hardness of integer factoring, which is obtained by extrapolating the best factorization results to date.

7.1   Hardness of Factoring

The fastest general-purpose factoring algorithm today is the general number field sieve. According to [4,11] on heuristic grounds the number field sieve is expected to require time proportional to $\gamma \exp((1.9229 + o(1))(\ln N)^{1/3}(\ln \ln N)^{2/3})$ for a constant γ. Following [4] we make an assumption that the $o(1)$-term can be treated as zero. From this we can calculate γ.

Let $L(n)$ be the number of clock cycles needed to factor an n-bit integer. We assume that $L(n) \approx \gamma \exp(1.9229(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3})$. Experience from available data points suggests that $L(512) \approx 3 \cdot 10^{17}$ clock cycles, therefore $\gamma \approx 2.8 \cdot 10^{-3}$ and

$$L(n) \approx 2.8 \cdot 10^{-3} \cdot \exp(1.9229(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3}). \tag{8}$$

Assumption 2. *No algorithm can factor a randomly chosen n-bit Blum-integer in expected time $T < L(n)$, where $L(n)$ is given by (8).*

All the results below hold under the above assumption.

Theorem 3 (Concrete security of the BBS). *Under Assumption 2, the BBS pseudorandom generator is (T_A, ε)-secure if*

$$T_A \leq \frac{L(n)}{36n(\log_2 n)\delta^{-2}} - 2^{2j+9}n\delta^{-4}, \tag{9}$$

where $\delta = (2^j - 1)^{-1}M^{-1}\varepsilon$.

7.2   Comparison with Known Results

Thus we have shown that there exist a reduction a successful attack on the BBS generator to factoring. While for asymptotic security it suffices to give a polynomial time reduction, we need for concrete security a reduction that is as efficient as possible. In this subsection we compare the complexity of our reduction, given by Theorem 3, with the results of [1,15,5].

A close look at the security proof of Alexi et al. [1] gives the following lemma.

Lemma 7 (Alexi et al.). *Under Assumption 2, the BBS pseudorandom generator is (T_A, ε)-secure if*

$$T_A \leq \frac{L(n)}{2^{27}2^{4j}n^3\varepsilon^{-8}M^8}. \tag{10}$$

Recall that M denotes the length of the output of the BBS generator (e.g., $M = 2^{20}$). Formula (10) has M^8 in the denominator whereas (9) has M^2. Thus our security proof is stronger than the one of [1].

A disadvantage of [15] is that this paper deals only with *deterministic* statistical tests, thus the result can not be expressed in terms of Definition 3. Furthermore [15] uses [1] as a building block so the complexity of the reduction proposed is of the same order.

The lemma below is due to Fischlin and Schnorr [5]. It establishes the concrete security of the BBS generator with 1 output bit per iteration.

Lemma 8 (Fischlin and Schnorr). *Under Assumption 2, the BBS pseudo-random generator with 1 output bit per iteration is (T_A, ε)-secure if*

$$T_A \leq \frac{L(n)}{6n(\log_2 n)\varepsilon^{-2}M^2} - 2^7 n\varepsilon^{-2}M^2 \log_2(8n\varepsilon^{-1}M). \qquad (11)$$

Here the denominator of the first component in the righthand side is essentially the same as the one in (9) for $j = 1$. The second component in (11) is smaller in the absolute value than the second component in (9) by a factor of $\varepsilon^{-2}M^{-2}$. The reason is that there is a trick in the reduction [5] (namely, processing all approximate locations simultaneously) that allows to decrease the second component. We do not know if it is possible to apply this trick for $j > 1$ and we leave this question as an open problem. In the below example the factor of $\varepsilon^{-2}M^{-2}$ in the second component does not affect the final conclusion about the optimal value of j.

7.3   Example

An important application of Theorem 3 is that it can be used to determine the optimal values for the parameters of the BBS generator.

Suppose our goal is to generate a sequence of $M = 2^{20}$ bits such that no adversary can distinguish this sequence from truly random binary sequence in time $T_A = 2^{100}$ clock cycles with advantage $\varepsilon = 1/2$. The question is what length of the modulus n and parameter j should be used to minimize the computational effort per output bit.

Inequalities (9) and (11) connect the security parameters (T_A, ε) with parameters of the BBS (M, n, j) for $j \geq 1$ and $j = 1$ respectively. In order to find the optimal n and j we fix T_A, ε, M and consider n as a function of j.
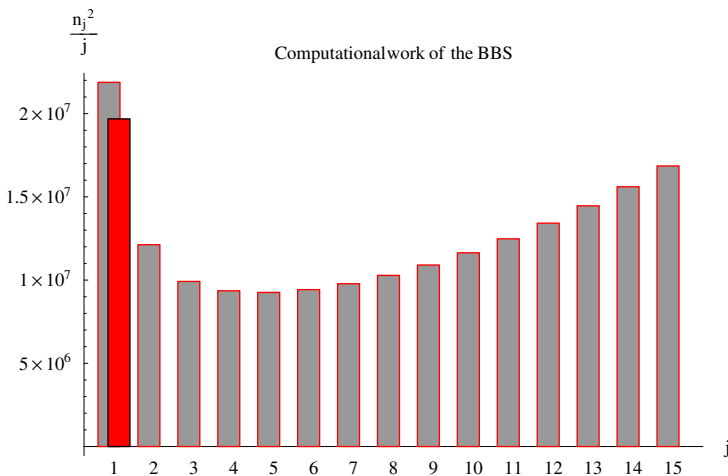

Fig. 1. The computational work of the BBS is minimal for $j = 5$

The computational work of the BBS (the running time needed to output a pseudorandom sequence) is proportional to n^2/j (each modular multiplication costs $O(n^2)$ binary operations so the generation of a BBS sequence takes $O(Mn^2/j)$ operations). Figure 1 displays the computational work of the BBS for $j \in [1, 15]$. There are two values of the computational work for $j = 1$ on the figure. The smaller value results from (11) and the larger one results from (9). For $j = 1$ the reduction [5] is more efficient. Nevertheless, the difference turns out to be not significant and we observe that extraction of 5 bits per iteration makes the BBS about 2 times faster in comparison with 1 bit case. However, even for $j = 5$ the BBS is quite slow since the corresponding length of the modulus $n = 6800$.

It is not true that it always pays off to extract more that 1 bit on each iteration. The optimal number of bits to be extracted on each iteration depends on the length of the output sequence M and on the security parameters T_A, ε. For instance, for $M = 2^{30}$ it turns out that the best choice is $j = 1$.

8   Concluding Remarks and Open Problems

Security of the BBS genarator has been thoroughly analyzed [1,2,5,6]. Nevertheless, it has been uncertain how to select the size of the modulus and the number of bits extracted on each iteration such that a desired level of security is reached, while minimizing the computational effort per output bit. In this paper we answer this question.

Generalizing the ideas of [5,15] we propose a new security proof for the BBS generator with several output bits per iteration. The new reduction is more efficient than all previously known reductions.

With minor changes our argument can be applied for the analysis of the RSA pseudorandom generator.

However, the new reduction is still not tight. There is a large gap between the security of BBS generator and the factoring problem. Moreover, using the information-theoretic approach, Fischlin and Schnorr [5] show that the security reduction for BBS cannot be significantly tighter. Searching for the constructions based on the factoring problem with tight security reduction is one of the challenging problems in the theory of provably secure pseudorandom generators.

Acknowledgements

References

1. W. Alexi, B. Chor, O. Goldeich and C. P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole. SIAM Journal on Computing 17, 1988, pp. 194–209.
2. L. Blum, M. Blum and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. SIAM Journal on Computing 15, 1986, pp. 364–383.

 3. M. Bellare and P. Rogaway. The exact security of digital signatures how to sign with RSA and Rabin. In Advances in Cryptology – Eurocrypt 1996, volume 1070 of Lecture Notes in Computer Science, pp. 399 – 416. Berlin: Springer-Verlag, 1996.
 4. ECRYPT Yearly Report on Algorithms and Keysizes (2004). Available at `http://www.ecrypt.eu.org/documents/D.SPA.10-1.1.pdf`.
 5. R. Fischlin and C. P. Schnorr. Stronger Security Proofs for RSA and Rabin Bits. Journal of Cryptology (2000) 13, pp. 221–244.
 6. R. Gennaro. An improved pseudo-random generator based on discrete log. In Mihir Bellare, editor, Advances in Cryptology – Crypto 2000, volume 1880 of Lecture Notes in Computer Science, pp. 469 – 481. Springer-Verlag, 20–24 August 2000.
 7. O. Goldreich. Three XOR-Lemmas – An Exposition. Technical report, ECCC TR95-056, 1995. Available at `ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1995/TR95-056/Paper.pdf`.
 8. W. Hoeffding. Probability in Equations for Sums of Bounded Random Variables. Journal of the American Statistical Association 56 (1963), pp. 13–30.
 9. D. E. Knuth. Seminumerical Algorithms, 3rd edition. Addison-Wesley, Reading, MA, 1997.
10. J. Katz, N. Wang. Efficiency Improvements for Signature Schemes with Tight Security Reductions. CCS'03, October 27-30, 2003, Washington, DC, USA.
11. A. K. Lenstra, E. R. Verheul. Selecting Cryptographic Key Sizes. Journal of Cryptology (2001) 14: 255–293.
12. A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. Handbook of Appied Cryptography. CRC Press series on discrete mathematics and its applications, 2000.
13. M. O. Rabin. Digitalized signatures and public-key functions as intractible as factorization. Technical report, TR-212, MIT Laboratory for Computer Science, 1979.
14. V. Shoup. On the security of a practical identification scheme. In Advances in Cryptology – Eurocrypt 1996, volume 1070 of Lecture Notes in Computer Science, pp. 344 – 353. Berlin: Springer-Verlag, 1996.
15. U. V. Vazirani, V. V. Vazirani. Efficient and Secure Pseudo-Random Number Generation. Proceedings 25th Symposium on Foundations of Computing Science IEEE, pp. 458–463, 1984.
16. A. C. Yao. Theory and Application of Trapdoor Functions. Proceedings of IEEE Symposium on Foundations of Computer Science, pp. 80–91, 1982.

A   Proof of Lemma 4

Lemma 4 states that if (3) holds then

$$\pi_{t,i}([2^{r-1}c_{t,i}x]_N) = \ell_1([c_{t,i}x]_N) + +\pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2.$$

To prove this lemma we will show that

$$\pi_{t,i}([2^{r-1}c_{t,i}x]_N) = \pi_{t,i}(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N) \qquad (12)$$

and

$$\ell_1([c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) = \\ \pi_{t,i}(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N) \bmod 2. \qquad (13)$$

It can be shown that if (3) holds then for all r, $0 \leq r \leq j$,

$$[2^{r-1}c_{t,i}x]_N = 2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N. \tag{14}$$

Applying function $\pi_{t,i}$ to both sides of (14) gives (12). To prove (13) we first note that

$$\begin{aligned} L_r(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N) = \\ (2^{r-1}\ell_1([c_{t,i}x]_N) + L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N)) \bmod 2^r. \end{aligned} \tag{15}$$

From (14) follows that $2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N \geq 0$. Hence in this case L_r corresponds to r least-significant bits. Thus applying function $\pi_{t,i}$ to the left-hand side of (15) gives

$$\pi_{t,i}(L_r(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N)) = \pi_{t,i}(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N). \tag{16}$$

Then we apply $\pi_{t,i}$ to the right-hand side of (15):

$$\begin{aligned} \pi_{t,i}((2^{r-1}\ell_1([c_{t,i}x]_N) + L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N)) \bmod 2^r) = \\ \pi_{t,i}(2^{r-1}\ell_1([c_{t,i}x]_N) + L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N)) = \\ \ell_1([c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N)) \bmod 2, \end{aligned} \tag{17}$$

since $\pi_{t,i} \subset \{1, \ldots, r\}$, $r \in \pi_{t,i}$. (15), (16), and (17) result in (13). It completes the proof of Lemma 4.