An Improved Pseudorandom Generator Based on Hardness of Factoring

Nenad Dedić
Boston University
nenad@cs.bu.edu

Leonid Reyzin
Boston University
reyzin@cs.bu.edu

Salil Vadhan
Harvard University
salil@eecs.harvard.edu

March 10, 2003

Abstract

We present a simple to implement and efficient pseudorandom generator based on the factoring assumption. It outputs more than $pn/2$ pseudorandom bits per p exponentiations, each with the same base and an exponent shorter than $n/2$ bits. Our generator is based on results by Håstad, Schrift and Shamir [HSS93], but unlike their generator and its improvement by Goldreich and Rosen [GR00], it does not use hashing or extractors, and is thus simpler and somewhat more efficient.

In addition, we present a general technique that can be used to speed up pseudorandom generators based on iterating one-way permutations. We construct our generator by applying this technique to results of [HSS93]. We also show how the generator given by Gennaro [Gen00] can be simply derived from results of Patel and Sundaram [PS98] using our technique.

1 Introduction

1.1 Background

Blum and Micali [BM84] and Yao [Yao82] introduced the notion of a pseudorandom generator secure against all polynomial-time adversaries. Since then, multiple constructions have been proposed. We do not survey them all here; rather, we focus on the ones that are of particular relevance to our work.

Almost all of the proposed constructions (starting with discrete-logarithm-based one of [BM84]) consist of repeatedly applying some one-way function and outputting its hardcore bits. This method is often called the "Blum-Micali" paradigm. Two notable exceptions are generators of Micali and Schnorr [MS91] and Gennaro [Gen00]. The work of [MS91] introduced pseudorandom generators "of weaning type". Such generators, like the Blum-Micali paradigm, also apply a one-way permutation in each iteration and output some bits of the result; unlike the Blum-Micali paradigm, however, they do not reuse the bits that are output as input to the next iteration, and can potentially be more efficeint. The constructions of such generators given in [MS91] are based on strong assumptions that inherently incorporate pseudorandomness. The work of [Gen00] gives a very efficient construction of a generator of the weaning type based on the somewhat strong assumption that discrete logarithms with short exponents are hard to compute. We present a new generator of the weaning type based on just the factoring assumption.

Factoring-based generators have a long history, starting with the generators of Yao [Yao82] (based on the quadric-residuousity assumption) and of Goldwasser, Micali and Tong [GMT82] (based on the factoring assumption). A more efficient generator, proposed by Blum, Blum and Shub [BBS86], iterates modular squaring with an n-bit modulus and extracts one bit of output per iteration. Originally proven secure based on the quadratic residuosity assumption, it was later improved by [VV84, ACGS88], who showed that that only the factoring assumption is needed and that $O(\log n)$ bits could be extracted per iteration. The [BBS86] generator is one of the most efficient provably secure pseudorandom generators known today (with the [VV84, ACGS88] improvements).

Håstad, Schrift and Shamir [HSS93] demonstrated that discrete logarithm modulo a product of two primes hides $n/2 + O(\log n)$ bits, based only on the factoring assumption. They suggested how to construct a pseudorandom generator based on this result by repeatedly exponentiating a fixed base g to an n-bit exponent; however, their construction required the use of hash function families to obtain uniformly pseudorandom bits (because the result of modular exponentiation could not be used in the next iteration of the generator, as it was not indistinguishable from a uniformly distributed n-bit string, nor from a uniformly distributed value in Z_N). This resulted in a loss of $\Theta(\log^2 n)$ bits per iteration, thus giving a generator that output $n/2 - \Theta(\log^2 n)$ pseudorandom bits per fixed-base modular exponentiation and one hashing. Note that having the same base for each modular exponentiation is important, because it allows for precomputation (as further described in Section 4).

Goldreich and Rosen [GR00] further improved this generator by demonstrating, in particular, that one can use exponents of length $n/2$ instead of full-length exponents. However, families of hash functions (or, rather, extractors) were still necessary, thus resulting in a loss of efficiency of each iteration, and the number of bits obtained. Namely, their generator obtains $n/2 - O(\log^2 n)$ pseudorandom bits per a fixed-base modular exponentiation (with half-length exponent) and one application of extractor.

1.2 Our Contributions

We improve the pseudorandom generators of [HSS93] and [GR00] by removing the need for hashing or extractors. Thus, our generator obtains $n/2 + O(\log n)$ bits of randomness per half of a fixed-base modular exponentiation (to be precise, our exponent is $n/2 - O(\log n)$ bits long). The resulting construction is thus simpler and faster than the ones of [HSS93] and [GR00].

Our generator is quite similar to the one of Gennaro [Gen00]: it also essentially repeatedly raises a fixed base to a short exponent, outputs some bits of the result, and uses the rest as an exponent for the next iteration. The main difference is that Gennaro's generator, while more efficient than ours, works modulo a prime and requires the nonstandard assumption that discrete logarithms with short exponents are hard. Our generator, on the other hand, requires only the assumption that factoring products of safe primes is hard. As explained later in the text, that is a trivial consequence of the standard factoring assumption, if safe primes are frequent. Gennaro's generator also requires the assumption that safe primes are frequent.

We present a more general technique, which abstracts and clarifies the analysis of our generator and those of [Gen00, GR00]. The technique is a modification of the Blum-Micali paradigm [BM84] for constructing pseudorandom generators by iterating a one-way permutation and outputting hardcore bits. We observe that, whenever the one-way permutation's hardcore bits are a substring of its input[1] (as is the case for most natural one-way permutations), then the pseudorandom generator construction can be modified so that a substring of the input to the one-way permutation remains fixed in all iterations. This can potentially improve efficiency through precomputation. This general technique already yields most of the efficiency gains in our generator and those of [Gen00, GR00], and also applies to other pseudorandom generators (such as [BBS86]). In particular, it explains how Gennaro's generator is obtained from Patel's and Sundaram's [PS98], providing a simpler proof than the one in [Gen00].

2 An Efficient Pseudorandom Generator

Let:

- $|x|$ denote the length of a string x, and x_i denote the i-th bit of x;
- P, Q of equal length be safe primes ($P = 2P' + 1, Q = 2Q' + 1$, P' and Q' are prime);[2]
- $N = PQ$ and g be a random element of the group QR_N of quadratic residues mod N; $\lceil \log_2 N \rceil = n$;
- $s, \overline{s} \notin QR_N$ be two quadratic non-residues, such that $J_N(s) = 1$ and $J_N(\overline{s}) = -1$;
- $m = m(n) = \lceil n/2 \rceil + O(\log n)$; $c = c(n) = n - m$;
- $p(\cdot)$ be a polynomial.

Construction 1.

```
INPUT y ∈ Z_N
FOR i = 0 TO p(n)
    OUTPUT y_m ... y_1
    LET y ← g^(y_n...y_{m+3}) · s^(y_{m+2}) · s̄^(y_{m+1}) mod N
```

[1] In fact, it is not required that the hardcore bits be precisely a substring; as long as the input can be fully decomposed into two independent parts, one of which is hardcore bits, our approach can be applied. For example, if the hardcore bits are a projection of the input onto a subspace of Z_2^n, and the "non-hardcore" bits are a projection onto the orthogonal subspace, then we are fine.

[2] We can relax the requirements on P and Q and ask only that $(P-1)/2$ and $(Q-1)/2$ be relatively prime; we would then have to appropriately modify our security assumption and to pick g specifically to be a generator of QR_N, because a random g may not be a generator.

On an input $y \in \mathbb{Z}_N$, this generator outputs m pseudorandom bits per one modular exponentiation with a c-bit exponent.

Construction 1 can be optimized, because it uses the same fixed base g in every iteration. Thus, we can precompute some powers of g to speed up exponentiation. Contrary to popular belief, the "naive" precomputation strategy of computing $g, g^2, g^4, g^8, \ldots, g^{2^{c-2}}$ is not very good (it would require roughly $(c-2)/2$ multiplications per iteration). It is better to use the technique of Lim and Lee [LL94]. As a simple example, consider precomputing just three values: g, $g^{2^{(c-2)/2}}$, and their product. Then one can perform the square-and-multiply algorithm "in parallel," looking simultaneously at two positions of the bit string that represents the exponent: i and $i + (c-2)/2$. This would result in an algorithm that takes $(c-2)/2$ squarings and fewer than $(c-2)/2$ multiplications. Generalizing this idea, we precompute the values $g, g^{2^a}, g^{2^{2a}}, g^{2^{3a}}, \ldots, g^{2^{c-2-a}}$ for some a (e.g., $a = (c-2)/2$ or $a = (c-2)/4$), as well as the products of all subsets of these values. Then we can perform the square-and-multiply algorithm "in parallel" by looking at $(c-2)/a$ positions in the exponent at once, thus obtaining an algorithm that takes $(c-2)/a$ squarings and multiplications. In particular, this algorithm would outperform the naive precomputation strategy when $a = (c-2)/4$, while precomputing just 15 values instead of $c-2$. In fact, if one has space to precompute $c-2$ values, then this algorithm will require just $(c-2)/\log(c-2)$ multiplications and squarings. Finally, it should be noted that this technique can also be used to speed up the pseudorandom generators given in [GR00], [HSS93] and [Gen00].

A further small optimization is possible if we choose P to be 3 modulo 8, and Q to be 7 modulo 8 (N is then called a Williams integer). In that case we can fix $s = -1$ and $\overline{s} = 2$, because $-1, 2 \notin QR_N$ and $J_N(-1) = -1$, $J_N(2) = 1$. The last line of the generator then becomes:

LET $y \leftarrow g^{y_n \cdots y_{m+3}} \cdot (-1)^{y_{m+2}} \cdot 2^{y_{m+1}} \bmod N$.

In Section 4, we prove that Construction 1 indeed is a pseudorandom generator, using the techniques from Section 3, under the following assumption:

The first assumption is that products of safe primes are hard to factor:

Assumption 2. *Let N_n denote the set of all n-bit products of equally sized safe primes:*

$$N_n = \{PQ \mid |PQ| = n,\ |P| = |Q|,\ P \textit{ and } Q \textit{ are safe primes}\}.$$

Let A be a probabilistic polynomial-time algorithm. There is no constant $c > 0$ such that for all sufficiently large n:

$$Pr[A(P \cdot Q) = P] > \frac{1}{n^c}$$

where $N = P \cdot Q$ is chosen uniformly from N_n.

To clarify the relation of the previous assumption, and the standard factoring assumption, let us take a look at:

Assumption 3. *For sufficiently large k, a polynomial in k fraction of integers between 2^k and 2^{k+1} are safe primes.*

Clearly, Assumption 2 follows from the standard factoring assumption and Assumption 3. However, in the rest of the paper, we will use Assumption 2 specifically, because it is possible that it holds independently of Assumption 3.

3 General Construction and Proof Technique

In this section, we describe a general technique that we use to prove the pseudorandomness of the generator given in Construction 1. This general technique can also be used to speed up other pseudorandom generators, because it allows one to keep some bits of the seed the same in every iteration, thus permitting precomputation. In particular, we demonstrate that it is possible to keep the bits at hardcore positions intact, throughout the execution of the generator. This technique can be used to prove pseudorandomness of Gennaro's generator from [Gen00] (see Section A).

3.1 Background

We recall without proof two well-known facts (see, e.g. [Gol01]). The first one says that one-way permutations plus hard-core bits give one a PRG with fixed expansion. The second one says that from a PRG with fixed expansion one can obtain a PRG with arbitrary (polynomial) expansion by iterating: dropping some bits of the output and using the remaining ones as a seed for the next iteration.

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation and $H : \{0,1\}^n \to \{0,1\}^m$ be hardcore for f. Let $\circ$ denote concatenation of strings.

Lemma 4. *$G_1 : \{0,1\}^n \to \{0,1\}^{n+m}$:*

$$G_1(x) \stackrel{def}{=} f(x) \circ H(x)$$

is a pseudorandom generator with an n-bit seed x.

Lemma 5. *Let $G_1 : \{0,1\}^n \to \{0,1\}^{n+m}$ (for m polynomial in n). Let $G : \{0,1\}^n \to \{0,1\}^{mp(n)}$, where p is a polynomial, be the following algorithm with an n-bit input x_0:*

`FOR` $i = 0$ `TO` $p(n)$
    `LET` $x_{i+1} \circ h_i \leftarrow G_1(x_i)$`, where` $|x_{i+1}| = n$ `and` $|h_i| = m$
    `OUTPUT` h_i

If G_1 is a pseudorandom generator, then G is a pseudorandom generator.

Traditionally (e.g., [BM84, Yao82]), the Lemmata 4 and 5 are combined in the following well-known construction.

Construction 6.

`FOR` $i = 0$ `TO` $p(n)$
    `LET` $x_{i+1} \leftarrow f(x_i)$
    `OUTPUT` $H(x_i)$

Visually,

$$\begin{array}{lllllll} G(x_0): & x_0 & \xrightarrow{f} & x_1 & \xrightarrow{f} & x_2 & \xrightarrow{f} \cdots \\ & \text{OUTPUT:} & & H(x_0) & & H(x_1) & \cdots \end{array}$$

The following summarizes this section:

Lemma 7. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation and $H : \{0,1\}^n \to \{0,1\}^m$ be hardcore for f. Let $p(\cdot)$ be a polynomial. Then Construction 6 is a pseudorandom generator.*

3.2 New General Constructions

There is no reason that the above two theorems have to be combined specifically as in Construction 6. Below we propose an alternative way to combine them.

Let:

- $[\cdot,\cdot]$ denote some bijective mapping, efficiently computable in both directions (direct and inverse), between pairs of strings h, r from $\{0,1\}^m \times \{0,1\}^{n-m}$ and strings x from $\{0,1\}^n$: $[h,r] \leftrightarrow x$ (such mapping can be as simple as concatenation, but there is no reason to restrict ourselves to it)

- U_S denotes the uniform distribution on set S

- $D_1 \sim D_2$ denote that D_1 is computationally indistinguishable from D_2

- $D_1 \simeq D_2$ denote that D_1 is statistically close to D_2

Suppose the n-bit value x represents the pair of values $[h,r]$, where $|h| = m$ and $|r| = n-m$. Suppose the hard-core function simply selects the first element of this pair: $H(x) = h$. Then f can be viewed as:

$$\begin{array}{|c|c|}\hline h_i & r_i \\ \hline\end{array} \xrightarrow{f} \begin{array}{|c|c|}\hline h_{i+1} & r_{i+1} \\ \hline\end{array}$$

and G_1 provided by Lemma 4 as $G_1(x_i) = G_1([h_i, r_i]) = x_{i+1} \circ h_i = [h_{i+1}, r_{i+1}] \circ h_i$ (i.e., we simply split the $(n+m)$-bit output of G_1 into two strings, x_{i+1} and h_i, and further split x_{i+1} into two strings h_{i+1} and r_{i+1}):

$$\begin{array}{|c|c|}\hline h_i & r_i \\ \hline\end{array} \xrightarrow{G_1} \begin{array}{|c|c|c|}\hline h_{i+1} & r_{i+1} & h_i \\ \hline\end{array}$$

Consider now a function G_1': $G_1'(x_i) = [h_i, r_{i+1}] \circ h_{i+1}$:

$$\begin{array}{|c|c|}\hline h_i & r_i \\ \hline\end{array} \xrightarrow{G_1'} \begin{array}{|c|c|c|}\hline h_i & r_{i+1} & h_{i+1} \\ \hline\end{array}$$

Clearly, since G_1 is a PRG, so is G_1' (because if the output G_1' could be distinguished from uniform, then so could G_1, by applying the appropriate bijections and swapping strings h_i and h_{i+1}).

Applying Lemma 5 to G_1', we get the following PRG G' with an n-bit seed $x_0 = [h_0, r_0]$:

Construction 8.
$$\boxed{\begin{array}{l}\texttt{FOR } i = 0 \texttt{ TO } p(n) \\ \quad \texttt{LET } [h_{i+1}, r_{i+1}] \leftarrow f([h_0, r_i]) \\ \quad \texttt{OUTPUT } h_{i+1}\end{array}}$$

Visually,

$$\begin{array}{llccccccc} G'(x): & [h_0, r_0] & \xrightarrow{G_1'} & [h_0, r_1] & \xrightarrow{G_1'} & [h_0, r_2] & \xrightarrow{G_1'} & \cdots \\ & \text{OUTPUT:} & & h_1 & & h_2 & & \cdots \end{array}$$

The potential benefit of this construction is that part of the input to f (namely, h_0) remains the same for every iteration. For many f (such as the one in the next section) this results in increased efficiency, because some precomputation is possible.

Suppose further (in addition to everything assumed in Construction 8) that there exists an efficient algorithm R, that, given $f([h, r])$ and h, computes $f([0^m, r])$ (for any h, r), and such that $R(U_n \times U_m) = U_n$. Let $G_1''(r) = f([0^m, r])$. Then G_1'' is a PRG (because otherwise $G_1([h, r]) = f([h, r]) \circ h$ could be distinguished from random by computing $R(G_1([h, r])) = G_1''(r)$).

Represent the n-bit output of $G_1''(r_i)$ as $[h_{i+1}, r_{i+1}]$. Then by applying Lemma 5 to G_1', we get the following PRG G'' with an $(n - m)$-bit seed r_0:

Construction 9.

```
FOR i = 0 TO p(n)
    LET [h_{i+1}, r_{i+1}] ← f([0^m, r_i])
    OUTPUT h_{i+1}
```

Visually,

$$
\begin{array}{llllll}
G'(x): & r_0 & \xrightarrow{G_1''} & r_1 \xrightarrow{G_1''} & r_2 \xrightarrow{G_1''} & \cdots \\
 & \text{OUTPUT:} & & h_1 & h_2 & \cdots
\end{array}
$$

This construction has the further benefit that the fixed part of the input to f is always 0^m (rather than a random value h_0). Thus, even more precomputation may be possible (as shown in the next section). Moreover, the seed for this construction is only $n - m$, rather than n, bits long.

The summary of this section is given in the following:

Lemma 10. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation and $H : \{0,1\}^n \to \{0,1\}^m$ be hardcore for f. Let $p(\cdot)$ be a polynomial. Let $[\cdot, \cdot]$ denote some bijective mapping, efficiently computable in both directions (direct and inverse), between pairs of strings h, r from $\{0,1\}^m \times \{0,1\}^{n-m}$ and strings x from $\{0,1\}^n$: $[h, r] \leftrightarrow x$, such that $H([h, r]) = h$. Then Construction 8 is a pseudorandom generator.*

Lemma 11. *Let (in addition to the conditions stated in Lemma 10) there exist an efficient algorithm R, that, given $f([h, r])$ and h, computes $f([0^m, r])$ for any h, r. Let $R(U_n \times U_m) = U_n$. Then, Construction 9 is a pseudorandom generator.*

3.3 Generalizing to other distributions

It is not necessary to restrict Lemmata 4 and 5 only to binary strings. Indeed, we can slightly generalize the notion of a pseudorandom generator to allow domains and ranges other than just $\{0,1\}^k$.

Definition 12. Let D and R be two efficiently samplable sets of binary strings. A deterministic polynomial-time algorithm A is called a $(D \to R)$-*pseudorandom generator* if it holds that $A(U_D)$ is indistinguishable from U_R.

Given this definition, we can reformulate Lemmata 4 and 5 as follows. Let $D, R \subset \{0,1\}^n$ be efficiently samplable and let $f : D \to R$ be a bijection. Let $H : D \to \{0,1\}^m$ be hardcore for f.

Theorem 13. *$G_1(x) \stackrel{def}{=} f(x) \circ H(x)$ is a $(D \to R \times \{0,1\}^m)$-pseudorandom generator.*

Suppose further that f is a permutation of D.

Theorem 14. *If G_1 is a $(D \to D \times \{0,1\}^m)$-pseudorandom generator, then a $(D \to \{0,1\}^{mp(n)})$-pseudorandom generator can be obtained by iterating G_1 like in Lemma 5 (but with $x_i \in D$).*

Construction 8 can also be formulated in terms of more general sets. Namely, suppose that the uniform distribution on D can be decomposed into a direct product of two independent parts: uniform on hardcore bits, and the appropriate distribution on the rest. Then we can swap the hardcore bits as in Construction 8, because both h_i and h_{i+1} are uniform and independent from r_{i+1} anyway. Of course, we do not need true uniformity and independence, but only enough to fool a polynomial-time adversary.

More precisely, let $D \subset \{0,1\}^n$ be an efficiently samplable domain; let G_1 be a $(D \to D \times \{0,1\}^m)$-pseudorandom generator that maps $[h,r] \in D$ to $[h',r'] \circ h$. Let V be the projection of D onto its non-hardcore bits: for $r \in \{0,1\}^{n-m}$, $\Pr[V=r] \stackrel{def}{=} \Pr_{x\in D}[(\exists h)\ x=[h,r]]$.

Lemma 15. *If the uniform distribution U_D is indistinguishable from $[U_{\{0,1\}^m}, V]$, then G_1' is a $([U_{\{0,1\}^m}, V] \to [U_{\{0,1\}^m}, V] \times \{0,1\}^m)$-pseudorandom generator, and can be iterated as in Construction 8 to obtain a $(D \to \{0,1\}^{mp(n)})$-pseudorandom generator.*

Lemmata 10 and 11 (and consequently Constructions 8 and 9) can be reformulated for more general sets D, R, just like Construction 8.

4 Proof of Correctness of Our Generator

We use the idea described in Section 3 to prove that the algorithm given in Construction 1 is a $(\mathbb{Z}_N \to \{0,1\}^{mp(n)})$-pseudorandom generator, under Assumption 2. To be precise, Section 3 was written in terms of one-way functions, while Construction 1 first of all selects a one-way function from a family by picking N and g. Of course, the generic constructions of Section 3 still apply (with appropriate technical modifications), because Lemmata 4 and 5 still hold (see [Gol01]).

4.1 Establishing a single iteration

We start by recalling Theorem 5 of [HSS93]. Let N be a random n-bit Blum integer, and let g be a random quadratic residue in $\mathbb{Z}_N^*$.

Theorem 16 ([HSS93]). *If Assumption 2 holds, then $h : \{0,\ldots,ord_N(g)-1\} \to \{0,1\}^m$, $h(x) = x_m \ldots x_1$ is hardcore for $f : \{0,\ldots,\mathrm{ord}_N(g)-1\} \to QR_N$, where $f(x) = g^x \bmod N$.*[3]

Using Theorem 13, a direct consequence of the previous theorem is:

[3] The usual definitions of one-way function and hardcore function require that their domain be efficiently samplable. However, that is not the case with the domain of f and h, since $ord_N(g) = \phi(N)/4$, which may not be revealed. This will not be a problem, because $\{0,\ldots,N/4\}$ is efficiently samplable, and the uniform distribution on that domain is statistically close to the uniform distribution on $\{0,\ldots,ord_N(g)-1\}$.

Corollary 17. *If Assumption 2 holds, then the distributions* $(g^x \bmod N, x_m \ldots x_1)$ *and* $(g^x \bmod N, r)$*, for uniformly randomly chosen* $x < ord_N(g), r \in \{0,1\}^m$*, are indistinguishable.*

Note that this does not automatically give a pseudorandom generator suitable for iteration, because it is not of the form $(D \to D \times \{0,1\}^m)$. In [HSS93], universal hash functions are used to achieve this.

However, by slightly modifying the function $f(x) = g^x$, it is possible to obtain a simple pseudorandom generator that permits iteration. The intuitive idea is to try to make f a bijection of the entire $\mathbb{Z}_N$. To do so, we first make sure that g generates the entire QR_N by picking N as a product of two safe primes: then a random $g \in QR(N)$ is overwhelmingly likely to generate QR_N.[4] We then observe that QR_N has 4 cosets: $s \cdot QR_N$, $\overline{s} \cdot QR_N$, $s \cdot \overline{s} \cdot QR_N$ and itself, QR_N. So we can add another two bits to the domain of f, and use these bits to pick which coset we map to, thereby covering the whole $\mathbb{Z}_N^*$. These modifications do not reduce the number of hardcore bits from Theorem 16.

The range of f is thus $\mathbb{Z}_N^*$, which is very close to $\mathbb{Z}_N$. The domain is also very close to $\mathbb{Z}_N$: it is $\{0,1,\ldots,\phi(N)/4-1\} \times \{0,1\} \times \{0,1\}$, which is essentially the same as $\mathbb{Z}_N^*$. Thus, even though our modified f is not a permutation of $\mathbb{Z}_N$, it is almost one.

Theorem 18. *Let* $h' : \{0,1,\ldots,\frac{\phi(N)}{4}-1\} \times \{0,1\} \times \{0,1\} \to \{0,1\}^m$ *be defined as:*

$$h'(x, b_1, b_2) = x_m \ldots x_1.$$

Let $f' : \{0,1,\ldots,\frac{\phi(N)}{4}-1\} \times \{0,1\} \times \{0,1\} \to \mathbb{Z}_N^*$ *be as follows:*

$$f'(x, b_1, b_2) = g^x \cdot s^{b_1} \cdot \overline{s}^{b_2}$$

($x \in \{0,1,\ldots,\frac{\phi(N)}{4}-1\}$*,* $b_1, b_2 \in \{0,1\}$*, all arithmetic is in* $\mathbb{Z}_N^*$*).*

If Assumption 2 holds, then $F' = (f', h')$ *is a* $(\{0,1,\ldots,\phi(N)/4-1\} \times \{0,1\} \times \{0,1\} \to \mathbb{Z}_N^* \times \{0,1\}^m)$*-pseudorandom generator.*

Proof. The proof is by a simple reduction to Corollary 17. The details are straightforward and are given in Appendix B. □

Now some care is needed to modify the generator F', so that it may be iterated, since its domain and range are not as required by the constructions of Section 3. What makes iteration possible is that there are efficiently computable "almost bijections" between $\{0,1,\ldots,\frac{\phi(N)}{4}-1\} \times \{0,1\} \times \{0,1\}$ and $\mathbb{Z}_N$, as well as between $\mathbb{Z}_N^*$ and $\mathbb{Z}_N$. Hence, we can use $\mathbb{Z}_N$ as the domain and as the range of f, and that should not make significant difference. The following is merely a formalization of this statement:

[4] For $N = PQ$ with safe P, Q, the subgroup QR_N of quadratic residues in $\mathbb{Z}_N^*$ is of order $\phi(N)/4 = P'Q'$ and is cyclic. QR_N is cyclic, because $QR_N \cong QR_P \times QR_Q$, and QR_P and QR_Q are cyclic of distinct prime orders P' and Q', respectively. Hence, the order of almost any element of QR_N is $P'Q'$, and therefore almost any element of QR_N is a generator (unless it is 1 modulo P or Q). Furthermore, such N are clearly Blum integers and Corollary 17 holds for them if Assumption 3 holds. Finally, as noted in Section 2, we can relax the requirement on P and Q and only ask that $\gcd((P-1)/2, (Q-1)/2) = 1$; then QR_N is still cyclic by the same argument and N is still a Blum integer; however, a random g is not necessarily a generator, and we will simply have to choose g specifically to be a generator.

Corollary 19. *Let*

$$F : \mathbb{Z}_N \to \mathbb{Z}_N \times \{0,1\}^m$$

be as follows:

$$F(x) = (g^{x_n \ldots x_{m+3} x_m \ldots x_1} \cdot s^{x_{m+2}} \cdot \overline{s}^{x_{m+1}}, x_m \ldots x_1)$$

(all arithmetic is in $\mathbb{Z}_N^*$). If Assumption 2 holds, then F is a $(\mathbb{Z}_N \to \mathbb{Z}_N \times \{0,1\}^m)$-pseudorandom generator.

Proof. Let $F'(x, b_1, b_2)$ be as in Theorem 18. Let $D' = \{0, \ldots, \frac{\phi(N)}{4} - 1\}$ and $D = \{0, \ldots, N_n \ldots N_{m+3} \overbrace{1 \ldots 1}^{m}\}$. Also, let $\overline{N} = N_n \ldots N_{m+3} 11 \overbrace{1 \ldots 1}^{m}$.

We will prove the following:

$$F(U_{\mathbb{Z}_N}) \simeq F(U_{\mathbb{Z}_{\overline{N}}}) = F'(U_D \times U_{\{0,1\}} \times U_{\{0,1\}}) \simeq F'(U_{D'} \times U_{\{0,1\}} \times U_{\{0,1\}}) \sim$$

$$\sim U_{\mathbb{Z}_N^*} \times \{0,1\}^m \simeq U_{\mathbb{Z}_N} \times \{0,1\}^m.$$

- $F(U_{\mathbb{Z}_N}) \simeq F(U_{\mathbb{Z}_{\overline{N}}})$ because $U_{\mathbb{Z}_{\overline{N}}} \simeq U_{\mathbb{Z}_N}$
- $F(U_{\mathbb{Z}_{\overline{N}}}) = F'(U_D \times U_{\{0,1\}} \times U_{\{0,1\}})$, because $F'(x, b_1, b_2) = F(x_{n-2} \ldots x_{m+1} b_1 b_2 x_m \ldots x_1)$
- $F'(U_D \times U_{\{0,1\}} \times U_{\{0,1\}}) \simeq F'(U_{D'} \times U_{\{0,1\}} \times U_{\{0,1\}})$ because $U_D \simeq U_{D'}$ (since $|D| = \frac{N}{4} + O(\sqrt{N})$)
- $F'(U_{D'} \times U_{\{0,1\}} \times U_{\{0,1\}}) \sim U_{\mathbb{Z}_N^*} \times \{0,1\}^m$ by Theorem 18
- $U_{\mathbb{Z}_N^*} \times \{0,1\}^m \simeq U_{\mathbb{Z}_N} \times \{0,1\}^m$ because $U_{\mathbb{Z}_N^*} \simeq U_{\mathbb{Z}_N}$.

□

4.2 Iterating

By Theorem 14, F can be iterated to yield polynomial expansion: in each iteration, we output $h(x)$, and use $g^{x_n \cdots x_3} \cdot s^{x_2} \cdot \overline{s}^{x_1}$ as the next input for F. Better yet we show that Construction 9 can be used to give a more efficient generator with polynomial expansion. There are two conditions that need to be satisfied for that: there has to be a pairing $[\cdot, \cdot]$ with properties described in Lemma 15, and F has to exhibit the self-reducibility property described in Construction 9.

In the following simple lemma, we define a pairing $[\cdot, \cdot]$ such that $[h, r] \mapsto h$ is hardcore for F, and demonstrate that the condition stated in Lemma 15 holds for $[\cdot, \cdot]$.

Lemma 20. *Let U_m denote the uniform distribution on $\{0,1\}^m$. Let V be the projection of $U_{\mathbb{Z}_N}$ onto the non-hardcore bits. Let $[\cdot, \cdot] : \{0,1\}^m \times \{0,1\}^c \leftrightarrow \{0,1\}^n$ such that $[h, r] = r_c \ldots r_1 h_m \ldots h_1$.*

Then, $[U_m, V] \sim U_{\mathbb{Z}_N}$.

Proof. We prove that $[U_m, V] \simeq U_{\mathbb{Z}_N}$ via a straightforward statistical distance argument. See Appendix B. □

Next, we show that F is self-reducible in the sense of Construction 9. To do that, we only need to provide the reduction algorithm R. We define it simply as $R(y, h) = y \cdot g^{-h}$. Clearly, R is efficiently computable, $R(U_{\mathbb{Z}_N} \times U_m) = U_{\mathbb{Z}_N}$, and a simple calculation verifies that $R(F''([h, r]), h) = F''([0, r])$, where $F'' : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$, $F''(x) = g^{x_n \cdots x_3} \cdot s^{x_2} \cdot \overline{s}^{x_1}$ (F'' is the one-way function that we use to obtain F).

Based on the observation about self-reducibility and Lemma 20, we can apply Construction 9 to F, thus obtaining the following pseudorandom generator:

Construction 21.

```
INPUT y ∈ Z_N
FOR i = 0 TO p(n)
    LET e ← y_n ... y_{m+3} 0...0   (m zeros)
    LET y ← g^e · s^{y_{m+2}} · s̄^{y_{m+1}}
    OUTPUT y_m ... y_1
```

$$\texttt{LET } e \leftarrow y_n \ldots y_{m+3} \overbrace{0 \ldots 0}^{m}$$

The output distribution of this generator is the same as that of Construction 1. That is because, in Construction 21, $g^e = (g^{2^m})^{y_n \cdots y_{m+3}}$, but g and g^{2^m} are distributed identically because $g \mapsto g^{2^m}$ is a permutation over QR_N, so instead of g^e, we can write $g^{y_n \cdots y_{m+3}}$.

Acknowledgment

We are grateful to Igor Shparlinski for suggesting a relaxation of the constraints on the modulus N.

References

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, April 1988.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–863, November 1984.

[Gen00] Rosario 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*, pages 469–481. Springer-Verlag, 20–24 August 2000.

[GMT82] Shafi Goldwasser, Silvio Micali, and Po Tong. Why and how to establish a private code on a public network. In *33rd Annual Symposium on Foundations* *of Computer Science*, pages 134–144, Pittsburgh, Pennsylvania, 24–27 October 1982. IEEE.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[GR00] Oded Goldreich and Vered Rosen. On the security of modular exponentiation with application to the construction of pseudorandom generators. Technical Report 2000/064, Cryptology e-print archive, `http://eprint.iacr.org`, 2000. Prior version appears in [Ros01].

[HSS93] J. Håstad, A. W. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides $O(n)$ bits. *Journal of Computer and System Sciences*, 47:376–404, 1993.

[LL94] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer-Verlag, 21–25 August 1994.

[MS91] Silvio Micali and Claus P. Schnorr. Efficient, perfect polynomial random number generator. *Journal of Cryptology*, 3(3):157–172, 1991.

[PS98] S. Patel and G. Sundaram. An efficient discrete log pseudo random generator. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 304–317. Springer-Verlag, 23–27 August 1998.

[Ros01] Vered Rosen. On the security of modular exponentiation with application to the construction of pseudorandom generators. Technical Report TR01-007, ECCC (*Electronic Colloquium on Computational Complexity*, `http://www.eccc.uni-trier.de/eccc`), 2001.

[VV84] Umesh V. Vazirani and Vijay V. Vazirani. Efficient and secure pseudo-random number generation. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 193–202. Springer-Verlag, 1985, 19–22 August 1984.

[Yao82] A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

A A Simple Proof of Gennaro's Generator

The technique of Section 3 can also be used to simplify the proof of pseudorandomness of the generator given in [Gen00]:

Let p be an n-bit safe prime, g a generator of $\mathbb{Z}_p^*$, and $c = \omega(\log n)$.

```
INPUT y ∈ Z_p
LET γ ← g^{2^{n-c+2}}
FOR i = 0 TO p(n)
    LET x ← y
    LET y ← γ^{x_n...x_{n-c+2}} · g^{x_1}      (= g^{x_n...x_{n-c+2} 0...0 x_1})
    OUTPUT y_{n-c+1} ... y_2
```

$$\begin{aligned}&\texttt{INPUT } y \in \mathbb{Z}_p\\ &\texttt{LET } \gamma \leftarrow g^{2^{n-c+2}}\\ &\texttt{FOR } i = 0 \texttt{ TO } p(n)\\ &\quad \texttt{LET } x \leftarrow y\\ &\quad \texttt{LET } y \leftarrow \gamma^{x_n \dots x_{n-c+2}} \cdot g^{x_1} \qquad (= g^{x_n \dots x_{n-c+2} \overbrace{0 \dots 0}^{n-c} x_1})\\ &\quad \texttt{OUTPUT } y_{n-c+1} \dots y_2\end{aligned}$$

Informally, this generator is based on the assumption that, for n, p and g as above, exponentiation $x \mapsto g^x \bmod p$ is one-way even if we restrict its domain only to short $\omega(\log n)$-bit exponents.

In [PS98], it is shown that, if the above assumption holds (also known as *Discrete Logarithm with Short Exponent*, or *DLSE*), then the full-length-exponent exponentiation in $\mathbb{Z}_p^*$ has $n - \omega(\log n)$ hardcore bits. More precisely:

Theorem 22. *Let p be an n-bit safe prime and let g be a generator of $\mathbb{Z}_p^*$. Define $f : Z_{p-1} \rightarrow Z_p^*$ as $f(x) = g^x \bmod p$. Fix some $c = \omega(\log n)$. If* DLSE *holds, then $H(x) = x_{n-c} \dots x_2$ is hardcore for f.*

The proof of this theorem can be found in [PS98].

The proof of pseudorandomness of the above generator is similar to the proof in Section 4. First, similarly to Corollary 19, it can be shown that extending both the domain and the range of f to $\mathbb{Z}_p$ does not hurt pseudorandomness. More precisely, for $f' : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ such that $f'(x) = g^x \bmod p$, it holds that $x \mapsto (f'(x), H(x))$ is a $(\mathbb{Z}_p \rightarrow \mathbb{Z}_p \times \{0,1\}^m)$-pseudorandom generator[5] ($m = n - c - 1$).

By applying Construction 9 to f', we obtain exactly the generator given above. So we only need to demonstrate that the two conditions for Construction 9 are fulfilled.

Let us first define the pairing described in Lemma 15. Let $[\cdot,\cdot] : \{0,1\}^m \times \{0,1\}^{n-m} \leftrightarrow \{0,1\}^n$ such that $[h, r] = r_{n-m} \dots r_2 h_m \dots h_1 r_1$. Obviously, $[h, r] \mapsto h$ is hardcore for f (it is the same as H). Let V be the projection of $U_{\mathbb{Z}_p}$ onto bits that are not hardcore for f (1, $m{+}2$, $\dots$, n). Then, an argument similar to the one in Lemma 20 shows that $[U_{\{0,1\}^m}, V] \sim \mathbb{Z}_p$

We still have to give the reduction algorithm R required for Construction 9. But, $R(y, h)$ is simply $y \cdot g^{-2h}$. It is easy to check that $R([h, r], h) = [h, 0]$ and that $R(U_{\mathbb{Z}_p} \times U_m) = U_{\mathbb{Z}_p}$.

B Proofs

Proof of Theorem 18 Consider the function $f : \{0, 1, \dots, \frac{\phi(N)}{4} - 1\} \rightarrow \mathbb{Z}_N^*$, $f(x) = g^x$. If Assumption 2 holds, then by Corollary 17, $x \mapsto (g^x, x_m \dots x_1)$ is a $(\{0, 1, \dots, \frac{\phi(N)}{4} - 1\} \rightarrow QR_N \times \{0,1\}^m)$-pseudorandom generator. This is because $ord_N(g) = \frac{\phi(N)}{4}$.

Suppose that a PPT A distinguishes, with non-negligible probability, $F'(U_{\{0,1,\dots,\frac{\phi(N)}{4}-1\}} \times U_{\{0,1\}} \times U_{\{0,1\}})$ against $U_{\mathbb{Z}_N^*} \times \{0,1\}^m$. Then, we can build a PPT B that distinguishes, with

[5]More precisely, the output of this generator is indistinguishable from $H(U_{\mathbb{Z}_p^*})$. But, it is not hard to see that $H(U_{\mathbb{Z}_p^*}) \simeq U_{\{0,1\}^m}$.

non-negligible probability, $(g^x, x_m \ldots x_1)$ against (g^x, r) ($r \in_R \{0,1\}^m$, $x \in_R \{0, \ldots, \frac{\phi(N)}{4} - 1\}$): on input $(y,h) \in \mathbb{Z}_N^* \times \{0,1\}^m$, B generates random $b_1 \in_R \{0,1\}$ and $b_2 \in_R \{0,1\}$ and outputs $A(y \cdot s^{b_1} \cdot \overline{s}^{b_2}, h)$. Clearly, if (y,h) is distributed according to $U_{QR_N} \times U_{\{0,1\}^m}$, then $(y \cdot s^{b_1} \cdot \overline{s}^{b_2}, h)$ is distributed according to $U_{\mathbb{Z}_N^*} \times U_{\{0,1\}^m}$. Similarly, if (y,h) is distributed according to $(g^x, x_m \ldots x_1)$ (x is a random variable on $\{0, \ldots, \frac{\phi(N)}{4} - 1\}$), then $(y \cdot s^{b_1} \cdot \overline{s}^{b_2}, h)$ is distributed according to $F'(U_{\{0,1,\ldots,\frac{\phi(N)}{4}-1\}} \times U_{\{0,1\}} \times U_{\{0,1\}})$. $\square$

Proof of Lemma 20 Let $M = N_n \ldots N_{m+1}$. Obviously, $\Pr_{r \in V}\Big[r = s\Big] = \frac{2^m}{N}$, for $s < M$ and $|s| = c$.

Let $M' = N_n \ldots N_{m+1} \underbrace{0 \ldots 0}_{m}$. Then, $\Pr_{r \in V}\Big[r = M\Big] = \frac{N - M'}{N}$.

So, $\Pr_{h \in U_m,\ r \in V}\Big[[h,r] = j\Big] = \frac{1}{N}$, for $j < M'$. Also, $\Pr_{h \in U_m,\ r \in V}\Big[[h,r] = j\Big] = \frac{N - M'}{N2^m}$, for $j \geq M'$.

Finally, the statistical distance between $U_{\mathbb{Z}_N}$ and $[U_m, V]$ is

$$\sum_{j=0}^{M'+2^m-1} \left| \Pr_{x \in U_{\mathbb{Z}_N}}\Big[x = j\Big] - \Pr_{h \in U_m,\ r \in V}\Big[[h,r] = j\Big] \right| =$$

$$= \sum_{j=M'}^{M'+2^m-1} \left| \Pr_{x \in U_{\mathbb{Z}_N}}\Big[x = j\Big] - \Pr_{h \in U_m,\ r \in V}\Big[[h,r] = j\Big] \right| \leq$$

$$\leq \sum_{j=M'}^{M'+2^m-1} \left| \Pr_{x \in U_{\mathbb{Z}_N}}\Big[x = j\Big] \right| + \sum_{j=M'}^{M'+2^m-1} \left| \Pr_{h \in U_m,\ r \in V}\Big[[h,r] = j\Big] \right| \leq$$

$$\leq \frac{2^m}{N} + \frac{2^m(N - M')}{N2^m} =$$

$$= \frac{2^m + N - M'}{N} = negl(n).$$

$\square$