Pseudo-Random Functions and Factoring[*]

Moni Naor[†]
Department of Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel.
naor@wisdom.weizmann.ac.il

Omer Reingold[‡]
AT&T Labs - Research.
180 Park Avenue, Bldg. 103,
Florham Park, NJ, 07932, USA.
omer@research.att.com

Alon Rosen
Department of Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel.
alon@wisdom.weizmann.ac.il

May 9, 2001

Abstract

Factoring integers is the most established problem on which cryptographic primitives are based. This work presents an efficient construction of *pseudorandom functions* whose security is based on the intractability of factoring. In particular, we are able to construct efficient length-preserving pseudorandom functions where each evaluation requires only a *constant* number of modular multiplications per output bit. This is substantially more efficient than any previous construction of pseudorandom functions based on factoring, and matches (up to a constant factor) the efficiency of the best known factoring-based *pseudorandom bit generators*.

[*] An extended abstract appeared in the thirty second annual ACM Symposium on the Theory of Computing, 2000.

[†] Research Supported by a grant from the Israel Science Foundation adminstered by the Israel Academy of Sciences.

[‡] Research performed while still at the Weizmann Institute, Rehovot, Israel. Research supported by an Eshkol Fellowship of the Israeli Ministry of Science.

1 Introduction

Almost any interesting cryptographic task must be based on the computational hardness of some problem. Proving such hardness assumptions exceeds by far the the state of the art of Complexity Theory. It is therefore desirable to base the security of a cryptographic construction on as reasonable assumption as possible. A natural approach is to rely on a well studied problem where many algorithms have been tried and their complexity is well understood. The most established candidate in these respects, and certainly the one with the best pedigree, is the problem of factoring integers (see [21] for the state of the art of factoring).

The focus of this paper is an efficient construction of *pseudorandom functions* (see definition below) whose security is based on the intractability of factoring. In particular, we are able to construct efficient length-preserving pseudorandom functions whose evaluation requires only a constant number of modular multiplications per output bit. This is substantially more efficient than any previous construction of pseudorandom functions based on factoring, and matches (up to a constant factor) the efficiency of the best known factoring-based *pseudorandom bit generators.*

Pseudorandom functions[1], originally defined by Goldreich, Goldwasser and Micali [13] are an important cryptographic primitive. A distribution of functions is pseudorandom if it satisfies the following requirements:

Easy to sample: It is easy to sample a function according to the distribution.

Easy to compute: Given such a function it is easy to evaluate it at any given point.

Pseudorandom: It is hard to tell apart a function sampled according to the pseudorandom distribution from a uniformly distributed function when the distinguisher is given access to the function as a black-box.

Pseudorandom functions have a wide range of applications, most notably in cryptography, but also in computational complexity and computational learning theory. Coming up with efficient constructions for such functions is a challenge of great practical and theoretical interest.

The new construction improves the one by Naor and Reingold [19], who showed how to construct pseudorandom functions based on factoring, where the cost of evaluation is comparable to two modular exponentiations. The drawback of those functions is that the output is only a single bit. In order to apply them for achieving a length preserving pseudorandom function one would need to repeat the process n times, rendering it inefficient. The improvement we propose lies in a method to expand the one bit output of the NR functions to polynomially many bits while paying only a small overhead in the complexity of the evaluation (i.e. one modular multiplication for each additional output bit). This improvement will be achieved through a surprising combination of the NR functions and the Blum-Blum-Shub pseudorandom generator [5]. As will be demonstrated in the sequel, in general such a composition does *not* necessarily yield a pseudorandom function. This in particular implies a non-straightforward proof of security.

The method we suggest enables us to construct efficient length preserving pseudorandom functions which are at least as secure as factoring Blum-integers and can be evaluated at the cost of fewer than three modular exponentiations. This is comparable to another attractive construction by NR [19], of pseudorandom functions which are at least as secure as the *Decisional Diffie-Hellman (DDH)* problem. While the DDH problem has received much attention recently (see [4]), it is not nearly as well established as factoring.

[1] Note the difference between a pseudorandom function and a bit generator - the latter expands a random seed to some fixed length sequence that should be indistinguishable from a random sequence of similar length; there is no "probing" in the attack.

Organization: The next section contains the background material and our construction. The main result of our work, the efficient construction of a pseudorandom function which is at least as secure as factoring Blum-integers, is presented in Section 4. The proof of security is given in Section 5.

2 Old and New Constructions

2.1 Background

When Goldreich, Goldwasser and Micali originally defined pseudorandom functions [13] they were at least partly motivated by the construction of the Blum-Blum-Shub (BBS) pseudorandom *generator* [5] and, in particular, by an open question suggested there — the *easy-access* problem[2]. Nevertheless, the actual GGM construction of pseudorandom functions did not appear to have any direct connection to the BBS generator (apart from the fact that the BBS generator can be used as a building block for the GGM construction). The current work suggests a construction which is directly related to the BBS generator. We now turn to survey previous constructions of pseudorandom functions (as well as the BBS generator).

2.1.1 The Blum-Blum-Shub Generator

The Blum-Micali Paradigm: Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be a one-way permutation (i.e. one where it is easy to compute $f(x)$ but intractable to find $x = f^{-1}(y)$), and let $\mathcal{B}(\cdot)$ be a a hard-core predicate for $f(\cdot)$ (i.e. given y it is difficult to guess $\mathcal{B}(f^{-1}(y))$). Blum and Micali proposed the following scheme to construct a generator stretching an n-bit seed, x, to an ℓ-bit pseudorandom string:

$$\mathcal{B}(x), \mathcal{B}(f(x)), \ldots, \mathcal{B}(f^{(k)}(x)), \ldots, \mathcal{B}(f^{(\ell(n)-1)}(x)) \tag{1}$$

As a candidate one-way permutation (based on factoring) BBS [5] considered the squaring function modulo an integer $N = P \cdot Q$ (i.e. the mapping $x \mapsto x^2 \bmod N$)[3]. Following [5], the values of N are restricted to integers of the form $N = P \cdot Q$ where P and Q are two distinct primes both congruent to 3 mod 4 (such integers are known as Blum-integers).

Blum-integers: Restricting N to be a Blum-integer enabled BBS to prove that the squaring function is indeed a permutation (when its domain is limited to the subgroup of *quadratic residues* in $\mathbb{Z}_N^*$). Let $N = P \cdot Q$ be an integer, an element in $\mathbb{Z}_N^*$ is called a quadratic residue if it has a square root, namely there is a $y \in \mathbb{Z}_N^*$ such that $y^2 = x \bmod N$. It is easy to verify that the set of quadratic residues in $\mathbb{Z}_N^*$ forms a subgroup (which we denote by Q_N). We note that every $x^2 \in Q_N$ has exactly four distinct square roots, $\pm x, \pm y \in \mathbb{Z}_N^*$, and in the special case that N is a Blum-integer it is possible to prove [5] that exactly one of these square roots resides in Q_N (which implies that squaring is indeed a permutation over Q_N).

Constructing the BBS Generator: The BBS pseudorandom generator is obtained by applying the Blum–Micali paradigm to the squaring permutation together with the $\mathcal{LSB}(\cdot)$ (least significant

[2] The *easy-access* problem arises when one notices that it is easy to access exponentially far away bits in the BBS pseudorandom pad. The question is whether the BBS pad remains pseudorandom even when the distinguisher has access to these exponentially far away bits.

[3] It was shown by Rabin [22] that the problem of factoring an integer $N = P \cdot Q$ can be reduced to the problem of extracting square roots in $\mathbb{Z}_N^*$. Thus, if factoring $N = P \cdot Q$ is hard, then squaring is indeed one-way.

bit) hard-core predicate. This generator has been originally proven secure assuming intractability of Quadratic Residuosity Problem in [5], and subsequently under the assumption that factoring Blum-integers is hard (Assumption 4.1) in [24] (by adapting the techniques in [1]). Note also that it is the basis for the Blum-Goldwasser public-key encryption scheme [6]. For simplicity of exposition , we choose to replace the $\mathcal{LSB}(\cdot)$ hard-core predicate with the Goldreich-Levin $\mathcal{B}_r(\cdot)$ predicate [14] (where $\mathcal{B}_r(m)$ denotes the inner product, $\langle m, r\rangle \bmod 2$). We obtain a generator which stretches an n-bit seed, $x \in Q_N$, to an ℓ-bit pseudorandom string (and is completely analogous to the BBS generator):[4]

$$\mathcal{B}_r(x), \mathcal{B}_r(x^2), \ldots, \mathcal{B}_r(x^{2^k}), \ldots, \mathcal{B}_r(x^{2^{\ell(n)-1}}) \tag{2}$$

The BBS generator is considered efficient (relative to other generators based on factoring), each bit in its output can be obtained at the cost of one modular multiplication. In particular, by performing $2n$ modular multiplications it is possible to stretch an n-bit seed to a $2n$-bit pseudorandom string.

The Easy-Access Problem: In the BBS generator a seed (x, N) defines an infinite (ultimately periodic) bit-sequence $b_0, b_1, \ldots$ (even though a pseudorandom string generated with an n-bit long seed consists of only polynomially many (in n) bits). An interesting feature of the BBS generator is that knowledge of the factorization of N allows *easy access* to each of the first 2^n bits; that is, if $\log i < n$, the i^{th} bit, b_i, can be computed in poly(n) time (by first computing $\beta_i = 2^{i-1} \bmod \varphi(N)$ and then setting $b_i = \mathcal{B}_r(x^{\beta_i})$). However, as GGM noted [13], this easily accessible exponentially long bit-string *may not* appear "random". What BBS have proved, is that any single polynomially long interval of *consecutive* bits in the string is pseudorandom (provided that factoring Blum-integers is hard). Indeed, it might be the case that, say, given $b_1, \ldots, b_n$ and $b_{2\sqrt{n}+1}, \ldots, b_{2\sqrt{n}+n}$, it is easy to compute any other bit in the string.

The *easy-access* open problem is whether direct access to exponentially far away bits in the BBS bit-sequence is an operation which preserves pseudorandomness. This problem was discussed in [2, 8, 5, 13].

2.1.2 The GGM Construction

Motivated in part by the easy-access problem, Goldreich, Goldwasser and Micali [13] introduced the notion of pseudorandom functions and provided a generic construction based on any length doubling pseudorandom generator. Note that a pseudorandom function may be viewed as an exponentially long bit-string which remains pseudorandom even after its bits are accessed in a direct manner. Thus, in some sense, GGM have solved the easy access problem.[5]

When applied to an efficient pseudorandom generator based on factoring (e.g. the BBS generator), the GGM construction yields a length-preserving pseudorandom function which is as secure as factoring, but requires as much as $O(n^2)$ modular multiplications per evaluation. On the other hand, a positive answer to the easy-access problem would have implied that the function:

$$f^{BBS}_{N,g,r}(i) = G^{BBS}_{N,r,n}(g^{2^{i \cdot n}}) \tag{3}$$

is a length preserving pseudorandom function which is at least as secure as factoring and requires only $O(n)$ modular multiplications per evaluation. Thus, in some sense, the question of whether it

[4] We denote by n the size (in bits) of N, and by x^{2^j} the value of $x^{2^j} \bmod N$.

[5] What GGM have actually demonstrated is how to construct exponentially long, easily accessible, pseudorandom strings based on *any* one-way function (following [15]). However, this does not imply that the *specific* BBS bit-sequence remains pseudorandom given direct access to exponentially many of its bits.

is possible to construct such efficient pseudorandom functions based on factoring (which require as much modular multiplications as $f^{BBS}_{N,g,r}$), remained open.

2.1.3 The NR Constructions

About a decade after the GGM paper appeared, Naor and Reingold (NR) [18] suggested a parallel construction of pseudorandom functions. The NR construction was obtained by introducing a new cryptographic primitive, the *pseudorandom synthesizer.* By applying their method to specific constructions of pseudorandom synthesizers, they were later able to present efficient pseudorandom functions based on standard number-theoretic assumptions [19]. These constructions are considerably more efficient than the constructions which would have been obtained by applying the generic GGM construction to specific pseudorandom generators that are based on the same assumptions.

The DDH Construction: A construction of length preserving pseudorandom functions as secure as the *Decisional Diffie-Hellman (DDH)* problem which require roughly $2n$ modular multiplications per evaluation [19]. This already matches the efficiency offered by $f^{BBS}_{N,g,r}$ and, to the best of our knowledge, is the most efficient construction of pseudorandom functions to date (based on standard intractability assumptions).

The Factoring Construction: A construction of pseudorandom functions at least as secure as factoring, which require roughly $2n$ modular multiplications per evaluation [19] (their proof of security utilizes Biham, Boneh and Reingold's [3] result that breaking the Generalized Diffie-Hellman assumption over composites implies an efficient algorithm for factoring.)

For every $n \in \mathbb{N}$, a key of a function in the NR pseudorandom function ensemble, F_n, is a tuple $\langle N, \vec{a}, g, r\rangle$, where N is an n-bit Blum-integer, $\vec{a} = \langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1}\rangle$ is a sequence of $2n$ elements in $[N]$, g is a quadratic-residue in $\mathbb{Z}^*_N$ and r is an n-bit string. For any n bit input $x = x_1 x_2 \ldots x_n$, the NR function (with a single bit of output) is defined as:

$$f_{N,\vec{a},g,r}(x) = \mathcal{B}_r(g^{\Pi_{i=1}^{n} a_{i,x_i}}) \tag{4}$$

The NR construction gives a pseudorandom function which *seems* to be as efficient as $f^{BBS}_{N,g,r}$. Note however, that the NR function has only one bit of output, whereas $f^{BBS}_{N,g,r}$ has linear output length. While this may be sufficient for some applications, in most scenarios it is not. The goal of our work is to match the result one would have obtained by proving that $f^{BBS}_{N,g,r}$ are indeed pseudorandom functions. That is, we provide a new construction of pseudorandom functions that: (1) are at least as secure as factoring Blum-integers, (2) have linear output length, and (3) require only $O(n)$ modular multiplications per evaluation.

2.2 Our Construction

The NR pseudorandom function, $f_{N,\vec{a},g,r}$, is obtained by extracting the $\mathcal{B}_r(\cdot)$ predicate from the value of the function:

$$h_{N,\vec{a},g}(x) = g^{\Pi_{i=1}^{n} a_{i,x_i}} \tag{5}$$

It turns out that, even though it is not pseudorandom in itself, the function $h_{N,\vec{a},g}$ is unpredictable in some weak sense. Assuming the intractability of factoring Blum-integers, Naor and Reingold have shown [19, 20] that $h_{N,\vec{a},g}$ is unpredictable against an *adaptive* sample and a *random* challenge. That is, for a random $x \in \{0,1\}^n$, no polynomial-time adversary is able to predict the value of $h(x)$ after adaptively querying the value of $h(y)$ for polynomially many $y \neq x$ of his choice.

The main idea behind our construction is using the value of $h_{N,\vec{a},g}$ as a seed to the BBS pseudorandom generator. At first glance, it is not clear why this method should work *at all*. Indeed, applying a pseudorandom generator to an "unpredictable" value does *not* necessarily yield a pseudorandom function (see section 5.1 for a more detailed discussion on the subject). The reason for which our construction *does* work lies in the *specific* number theoretic features which the function $h_{N,\vec{a},g}$ and the BBS generator have in common.

This enables us to expand the output length of the NR function to polynomially many bits while paying a "reasonable" overhead in the complexity of the evaluation (i.e. one modular multiplication for each additional output bit). Specifically, let $N, \vec{a}, g$ and r be defined as in the NR function, the function we propose is defined as:

$$f_{N,\vec{a},g,r}(x) = G^{BBS}_{N,r,\ell}(g^{\Pi_{i=1}^{n} a_{i,x_i}}) \tag{6}$$

Even though this does not solve the particular easy-access problem, it *does* match the efficiency one would have obtained by proving that $f^{BBS}_{N,g,r}$ are indeed pseudorandom functions (as well as the efficiency of the DDH-based pseudorandom functions by NR [19]). By taking $\ell(n) = n$, we obtain a length preserving pseudorandom function which is at least as secure as factoring, has linear output length, and requires only $3n$ modular multiplications per evaluation. This already matches (up to a constant factor) the efficiency of the best known factoring-based pseudorandom *generators* (which also require $O(n)$ multiplications per evaluation) and certainly improves the efficiency of the GGM pseudorandom functions which use BBS as a building block.

3 Preliminaries

For the sake of completeness we present the formal definition of pseudorandom functions. Our exposition follows the ones appearing in [11, 12, 18].

3.1 Pseudorandom Functions - definition

Pseudorandom functions were defined by Goldreich, Goldwasser and Micali [13]. Loosely speaking, these are efficient distributions of functions that cannot be efficiently distinguished from the uniform distribution. That is, an efficient algorithm that gets a function as a black box cannot tell (with non-negligible advantage) from which of the distributions it was sampled.[6] To formalize the notion of pseudorandom functions, we will need to consider ensembles of functions.

Definition 3.1 *Let ℓ_d and ℓ_r be any two integer functions. An $I^{\ell_d} \rightarrow I^{\ell_r}$* function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbb{N}}$ of random variables, such that the random variable F_n assumes values in the set of $I^{\ell_d(n)} \rightarrow I^{\ell_r(n)}$ functions. The* uniform $I^{\ell_d} \rightarrow I^{\ell_r}$ function ensemble*, $R = \{R_n\}_{n\in\mathbb{N}}$, has R_n uniformly distributed over the set of $I^{\ell_d(n)} \rightarrow I^{\ell_r(n)}$ functions.*

An explicit description of a function $f : I^{\ell_d} \rightarrow I^{\ell_r}$ requires as much as $2^{\ell_r 2^{\ell_d}}$ bits. This suggests an alternative view of pseudorandom functions: These are distributions of exponentially long bit-sequences that cannot be distinguished from random by an efficient algorithm which has direct access to the sequence. To be of practical value however, we require that pseudorandom functions can be efficiently sampled and computed. This property is not satisfied by every function ensemble (e.g. the uniform function ensemble: It contains $2^{\ell_r 2^{\ell_d}}$ functions whose mere representation requires as much as $\ell_r 2^{\ell_d}$ bits), we therefore restrict ourselves to efficiently computable function ensembles.

[6] For a detailed exposition on pseudorandom functions and their applications we refer the reader to [23].

Definition 3.2 *A function ensemble, $F = \{F_n\}_{n \in \mathbb{N}}$, is* efficiently computable *if there exist probabilistic polynomial-time algorithms, $\mathcal{I}$ and $\mathcal{V}$, and a mapping from strings to functions, ϕ, such that $\phi(\mathcal{I}(1^n))$ and F_n are identically distributed and $\mathcal{V}(i, x) = (\phi(i))(x)$.*

We denote by f_i the function assigned to i (i.e. $f_i \overset{\text{def}}{=} \phi(i)$). We refer to i as the key of f_i and to $\mathcal{I}$ as the key-generating algorithm of F.

In particular, functions in efficiently computable function ensembles have relatively succinct representation (i.e. of polynomial rather than exponential length). As a consequence, these ensembles may have only exponentially many functions (out of double-exponentially many possible functions).

The distinguisher, in our setting, is defined to be an oracle machine that can make queries to a function (which is either sampled from the pseudorandom function ensemble[7] or from the uniform function ensemble[8]). We assume that on input 1^n the oracle machine makes only n-bit queries. For any probabilistic oracle machine, $\mathcal{M}$, and any $I^n \rightarrow I^{\ell(n)}$ function, O, we denote by $\mathcal{M}^O(1^n)$ the distribution of $\mathcal{M}$'s output on input 1^n and with access to O.

Definition 3.3 *An efficiently computable $I^n \rightarrow I^{\ell(n)}$ function ensemble, $F = \{F_n\}_{n \in \mathbb{N}}$, is* pseudorandom *if for every probabilistic polynomial-time oracle machine $\mathcal{M}$, every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\left| \Pr\left[\mathcal{M}^{F_n}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{R_{n,\ell}}(1^n) = 1\right] \right| < \frac{1}{p(n)}$$

where $R = \{R_{n,\ell}\}_{n \in \mathbb{N}}$ is the uniform $I^n \rightarrow I^{\ell(n)}$ function ensemble.

The term "pseudorandom functions" is hereafter used as an abbreviation for "efficiently computable pseudorandom function ensemble".

3.2 Notation

- $\mathbb{N}$ denotes the set of all natural numbers.
- For any integer $k \in \mathbb{N}$, denote by $[k]$ the set of integers $\{0, 1, \ldots, k-1\}$.
- For any integer $N \in \mathbb{N}$ the multiplicative group modulo N is denoted by $\mathbb{Z}_N^*$.
- The order of $\mathbb{Z}_N^*$ (i.e. the number of $x \in [N]$ such that $\gcd(x, N) = 1$) is denoted by $\varphi(N)$.
- I^n denotes the set of all n-bit strings, $\{0,1\}^n$.
- U_n denotes the random variable uniformly distributed over I^n.
- Let x and y be any two bit strings then x, y denotes the string x concatenated with y.

[7] We stress that in the case that the function is sampled from the pseudorandom function ensemble the distinguisher is *not* given the representation of the function f_i (i.e. the key i).

[8] As we have mentioned, it is not clear even how to efficiently represent a uniformly distributed function (as the representation it is too large to store). Still, one may simulate such a function by answering given queries with independently and uniformly chosen answers (while memorizing previous answers for the sake of consistency).

4 The Main Result

We are now ready to present the main result of our work, an efficient construction of pseudorandom functions whose security is based on the intractability of factoring. Specifically, we are able to show how any procedure which is able to distinguish our functions from randomly chosen ones can be turned into an algorithm which factors a non-negligible fraction of Blum-Integers. We begin by formalizing the assumption that factoring Blum-integers is hard.

4.1 The Factoring Assumption

In order to keep our result general, we let N be generated by *some* polynomial-time algorithm FIG (where FIG stands for factoring-instance-generator).

Definition 4.1 *The factoring-instance-generator, FIG, is a probabilistic polynomial-time algorithm such that on input* 1^n *its output,* $N = P \cdot Q$*, is distributed over* $n - bit$ *integers, where* P *and* Q *are two* $\frac{n}{2} - bit$ *primes and* $P = Q = 3 \bmod 4$ *(such* N *is known as a Blum-integer).*

A natural way to define FIG would be to let $FIG(1^n)$ be uniformly distributed over $n - bit$ Blum-integers.[9] However, other choices were previously considered (e.g., letting P and Q obey some "safety" conditions). We now formalize the assumption that factoring Blum-integers is hard.

Definition 4.2 (ϵ-factoring) *Let* $\mathcal{A}$ *be a probabilistic Turing machine and let* $\epsilon = \epsilon(n)$ *be a real-valued function.* $\mathcal{A}$ ϵ*-factors if for infinitely many* n*'s*

$$\Pr[\mathcal{A}(P \cdot Q) \in \{P, Q\}] > \epsilon(n)$$

where the distribution of $N = P \cdot Q$ *is* $FIG(1^n)$*.*

In spite of the extensive research directed towards the construction of efficient integer factoring algorithms, the best algorithms currently known for factoring an integer N, have (heuristic) running time $L(N) \overset{\text{def}}{=} e^{1.92(\log N)^{1/3}(\log\log N)^{2/3}}$. This (together with the fact that Blum-integers are a non-negligible fraction of all n-bit integers) leads us to the following assumption.

Assumption 4.1 (Factoring Blum-Integers) *Let* $\mathcal{A}$ *be any probabilistic polynomial-time oracle machine. There is no positive constant* α *such that* $\mathcal{A}$ $\frac{1}{n^\alpha}$*-factors.*

All exponentiations in the rest of this Section are in $\mathbb{Z}_N^*$. To simplify the notations, we omit the expression "mod N" from now on.

4.2 The Construction

Construction 4.1 *We define a function in the ensemble* $F = \{F_n\}_{n \in \mathbb{N}}$*. For every* $n \in \mathbb{N}$*, a key of a function in* F_n *is a tuple* $\langle N, \vec{a}, g, r \rangle$*, where* N *is an* n*-bit Blum-integer,* g *is a quadratic-residue in* $\mathbb{Z}_N^*$*,* $\vec{a} = \langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1} \rangle$ *is a sequence of* $2n$ *elements in* $[N]$ *and* r *is an* n*-bit string. For any* n *bit input* $x = x_1 x_2 \ldots x_n$*, and for every integer valued function* $\ell = \ell(n)$*, the function* $f_{N,\vec{a},g,r} : I^n \rightarrow I^{\ell(n)}$ *is defined by:*

$$f_{N,\vec{a},g,r}(x) \overset{\text{def}}{=} \mathcal{B}_r(g^{\Pi_{i=1}^n a_{i,x_i}}), \mathcal{B}_r(g^{2\Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^{\ell-1} \Pi_{i=1}^n a_{i,x_i}})$$

where $\mathcal{B}_r(m)$ *denotes the inner product,* $\langle m, r \rangle \bmod 2$*. The distribution of functions in* F_n *is induced by the following distribution on their keys:* $\vec{a}$*,* g *and* r *are uniform in their range and the distribution of* N *is* $FIG(1^n)$*.*

[9] We note that $n - bit$ Blum-integers are a non-negligible fraction of all $n - bit$ integers and that it is easy to sample a uniformly distributed $n - bit$ Blum-integer.

4.3 Efficiency of the Construction

Consider a function $f_{N,\vec{a},g,r} \in F_n$ as in Construction 4.1. Computing the value of this function at any given point, x, involves one multiple product $y = \prod_{i=1}^{n} a_{i,x_i}$ (which can be performed modulo $\varphi(N)$), one modular exponentiation, $z = g^y \bmod N$, and $\ell(n) - 2$ successive modular squarings $z^2, \ldots, z^{2^k}, \ldots, z^{2^{\ell-1}}$ (which require less than one modular multiplication each). The value of the function is finally obtained by computing $\mathcal{B}_r(z), \mathcal{B}_r(z^2), \ldots, \mathcal{B}_r(z^{2^{\ell-1}})$ (which is a cheap operation compared to modular multiplication). As discussed in [19], it is possible to use preprocessing in order to get improved efficiency.[10] This gives us a pseudorandom function which can be evaluated roughly at the cost of $2n + \ell(n)$ modular multiplications (a modular exponentiation is counted as n modular multiplications).

An attractive feature of our construction is that for each input we can have a *variable* length output, i.e. if for some x's one needs more bits in the output of $f(x)$, then the natural way of simply taking more bits of the form $\mathcal{B}_r(z^{2^j})$ works. While it is possible to get this feature generically, by combining a pseudorandom function and a generator, here we get it "for free."

4.4 Using Other Hard-Core Bits

In our construction (Construction 4.1) we use the Goldreich-Levin hard-core bit, $\mathcal{B}_r$. The other, more natural, hard-core bit in this context is the $\mathcal{LSB}$ predicate (shown to be secure by Alexi et al. [1]). The key property which we require from the $\mathcal{B}_r$ predicate is that its reconstruction algorithm *fixes* the unknown value and changes r throughout the process (see Footnote 14). As pointed out to us by Roger Fischlin [9], a similar property is satisfied by the $\mathcal{LSB}$-based reconstruction techniques [1, 10]. As a consequence, Construction 4.1 will remain secure even if we use the $\mathcal{LSB}$ predicate (which was the one originally used in the BBS construction). For simplicity of exposition we prefer to describe the proof of security while using the $\mathcal{B}_r$ predicate. The same arguments (with small modifications) will apply in the $\mathcal{LSB}$ setting.

5 Proof of Security

Theorem 5.1 *If the Factoring-assumption holds (Assumption 4.1) then $F = \{F_n\}_{n\in\mathbb{N}}$ (as in Construction 4.1) is an efficiently computable pseudorandom function ensemble.*

Remark 5.1 *The proof of Theorem 5.1 yields a more quantitative version as well: Assume that there exists a probabilistic polynomial-time oracle machine with running time $t(n)$ that distinguishes $f_{N,\vec{a},g,r}$ from $\rho_{n,\ell}$ with advantage $\epsilon(n)$ (where $\rho_{n,\ell}$ is uniformly distributed in the set of functions with domain $\{0,1\}^n$ and range $\{0,1\}^{\ell(n)}$). Let $q = q(n)$ be a bound on the number of queries made this machine. Then there exists a probabilistic polynomial-time algorithm with running time poly$(\frac{1}{\epsilon(n)}, t(n), \ell(n))$ that ϵ''-factors, for $\epsilon''(n)$ which equals $\Omega\left(\frac{\epsilon(n)^2}{q(n)^2 \cdot \ell(n)^2}\right)$.*

5.1 On the Methodology

5.1.1 A simple approach does not work

In order to prove Theorem 5.1, it might be tempting to use the following approach: Recall the definition of the function $h_{N,\vec{a},g}(x) = g^{\Pi_{i=1}^{n} a_{i,x_i}}$ in Equation 5. As mentioned above, it was shown

[10] The most obvious preprocessing would be to compute the values g^{2^i} (for every positive integer i up to the length of $(P-1)\cdot(Q-1)$). See [19] for additional preprocessing techniques which further improve the efficiency.

in [19, 20] that $h_{N,\vec{a},g}$ is unpredictable against an *adaptive* sample and a *random* challenge. In light of this, Construction 4.1 can be viewed as based on the following methodology:

1. Take an "unpredictable" function $h_s : \{0,1\}^n \rightarrow \{0,1\}^n$
2. Take a pseudorandom generator $G : \{0,1\}^n \rightarrow \{0,1\}^\ell$
3. Obtain a pseudorandom function $f : \{0,1\}^n \rightarrow \{0,1\}^\ell$ by setting $f_s(x) = G(h_s(x))$.

Unfortunately, this method does not work in general. As will be demonstrated next, there exists an "unpredictable" function and a pseudorandom generator such that their composition is *not* a pseudorandom function.

5.1.2 The Counter Example

Consider the following (unnatural) counter-example:

1. (a) Let $h'_s : \{0,1\}^n \rightarrow \{0,1\}^{\frac{n}{2}}$ be an unpredictable function.
   (b) For $y \in \{0,1\}^{\frac{n}{2}}$ define $h_{s,y} : \{0,1\}^n \rightarrow \{0,1\}^n$ as $h_{s,y}(x) = \langle h'_s(x), y \rangle$.
   (c) Clearly $h_{s,y}$ is unpredictable.
2. (a) Let $G' : \{0,1\}^{\frac{n}{2}} \rightarrow \{0,1\}^\ell$ be a pseudo-random generator.
   (b) For $z, y \in \{0,1\}^{\frac{n}{2}}$ define $G : \{0,1\}^n \rightarrow \{0,1\}^\ell$ as $G(z,y) = G'(y)$.
   (c) Clearly G is pseudo-random.
3. However, the function $f : \{0,1\}^n \rightarrow \{0,1\}^\ell$ obtained by setting $f_{s,y}(x) = G(h_{s,y}(x))$ is always equal to $G'(y)$ (regardless of the value of x). Obviously, $f_{s,y}$ cannot be pseudo-random.

It seems that the reason our construction *does* work lies in the *specific* number theoretic features which the function $g^{\prod_{i=1}^{n} a_{i,x_i}}$ and the BBS generator have in common. Since we do not know what precisely are the features of a function h_s and of a pseudorandom generator G that are needed in order to obtain a pseudorandom function (using the above methodology), we are forced to provide a direct proof for our construction. As most proofs of pseudorandomness we will use a *hybrid argument*, i.e. mixing a truly random and pseudorandom distribution. The type of hybrid we apply is the *reverse hybrid*, where first the random part is used and only then the pseudorandom one. This is an instance of the principle of deferred decision (see [17]): Do not commit to any value in the pseudorandom part of the distribution until you have to.

5.2 Proof of Theorem 5.1

Proof: Let $F = \{F_n\}_{n \in \mathbb{N}}$ be as in construction 4.1. It is clear that F is efficiently computable. Assume that F is not pseudorandom, then there exists a probabilistic polynomial-time oracle machine $\mathcal{M}$ and a non-negligible real valued function $\epsilon = \epsilon(n)$, such that for infinitely many n's

$$\left| \Pr\left[\mathcal{M}^{f_{N,\vec{a},g,r}}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{\rho_{n,\ell}}(1^n) = 1\right] \right| > \epsilon(n) \qquad (7)$$

where in the first probability $f_{N,\vec{a},g,r}$ is distributed according to F_n and in the second probability $\rho_{n,\ell}$ is distributed according to $R_{n,\ell}$ (the uniform ensemble of functions with domain $\{0,1\}^n$ and range $\{0,1\}^{\ell(n)}$).

A Hybrid Black-Box: Equation (7) tells us that there is a non-negligible difference between the output behavior of $\mathcal{M}$ in the case it is given access to a black-box which answers according to $f_{N,\vec{a},g,r}$ and in the case it is given access to a black-box which answers according to $\rho_{n,\ell}$. However, $\mathcal{M}$'s response is still a well defined random variable even when its queries are answered according to some other distribution. This means that we are allowed to invoke $\mathcal{M}$, and answer its queries in whatever way we find suitable for our purposes. The way we choose to do it is by defining a *hybrid black-box.*[11] Informally, this is a black-box which starts by answering $\mathcal{M}$'s queries according to $\rho_{n,\ell}$, and then switches mode to continue and answer according to $f_{N,\vec{a},g,r}$.

Let $t = t(n)$ be a polynomial that bounds the running time of $\mathcal{M}$, assume wlog that $\mathcal{M}$ always makes exactly t queries. Since each single answer given to these queries is ℓ-bit long, we have that the total number of bits which $\mathcal{M}$ gets as answers during its execution is precisely $t \cdot \ell$. Each one of these bits corresponds to a location for which one of the hybrid black-box distributions will switch from answering according to $\rho_{n,\ell}$ to answering according to $f_{N,\vec{a},g,r}$ (note that this may also happen in the middle of an answer).

Definition 5.1 (Hybrid Black-Box) *Let J be an element in $[t \cdot \ell + 1]$ written as $J = I \cdot \ell + k$ (where $0 \leq I \leq t$ and $0 \leq k < \ell$). The J^{th} hybrid black-box, $H^J_{N,\vec{a},g,r}$, is defined by the answers it gives to $\mathcal{M}$'s queries. The first I queries are answered according to $\rho_{n,\ell}$ (i.e. at random), the answer to the $(I+1)^{st}$ query up to the k^{th} bit-location is according to $\rho_{n,\ell}$, and from then on, according to $f_{N,\vec{a},g,r}$. All remaining queries are answered according to $f_{N,\vec{a},g,r}$.*

Notice that $H^0_{N,\vec{a},g,r}$ is a black-box which always answers according to $f_{N,\vec{a},g,r}$, whereas $H^{t\cdot\ell}_{N,\vec{a},g,r}$ always answers according to $\rho_{n,\ell}$. By Equation 7, we have that $\mathcal{M}$ distinguishes between $f_{N,\vec{a},g,r}$ and $\rho_{n,\ell}$ with advantage $\epsilon(n)$. Therefore, if we pick J at random, the expected advantage that $\mathcal{M}$ has in distinguishing between $H^J_{N,\vec{a},g,r}$ and $H^{J+1}_{N,\vec{a},g,r}$ is at least $\epsilon'(n) = \frac{\epsilon(n)}{t(n)\cdot\ell(n)}$.

5.2.1 Simplified Proof

We start by giving a simplified version of the proof under the assumption that $\mathcal{M}$ decides membership in F_n with advantage $\epsilon(n)$ for *any* sequence $\vec{a}$ of $2n$ elements in $[N]$ (recall that $\vec{a}$ equals $\langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1}\rangle$). We then proceed and show how to modify the proof so it will work for a randomly chosen $\vec{a}$.

Roughly speaking, we show how given a distinguisher for our pseudorandom functions, we can construct an algorithm that on input $\langle v^2 \bmod N, N, r\rangle$ predicts the value of $\mathcal{B}_r(u)$, where u is the unique quadratic residue in $\mathbb{Z}^*_N$ which satisfies $u^2 = v^2 \bmod N$. Using the Goldreich-Levin reconstruction algorithm we are then able to retrieve u (see [11] for details). This means that we can extract square-roots in $\mathbb{Z}^*_N$ and consequently factor Blum-integers (as described in [22]).

Theorem 5.2 (Goldreich-Levin [14]) *Let $z, r \in \{0,1\}^n$, using an oracle that, on input r, predicts the value of $\mathcal{B}_r(z)$ with advantage $\epsilon(n)$ (over the choice of r) in time $t(n)$, there exists a probabilistic polynomial-time algorithm with running time $O(\frac{n^2\cdot t(n)}{\epsilon(n)^2})$ that retrieves z with probability at least $\Omega(\epsilon(n))$.*

The following Lemma can be viewed as the heart of the simplified part of the proof. It describes how given a distinguisher for our pseudorandom functions, it is possible to construct an algorithm, $\mathcal{D}$, which can be used by the reconstruction algorithm as an oracle for the value of $\mathcal{B}_r(u)$.

[11] This is just a methodological modification of the standard hybrid technique (see [11] for details on the hybrid technique).

Lemma 5.3 *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$, satisfying Equation 7 for* any *choice of $\vec{a}$. Then there exists a probabilistic polynomial-time algorithm $\mathcal{D}$, such that for infinitely many n's*

$$\left|\Pr\left[\mathcal{D}(v^2, N, r, \mathcal{B}_r(u)) = 1\right] - \Pr\left[\mathcal{D}(v^2, N, r, b) = 1\right]\right| > \epsilon'(n)$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}_N^*$, r is a random n-bit string, $b \in_R \{0,1\}$, and u is the unique quadratic residue in $\mathbb{Z}_N^*$ which satisfies $u^2 = v^2 \bmod N$.

Using an averaging argument, it can be shown that on at least an $\frac{\epsilon'(n)}{2}$ fraction of the choices of N and v^2, algorithm $\mathcal{D}$ distinguishes the value of $\mathcal{B}_r(u)$ from random with advantage $\frac{\epsilon'(n)}{2}$ (over the choice of r). In particular, $\mathcal{D}$ can be used in order to predict $\mathcal{B}_r(u)$ with probability $\frac{1}{2} + \frac{\epsilon'(n)}{4}$. By Theorem 5.2 we know that we can use $\mathcal{D}$ in order to construct a probabilistic polynomial-time algorithm that retrieves $u \bmod N$ with probability $\Omega\left(\epsilon'(n)\right)$. We now have that $u^2 = v^2 \bmod N$ and $\Pr\left[u = \pm v\right] = 1/2$. This implies (cf. [22]) that $\Pr\left[gcd(u - v, N) \in \{P, Q\}\right] = 1/2$, which enables us to construct an algorithm that $\Omega\left(\epsilon'(n)^2\right)$-factors, in contradiction to Assumption 4.1.

Description of $\mathcal{D}$: Let the input of $\mathcal{D}$ be $\langle v^2, N, r, \alpha\rangle$, where the distribution of N, v and r is as in Lemma 5.3. Let u be the unique quadratic residue in $\mathbb{Z}_N^*$ which satisfies $u^2 = v^2 \bmod N$. On this input, $\mathcal{D}$ first picks $J = I \cdot \ell + k$ at random in $[t \cdot \ell]$. Then $\mathcal{D}$ invokes $\mathcal{M}$ and answers its queries in a way that simulates $H^J_{N,\vec{a},g,r}$ (for some value of $\vec{a}$ and g) if α equals $\mathcal{B}_r(u)$, and simulates $H^{J+1}_{N,\vec{a},g,r}$ if α is a random bit. $\mathcal{D}$ will now be able to utilize the expected advantage that $\mathcal{M}$ has in distinguishing between $H^J_{N,\vec{a},g,r}$ and $H^{J+1}_{N,\vec{a},g,r}$ in order to guess the actual distribution of α. Specifically, $\mathcal{D}$ will answer $\mathcal{M}$'s queries in the following way:

1. Answer the first I queries according to $\rho_{n,\ell}$.

2. Answer the $(I+1)^{st}$ query with $b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(u^2), \ldots, \mathcal{B}_r(u^{2^{\ell-k-1}})$ (where $b_0 b_1 \ldots b_{k-1}$ denotes a random k-bit string).

3. Answer the remaining queries consistently according to $f_{N,\vec{a},g,r}$.

The challenge in constructing $\mathcal{D}$ is to assign values to $\vec{a}$ and g such that the above answers will be distributed according to the correct hybrid black-box distribution, and will be efficiently computable by $\mathcal{D}$.

Defining $\vec{a}$ and g: The way we define the value of g depends on the choice of J, whereas the values assigned to $\vec{a}$ will depend on the $(I+1)^{st}$ query made by $\mathcal{M}$. We require that if x is the $(I+1)^{st}$ query, then the value of the $(k+1)^{st}$ bit of $f_{N,\vec{a},g,r}(x)$ (which is answered with α by $\mathcal{D}$), will be equal to $\mathcal{B}_r(u)$. In addition, we require that $\mathcal{D}$ will be able to efficiently compute the answers to all the subsequent queries made by $\mathcal{M}$ (starting from the $(k+2)^{nd}$ bit-location in the answer to the $(I+1)^{st}$ query).

The definition of g: $\mathcal{D}$ computes $s = \ell \cdot n - k$ and sets $g = v^{2^s} \bmod N$.

Claim 5.1 *Let γ be the order of g in $\mathbb{Z}_N^*$, then γ is odd.*

Proof: It will be sufficient to show that the size of Q_N is odd. This implies that all quadratic residues in $\mathbb{Z}_N^*$ (and in particular g) have odd order. Since N is a Blum-integer, for every quadratic residue in $\mathbb{Z}_N^*$ exactly one of its four square roots resides in Q_N. This means that $|Q_N| = \frac{|\mathbb{Z}_N^*|}{4} = \frac{(P-1)\cdot(Q-1)}{4}$. Since $P = Q = 3 \bmod 4$, then $\frac{P-1}{2}$ and $\frac{Q-1}{2}$ are odd, which implies that $\frac{(P-1)\cdot(Q-1)}{4}$ is also odd and the claim follows. □

Claim 5.2 *For every $0 < i < s$, $g^{2^{-i} \bmod \gamma} = v^{2^{s-i}} \bmod N$.*[12]

Proof: By Claim 5.1 we have that γ is odd. This implies that $2 \in \mathbb{Z}_\gamma^*$ and therefore $2^{-1} \bmod \gamma$ exists (and is simply $\frac{\gamma+1}{2}$). For simplicity of exposition, let us denote by $g^{2^{-1}}$ the value $g^{2^{-1} \bmod \gamma} \bmod N$. We now have that whenever 2^{-1} appears in the exponent it denotes the value $2^{-1} \bmod \gamma = \frac{\gamma+1}{2}$. Similarly, 2^{-i} in the exponent denotes the value $2^{-i} \bmod \gamma = (\frac{\gamma+1}{2})^i \bmod \gamma$. Therefore, for every i the value $g^{2^{-i}} \bmod N$ is a quadratic-residue (since g is a quadratic-residue). Take $i = 1$, we now have that both $g^{2^{-1}}$ and $v^{2^{s-1}}$ are square roots of g and they are both quadratic-residues. Since squaring is a permutation over the set of quadratic-residues in $\mathbb{Z}_N^*$ (for any Blum-integer N) we must have that $g^{2^{-1}}$ and $v^{2^{s-1}}$ are equal. By induction on $0 < i < s$, we get that $g^{2^{-i}} = v^{2^{s-i}} \bmod N$. □

Corollary 5.1 *Let $u = g^{2^{-s}} \bmod N$, then $u^2 = v^2 \bmod N$.*

The definition of $\vec{a}$: Since the first I queries of $\mathcal{M}$ are answered randomly, we can defer the assignment to the values of $\vec{a}$ until $\mathcal{D}$ is given the $(I+1)^{st}$ query, $x = x_1 x_2 \ldots x_n$. It is then possible to define $\vec{a}$ so that the value of the $(k+1)^{st}$ bit of $f_{N,\vec{a},g,r}(x)$ (namely, $\mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}})$) will be equal to $\mathcal{B}_r(u)$.[13] Let $\vec{a}$ be the vector $\langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1} \rangle$, where for all i, $a_{i,x_i} = 2^{-\ell} \bmod \gamma$ and $a_{i,\overline{x_i}}$ is uniformly distributed in $[N]$.

Claim 5.3 *Let $\vec{a}, g$ and u be defined as above, then*

$$\mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}) = \mathcal{B}_r(u)$$

Proof: By the above notations,

$$g^{2^k \Pi_{i=1}^n a_{i,x_i}} = g^{2^{k-\ell n}} = g^{2^{-s}} = u \bmod N$$

and the claim follows. □

The Running-Time of $\mathcal{D}$: We now show that $\mathcal{D}$ is indeed able to complete steps (2) and (3) (i.e. answer all the remaining queries of $\mathcal{M}$ starting from the $(k+2)^{nd}$ bit-location in the answer to the $(I+1)^{st}$ query in a way which is consistent with the definition of $\vec{a}$ and g). The key point is that $\mathcal{D}$ can achieve this task even though it does not actually know the values of a_{i,x_i}.

Claim 5.4 *Algorithm $\mathcal{D}$ is able to efficiently complete the answer to the $(I+1)^{st}$ query (step 2).*

Proof: Since $v^2 = u^2 \bmod N$, and since v^2 and r are given to him in the input, $\mathcal{D}$ is able to complete the answer to the $(I+1)^{st}$ query and answer $\mathcal{M}$ with

$$b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(v^2), \ldots, \mathcal{B}_r(v^{2^{\ell-k-1}})$$

as required. □

[12] Note that γ is not known to $\mathcal{D}$. However, as long as $s - i > 0$, it is possible for $\mathcal{D}$ to compute $v^{2^{s-i}} \bmod N$.

[13] It is worth noticing that this would not have been possible in the case that $\mathcal{D}$ would have answered $\mathcal{M}$'s queries in a reverse order (i.e by first answering according to $f_{N,\vec{a},g,r}$ and then switching mode to $\rho_{n,\ell}$).

Claim 5.5 *For any query $y \neq x$, $\mathcal{D}$ is able to efficiently compute the value of $f_{N,\vec{a},g,r}(y)$ (step 3).*

Proof: It will be sufficient to show that for any query $y = y_1 y_2 \dots y_n \neq x$, $\mathcal{D}$ is able to compute the value $g^{\Pi_{i=1}^n a_{i,y_i}}$, and thus it is always able to answer the query with the value of $f_{N,\vec{a},g,r}(y)$. Now:

$$\begin{aligned} g^{\Pi_{i=1}^n a_{i,y_i}} &= g^{(\Pi_{\{y_i=x_i\}} a_{i,y_i})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \\ &= g^{(2^{-\ell j})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \\ &= v^{(2^{s-\ell j})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \bmod N \end{aligned}$$

where j is the number of locations for which y_i equals x_i. Since $j < n$ (remember that $y \neq x$), and since $k < \ell$ then we always have that $s - \ell j = (n-j)\ell - k$ is at least 1 over the integers. As we have already stated, $\mathcal{D}$ knows the value of v^2, therefore by performing the appropriate exponentiations, it is always able to compute $g^{\Pi_{i=1}^n a_{i,y_i}} = v^{(2^{s-\ell j})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})}$, as required (remember that $\mathcal{D}$ knows the value of a_{i,y_i} for all $y_i \neq x_i$). □

We are finally ready to establish Lemma 5.3. Recall that given some J, the value of α determines which of the possible hybrid black-boxes is simulated by $\mathcal{D}$. If α equals $\mathcal{B}_r(u)$, then $\mathcal{D}$ simulates $H^J_{N,\vec{a},g,r}$, whereas if α is a random bit, $\mathcal{D}$ simulates $H^{J+1}_{N,\vec{a},g,r}$. Since the expected advantage $\mathcal{M}$ has in distinguishing the above neighboring hybrid black-boxes is $\epsilon'(n)$ and since $\mathcal{D}$ picks J at random, we expect that $\mathcal{D}$ will be able to decide with advantage $\epsilon'(n)$ whether α is indeed the value of $\mathcal{B}_r(u)$.

5.2.2 Completing the Proof

To complete the proof we follow the same lines, along with an additional "randomization" of the values in $\vec{a}$, achieved by taking $a_{i,x_i} = \xi_i + 2^{-\ell} \bmod \gamma$. This causes the value of the $(k+1)^{st}$ bit in the answer to the $(I+1)^{st}$ query to change into $\mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}) = \mathcal{B}_r(g^{2^k \Pi_{i=1}^n (\xi_i + 2^{-\ell})}) = \mathcal{B}_r(u \cdot w)$, where w is an element in $\mathbb{Z}_N^*$ which is completely determined by the ξ_i's and by the value of v^2 (and is efficiently computable given the above values). Note that now algorithm $\mathcal{D}$ becomes an oracle for the value of $\mathcal{B}_r(u \cdot w)$, and will therefore be used in order to retrieve $u \cdot w$ (rather than u).

Jumping ahead, we note that letting $\mathcal{D}$ pick the random ξ_i's by himself would have caused the value of w (and therefore $u \cdot w$) to change each time $\mathcal{D}$ is invoked. This would not allow the reconstruction algorithm to retrieve $u \cdot w$ for any specific value of w. The solution to this problem will be to fix the values which determine w in advance, and then use $\mathcal{D}$ (which now takes only r as input) as an oracle for $\mathcal{B}_r(u \cdot w)$.[14] For the time being, we ignore the above issue, and let $\mathcal{D}$ pick the random ξ_i's by himself. We now give the (full) analogue of Lemma 5.3.

Lemma 5.4 *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$, satisfying Equation 7. Then there exists a probabilistic polynomial-time algorithm $\mathcal{D}$, such that for infinitely many n's*

$$\left| \Pr\left[\mathcal{D}(v^2, N, r, \mathcal{B}_r(u \cdot w)) = 1\right] - \Pr\left[\mathcal{D}(v^2, N, r, b) = 1\right] \right| > \epsilon'(n) - n \cdot 2^{-O(n)}$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}_N^*$, r is a random n-bit string, $b \in_R \{0,1\}$, $u \cdot w$ is the unique quadratic residue in $\mathbb{Z}_N^*$ which satisfies $(u \cdot w)^2 = v^2 \cdot w^2 \bmod N$, and w is a randomly chosen quadratic residue in $\mathbb{Z}_N^*$ (which is completely determined by the value of v^2 and $\mathcal{D}$'s internal coin tosses and is efficiently computable by $\mathcal{D}$).

[14] Here we use a special property of the $\mathcal{B}_r$ predicate, which enables the reconstruction of $u \cdot w$ by only asking queries which refer to $u \cdot w$ (i.e. $u \cdot w$ is *fixed* throughout the process, and only r changes from one query to another). As discussed in Section 4.4, a similar property is satisfied by the $\mathcal{LSB}$-based reconstruction techniques by Alexi et al. [1, 10]. As a consequence, Construction 4.1 remains secure even if we use the $\mathcal{LSB}$ predicate (which was the one originally used in the BBS construction).

Proof: On input $\langle v^2, N, r, \alpha \rangle$, $\mathcal{D}$ is defined as follows:

1. (a) Sample $J = I \cdot \ell + k$ uniformly at random in $[t \cdot \ell]$.
   (b) Sample J random bits (needed in order to simulate the hybrid black-box).
   (c) Sample $\vec{\xi} = \langle \xi_1, \xi_2, \ldots, \xi_n, \xi'_1, \xi'_2, \ldots, \xi'_n \rangle$ by uniformly picking $2n$ elements in $[N]$.
2. Compute $s = \ell \cdot n - k$ and set $g = v^{2^s} \bmod N$.
3. Invoke $\mathcal{M}$ on input $\langle 1^n \rangle$:
   (a) Answer each of its first I queries with a random string in $\{0,1\}^\ell$.
   (b) Let $x_1 x_2 \ldots x_n$ be $\mathcal{M}$'s $(I+1)^{st}$ query.
   For $1 \leq i \leq n$, denote by a_{i,x_i} the value $\xi_i + 2^{-\ell} \bmod \gamma$, and by $a_{i,\overline{x_i}}$ the value ξ'_i. Denote by $\vec{a}$ the sequence $\langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1} \rangle$ (a_{i,x_i} is not actually known to $\mathcal{D}$).
   Let $b_0 b_1 \ldots b_{k-1}$ be a random k-bit string, answer the $(I+1)^{st}$ query with the ℓ-bit string
   $$b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(g^{2^{k+1}\Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^{\ell-1}\Pi_{i=1}^n a_{i,x_i}})$$
   (c) Answer each remaining query of $\mathcal{M}$, y, with the value $f_{N,\vec{a},g,r}(y)$.
4. If $\mathcal{M}$ outputs 1, then output 1.
   If $\mathcal{M}$ outputs 0, then output 0.

Why does $\mathcal{D}$ Predict the Value of $\mathcal{B}_r(u \cdot w)$: To define u, we set $u = g^{2^{-s}} \bmod N$. As before (see Corollary 5.1), we have that $u^2 = v^2 \bmod N$. To define w, we set $w = v^{(\Sigma_{i=0}^{n-1} \beta_i 2^{\ell(n-i)})} \bmod N$, where β_i are the coefficients (over $\mathbb{Z}$) of the polynomial $p(x) = \prod_{i=1}^n (\xi_i + x) = x^n + \sum_{i=0}^{n-1} \beta_i x^i$. Note that the β_i's can be efficiently computed given the ξ_i's (either recursively or by interpolation). Given the values of the β_i's and of v^2, we are able to compute w (note that the exponent of v in the definition of w is always even). Therefore, w is an efficiently computable quadratic residue in $\mathbb{Z}_N^*$ which is completely determined by the value of v^2 and $\mathcal{D}$'s internal coin tosses (i.e. the ξ_i's sampled in step (1c)).

Claim 5.6 *Let $\vec{a}, g, u$ and w be defined as above, then $\mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}) = \mathcal{B}_r(u \cdot w)$.*

Proof: Using the above notations (and Claim 5.2), we have:

$$\begin{aligned}
g^{2^k \Pi_{i=1}^n a_{i,x_i}} &= g^{2^k \Pi_{i=1}^n (\xi_i + 2^{-\ell})} \\
&= g^{2^k p(2^{-\ell})} \\
&= g^{2^{k-\ell n} + \Sigma_{i=1}^{n-1} \beta_i 2^{-(\ell i - k)}} \\
&= g^{2^{-s}} \cdot v^{(\Sigma_{i=1}^{n-1} \beta_i 2^{s-(\ell i-k)})} \\
&= u \cdot v^{(\Sigma_{i=1}^{n-1} \beta_i 2^{\ell(n-i)})} \\
&= u \cdot w \bmod N
\end{aligned}$$

and the claim follows. □

This implies that the $(k+1)^{st}$ bit in the answer that the $f_{N,\vec{a},g,r}$ black box is supposed to give to the $(I+1)^{st}$ query (and is answered with α instead), is equal to $\mathcal{B}_r(u \cdot w)$. As we have already seen, this fact can be used by $\mathcal{D}$ in order to decide whether α equals $\mathcal{B}_r(u \cdot w)$.

The Running-Time of $\mathcal{D}$: It is clear that steps (1),(2) and (3a) can be carried out in time $poly(n, \ell(n))$. In order to prove that steps (3b) and (3c) can be carried out in time $poly(n, \ell(n)) \cdot t(n)$, we observe that if for some j in $[\ell]$, $\mathcal{D}$ is able to compute $\lambda = g^{2^j \Pi_{i=1}^n a_{i,x_i}} \bmod N$, then by squaring and taking the inner product of the results with r, it is also able to compute any bit-sequence of the form $\mathcal{B}_r(\lambda), \mathcal{B}_r(\lambda^2), \ldots, \mathcal{B}_r(\lambda^{2^{\ell-j-1}})$.

Claim 5.7 *Algorithm $\mathcal{D}$ is able to efficiently complete the answer to the $(I+1)^{st}$ query (step 3b).*

Proof: By the above observation, it will be sufficient to show that $\mathcal{D}$ is able to efficiently compute the value of $g^{2^{k+1} \Pi_{i=1}^n a_{i,x_i}}$. To see that, notice that $g^{2^k \Pi_{i=1}^n a_{i,x_i}}$ equals $u \cdot w$ (see Claim 5.6). This implies that $g^{2^{k+1} \Pi_{i=1}^n a_{i,x_i}} = u^2 \cdot w^2 = v^2 \cdot w^2 \bmod N$. Since both the values of v^2 and w^2 are known to $\mathcal{D}$, it is possible for him to efficiently answer the $(I+1)^{st}$ query (as described in step 3b). □

Claim 5.8 *For* any *query $y \neq x$, $\mathcal{D}$ is able to efficiently compute the value of $f_{N,\vec{a},g,r}(y)$ (step 3c).*

Proof: By the above observation, it will be sufficient to show that $\mathcal{D}$ is able to compute the value $g^{\Pi_{i=1}^n a_{i,y_i}}$ (and consequently it will be able to compute the value of $f_{N,\vec{a},g,r}(y)$). Given a query $y = y_1 y_2 \ldots y_n \neq x$, $\mathcal{D}$ starts by computing (in time $poly(n)$) the coefficients, $\delta_i \in \mathbb{Z}$, of the polynomial $q(x) = \prod_{\{y_i = x_i\}} (\xi_i + x) = \sum_{i=0}^j \delta_i x^i$ (where j is the number of locations for which y_i equals x_i). Under this notation, $\prod_{\{y_i = x_i\}} a_{i,y_i}$ equals $q(2^{-\ell})$, we then have:

$$\begin{aligned} g^{\Pi_{i=1}^n a_{i,y_i}} &= g^{(\Pi_{\{y_i=x_i\}} a_{i,y_i})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \\ &= g^{(\Sigma_{i=0}^j \delta_i 2^{-\ell i})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \\ &= \prod_{i=0}^j g^{(\delta_i 2^{-\ell i})(\Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \\ &= \prod_{i=0}^j v^{2^{s-\ell i}(\delta_i \Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \bmod N \end{aligned}$$

since $i \leq j < n$ (remember that $y \neq x$), and since $k < \ell$ then we always have that the value of $s - \ell i = (n-i)\ell - k$ is at least 1 over the integers. Therefore, $\mathcal{D}$ is always able to compute all the values $v^{2^{s-\ell i}(\delta_i \Pi_{\{y_i \neq x_i\}} a_{i,y_i})} \bmod N$ (by performing the appropriate exponentiations of v^2, remember that $\mathcal{D}$ knows the value of all δ_i's, and the value of a_{i,y_i} for all $y_i \neq x_i$). Finally, by taking the product of the above values (reduced mod N), $\mathcal{D}$ is able to compute the value of $g^{\Pi_{i=1}^n a_{i,y_i}}$, as required. □

The Success-Probability of $\mathcal{D}$: To find the success probability of $\mathcal{D}$, we notice that the distribution of the function $f_{N,\vec{a},g,r}$ (which is induced by the way $\mathcal{D}$ chooses $\vec{a}$ and g) is statistically close to the distribution of functions in F_n. To see this, we will need the following claims regarding the distributions of $\vec{a}$ and g.

Claim 5.9 *g is a uniformly distributed quadratic residue in $\mathbb{Z}_N^*$.*

Proof: Since v^2 is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$ and squaring is a permutation over the set of quadratic-residues in $\mathbb{Z}_N^*$, it immediately follows that $g = v^{2^s}$ is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$. □

Claim 5.10 *Let ξ_i and a'_{i,x_i} be uniformly distributed elements in $[N]$, and denote by a_{i,x_i} the value $\xi_i + 2^{-\ell} \bmod \gamma$. Then the statistical distance of a_{i,x_i} and $a'_{i,x_i} \bmod \gamma$ is $2^{-O(n)}$.*

Proof: Note that γ divides $(P-1)(Q-1)$. Therefore, the distribution of a_{i,x_i} conditioned on the event that $\xi_i \in [(P-1)(Q-1)]$ is the same as the distribution of $a'_{i,x_i} \bmod \gamma$ conditioned on the event that $a'_{i,x_i} \in [(P-1)(Q-1)]$ (and in both cases it is simply the uniform distribution over $[\gamma]$). It remains to notice that:

$$\begin{aligned}
\Pr\left[\xi_i \in [(P-1)(Q-1)]\right] &= \Pr\left[a'_{i,x_i} \in [(P-1)(Q-1)]\right] \\
&= \frac{(P-1)(Q-1)}{N} \\
&= 1 - \tfrac{P+Q}{N} + \tfrac{1}{N} \\
&= 1 - 2^{-O(n)}
\end{aligned}$$

which completes the proof. □

Claim 5.11 *Let $f_{N,\vec{a}',g,r}$ be distributed according to F_n and let $f_{N,\vec{a},g,r}$ be distributed as in the construction of $\mathcal{D}$. Then the statistical distance of $f_{N,\vec{a}',g,r}$ and $f_{N,\vec{a},g,r}$ is $n \cdot 2^{-O(n)}$.*

Proof: Let each element in $\vec{a}' = \langle a'_{1,0}, a'_{1,1}, a'_{2,0}, a'_{2,1}, \ldots, a'_{n,0}, a'_{n,1}\rangle$ be uniformly distributed in $[N]$. By Claim 5.10 we have that for every $1 \le i \le n$, a_{i,x_i} and $a'_{i,x_i} \bmod \gamma$ are of statistical distance $2^{-O(n)}$. It follows (by the triangle inequality), that $\vec{a}$ and $\vec{a}' \bmod \gamma$ are of statistical distance $n \cdot 2^{-O(n)}$. It is then immediate that $f_{N,\vec{a},g,r}$ and $f_{N,\vec{a}',g,r}$ are of statistical distance $n \cdot 2^{-O(n)}$. □

Since applying any function (even a randomized one) does not increase the statistical distance, then by Claim 5.11 we have that for infinitely many n's

$$\left|\Pr\left[\mathcal{M}^{f_{N,\vec{a}',g,r}}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{f_{N,\vec{a},g,r}}(1^n) = 1\right]\right| < n \cdot 2^{-O(n)}$$

To complete the proof, we recall Definition 5.1 of the hybrid distribution, $H^J_{N,\vec{a},g,r}$. By Claim 5.6 we have that $g^{2^k \Pi_{i=1}^n a_{i,x_i}} = u \cdot w \bmod N$. Therefore, the value of the $(k+1)^{st}$ bit in the answer that $\mathcal{D}$ is supposed to give to the $(I+1)^{st}$ query is precisely equal to $\mathcal{B}_r(u \cdot w)$. Given that $J = j$ and that $\alpha = \mathcal{B}_r(u \cdot w)$ (where $u \cdot w$ is a quadratic residue in $\mathbb{Z}^*_N$), the distribution that $\mathcal{M}$ sees is $H^j_{N,\vec{a},g,r}$. On the other hand, if α is random then the distribution that $\mathcal{M}$ sees is $H^{j+1}_{N,\vec{a},g,r}$. We now have that for infinitely many n's

$$\begin{aligned}
&\left|\Pr\left[\mathcal{D}(v^2 \bmod N, N, r, \mathcal{B}_r(u \cdot w)) = 1\right] - \Pr\left[\mathcal{D}(v^2 \bmod N, N, r, b) = 1\right]\right| \\
=\ & \frac{1}{t(n)\cdot\ell(n)} \cdot \left|\sum_{j=0}^{t\cdot\ell-1} \left(\Pr\left[\mathcal{D}(v^2, N, r, \mathcal{B}_r(u \cdot w)) = 1 \mid J = j\right] - \Pr\left[\mathcal{D}(v^2, N, r, b) = 1 \mid J = j\right]\right)\right| \\
=\ & \frac{1}{t(n)\cdot\ell(n)} \cdot \left|\sum_{j=0}^{t\cdot\ell-1} \left(\Pr\left[\mathcal{M}^{H^j_{N,\vec{a},g,r}}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{H^{j+1}_{N,\vec{a},g,r}}(1^n) = 1\right]\right)\right| \\
=\ & \frac{1}{t(n)\cdot\ell(n)} \cdot \left|\Pr\left[\mathcal{M}^{f_{N,\vec{a},g,r}}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{\rho_{n,\ell}}(1^n) = 1\right]\right| \\
\in\ & \frac{\epsilon(n)}{t(n)\cdot\ell(n)} \pm n \cdot 2^{-O(n)} \qquad (8)
\end{aligned}$$

where the distribution of $N = P\cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}_N^*$, r is a random n-bit string, $b \in_R \{0,1\}$, $u\cdot w$ is the unique quadratic residue in $\mathbb{Z}_N^*$ which satisfies $(u\cdot w)^2 = v^2\cdot w^2 \bmod N$, and w is a randomly chosen quadratic residue in $\mathbb{Z}_N^*$. The proof of Lemma 5.4 is complete. ■

Remark 5.2 *From the proof of Lemma 5.4 we get that $\mathcal{D}$ works even if the distinguisher $\mathcal{M}$ has access to N, g and r.*

Reconstructing $u \cdot w \bmod N$**:** Technically speaking, $\mathcal{D}$ is not suitable to be used as an oracle for the Goldreich-Levin reconstruction algorithm. First of all, because it is not a predictor for the value $\mathcal{B}_r(u\cdot w)$, but rather a distinguisher. Furthermore, the value of w potentially changes each time $\mathcal{D}$ is invoked (since it depends on v^2 and $\vec{\xi}$), which does not allow the reconstruction algorithm to retrieve $u \cdot w$ for any specific value of w. The transformation to a suitable predictor however is not difficult. By fixing the values of v^2 and $\vec{\xi}$ in advance, we are able to construct an algorithm, $\mathcal{D}_{N,\vec{\xi},v^2}$, which invokes $\mathcal{D}$ as a subroutine, and with non-negligible probability succeeds in predicting $\mathcal{B}_r(u\cdot w)$. On input $\langle r\rangle$, $\mathcal{D}_{N,\vec{\xi},v^2}$ is defined as follows:

1. Sample two independent random bits α, β in $\{0,1\}$.
2. Invoke $\mathcal{D}$ on input $\langle v^2, N, r, \alpha\rangle$, feed it with $\vec{\xi}$ on its random tape.
3. (a) If $\mathcal{D}$ outputs 1, then output α.
   (b) If $\mathcal{D}$ outputs 0, then output β.

Note that now, the value of w does not change each time $\mathcal{D}_{N,\vec{\xi},v^2}$ is invoked. This means that it is possible to use $\mathcal{D}_{N,\vec{\xi},v^2}$ as an oracle in order to reconstruct $u \cdot w$.

Lemma 5.5 *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$, satisfying Equation 7, and let $\mathcal{D}_{N,\vec{\xi},v^2}$ be as above. Then with probability $\frac{\epsilon'(n)}{2}$ (over the choices of N,v^2 and $\vec{\xi}$), it holds that for infinitely many n's*

$$\Pr\left[\mathcal{D}_{N,\vec{\xi},v^2}(r) = \mathcal{B}_r(u\cdot w)\right] > \frac{1}{2} + \frac{\epsilon'(n)}{4}$$

where the distribution of $N = P\cdot Q$ is $FIG(1^n)$, v is uniformly drawn from $\mathbb{Z}_N^*$, r is a random n-bit string, $\vec{\xi}$ is a random vector of $2n$ elements in $[N]$, $w = w(\vec{\xi}, v^2)$ is a quadratic residue in $\mathbb{Z}_N^*$, and $u\cdot w$ is the unique quadratic residue in $\mathbb{Z}_N^*$ which satisfies $(u\cdot w)^2 = v^2 \cdot w^2 \bmod N$.

Proof: By Lemma 5.4, $\mathcal{D}$ has an $(\epsilon'(n) - n\cdot 2^{-O(n)})$-advantage in distinguishing $\mathcal{B}_r(u\cdot w)$ from a randomly chosen bit. Using an averaging argument, it is easy to see that on at least an $\frac{\epsilon'(n)}{2}$ fraction of the choices of N,v^2 and $\vec{\xi}$, algorithm $\mathcal{D}$ has an $\frac{\epsilon'(n)}{2}$-advantage in distinguishing $\mathcal{B}_r(u\cdot w)$ from a randomly chosen bit. It is then straightforward that $\mathcal{D}_{N,\vec{\xi},v^2}$ can predict the value of $\mathcal{B}_r(u\cdot w)$ with advantage $\frac{\epsilon'(n)}{4}$, as required. ■

The Factoring Algorithm: For the sake of completeness, we now turn to describe an algorithm, $\mathcal{A}$, which uses $\mathcal{D}_{N,\vec{\xi},v^2}$ as oracle and succeeds to $\epsilon''(n)$-factor (where $\epsilon''(n) = \Omega(\epsilon'(n)^2)$), this is in contradiction to Assumption 4.1, and will complete the proof.

Lemma 5.6 *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$, satisfying Equation 7. Then there exists a probabilistic polynomial-time algorithm, $\mathcal{A}$, that $\epsilon''(n)$-factors.*

Proof: On input $\langle N \rangle$, $\mathcal{A}$ is defined as follows:

1. (a) Sample v uniformly at random in $\mathbb{Z}_N^*$ and compute $v^2 \bmod N$.
   (b) Sample $\vec{\xi} = \langle \xi_1, \xi_2, \ldots, \xi_n, \xi'_1, \xi'_2, \ldots, \xi'_n \rangle$ by uniformly picking $2n$ elements in $[N]$.
2. Compute the value of $w = v^{(\Sigma_{i=0}^{n-1} \beta_i 2^{\ell(n-i)})} \bmod N$, where β_i are the coefficients (over $\mathbb{Z}$) of the polynomial $p(x) \stackrel{\text{def}}{=} \prod_{i=1}^{n}(\xi_i + x) = x^n + \sum_{i=0}^{n-1} \beta_i x^i$ (and are easily found given the ξ_i's).
3. Invoke the Goldreich-Levin reconstruction algorithm, $\mathcal{R}(1^n)$.
   (a) Whenever asked for $\mathcal{B}_{r_i}(z)$, invoke $\mathcal{D}_{N,\vec{\xi},v^2}$ on input $\langle r_i \rangle$ and give its output as an answer. (recall that $\mathcal{D}_{N,\vec{\xi},v^2}$ invokes $\mathcal{M}$ and answers its queries)
   (b) Denote by z the output of $\mathcal{R}$.
4. Compute $u = z \cdot w^{-1} \bmod N$. Given that $\mathcal{R}$ outputs the correct value (i.e. $z = u \cdot w$), then $u^2 = v^2 \bmod N$. If $u \neq \pm v \bmod N$, output $gcd(u - v, N)$ which is indeed in $\{P, Q\}$. Otherwise, output 'failed'.

The Running-Time of $\mathcal{A}$: It is clear that steps (1),(2) and (4) can be carried out efficiently by $\mathcal{A}$ (and, in addition, are independent of $\mathcal{D}_{N,\vec{\xi},v^2}$). As for step (3), assuming that $\mathcal{D}_{N,\vec{\xi},v^2}$ has non-negligible advantage in predicting $\mathcal{B}_r(u \cdot w)$ (which by Lemma 5.5 happens with non-negligible probability), we are guaranteed (by Theorem 5.2) that $\mathcal{R}$ will terminate in polynomial time.

The Success-Probability of $\mathcal{A}$: Since with probability $\frac{\epsilon'(n)}{2}$, $\mathcal{D}_{N,\vec{\xi},v^2}$ predicts the value of $\mathcal{B}_r(u \cdot w)$ with advantage $\frac{\epsilon'(n)}{4}$, then by Theorem 5.2 we have that $\mathcal{R}$ retrieves the value of $u \cdot w$ with probability at least $\Omega(\epsilon'(n)^2)$ (over the choices of N, v^2 and $\vec{\xi}$). Note that $u \cdot w$ and w are both quadratic residues in $\mathbb{Z}_N^*$, therefore u must also be a quadratic residue in $\mathbb{Z}_N^*$. Given that, the probability that u does not equal $\pm v \bmod N$ is exactly 1/2. We are finally able to conclude that $\mathcal{A}$ $\epsilon''(n)$-factors, as required. ■

This completes the proof of Theorem 5.1. ■

6 Further Research

The proof of Theorem 5.1 is tailored to the specific cryptographic primitives which are used in Construction 4.1 (i.e. the "unpredictable" function $g^{\Pi_{i=1}^{n} a_{i,x_i}}$ and the BBS generator). An interesting open problem would be to provide an alternative proof for Theorem 5.1. Such a proof might make use of more general notions and different techniques, and will hopefully shed more light on the reasons for which our construction yields a pseudorandom function. In particular, it may provide new constructions of pseudorandom functions based on more general (or more efficient) cryptographic primitives.

As we have demonstrated (in Section5.1, there exists an "unpredictable" function and a pseudo-random generator such that their composition is *not* a pseudo-random function. It should be interesting to recognize what precisely are the features of a function h_s (from an ensemble $H = \{h_s\}$) and of a pseudo-random sequence generator G that are needed in order to prove that our construction indeed yields pseudorandom functions.

Comparing to the DDH Pseudorandom Functions: As we have already mentioned, the efficiency of Construction 4.1 is comparable to the efficiency of the DDH-functions by Naor and Reingold [19]. Apart from being slightly more efficient than our functions, the DDH-functions have some additional properties:

- The simple algebraic structure of the DDH-functions implies several attractive features (e.g. zero-knowledge proof for the value of the function, function sharing and oblivious evaluation of the value of the function). In spite of the similarity between the two constructions, we do not know how to prove that similar protocols are secure in our case.
- As opposed to the proof of Theorem 5.1, the security of the DDH-functions does not decrease proportionally to the number of queries which are made by the adversary[15] (this is due to the random self-reducibility of the DDH-assumption [19]).

It is natural to consider the features of the DDH-functions as guidelines for further research regarding our functions.

7 Acknowledgments

We would like to thank Roger Fischlin for pointing out the applicability of the $\mathcal{LSB}$ reconstruction techniques of [1, 10] to our construction (as described in Section 4.4 and discussed in Footnote 14).

References

[1] W. B. Alexi, B. Chor, O. Goldreich and C. P. Schnorr, RSA and Rabin functions: certain parts are as hard as the whole, *SIAM J. Comput.*, vol. 17(2), 1988, pp. 194-209.

[2] D. Angluin, D. Lichtenstein, Provable security of cryptosystems:A survey. Tech. Rep. 288, Dept. of Computer Science, yale Univ. new Haven, Conn., 1983.

[3] E. Biham, D. Boneh and O. Reingold, Breaking generalized Diffie-Hellman modulo a composite is no easier than factoring, *Information Processing Letters*, vol. 70, 1999, pp. 83-87.

[4] D. Boneh, The decision Diffie-Hellman problem, *Proceedings of the Third Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, Vol. 1423, Springer-Verlag, 1998, pp. 48-63.

[5] L. Blum, M. Blum and M. Shub, A Simple Secure Unpredictable Pseudo-Random Number Generator, *SIAM J. Comput.*, vol. 15, 1984, pp. 364-383.

[6] M. Blum and S. Goldwasser, An Efficient Probabilistic Public-key Encryption Scheme Which Hides All Partial Information, *Advances in Cryptology - CRYPTO'84*, Lecture Notes in Computer Science, vol. 196, Springer-Verlag, 1985, pp. 289–302.

[7] M. Blum and S. Micali, How to generate cryptographically strong sequence of pseudo-random bits, *SIAM J. Comput.*, vol. 13, 1984, pp. 850-864.

[15] See [16] for a discussion of security preserving reductions (called poly-preserving in their terminology).

[8] G. Brassard, On computationally secure authentication tags requiring short secret shared keys, *Advances of Cryptology: Proceedings of Crypto-82*, D. Chaum, R.L. Rivest and A.T. Sherman, Eds. Plenum Press, New-York, 1983, pp. 79-86.

[9] R. Fischlin – Private communication.

[10] R. Fischlin and C. P. Schnorr, Stronger security proofs for RSA and Rabin bits, *Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, pp. 267-279.

[11] O. Goldreich, **Foundations of Cryptography** *Fragments of a Book*, 1995. Electronic publication: *http://www.eccc.uni-trier.de/eccc/info/ECCC-Books/eccc-books.html* (Electronic Colloquium on Computational Complexity).

[12] O. Goldreich, **Modern cryptography, probabilistic proofs and pseudo-randomness**. *Algorithms and Combinatorics*, vol. 17, Springer-Verlag, 1998.

[13] O. Goldreich, S. Goldwasser and S. Micali, How to construct random functions, *J. of the ACM.*, vol. 33, 1986, pp. 792-807.

[14] O. Goldreich and L. Levin, A hard-core predicate for all one-way functions, *Proc. 21st Ann. ACM Symp. on Theory of Computing*, 1989, pp. 25-32.

[15] J. Hastad, R. Impagliazzo, L. A. Levin and M. Luby, Construction of a pseudo-random generator from any one-way function, To appear in *SIAM J. Comput.* Preliminary versions by Impagliazzo et. al. in *21st STOC*, 1989 and Hastad in *22nd STOC*, 1990.

[16] M. Luby, **Pseudo-randomness and applications**, Princeton University Press, 1996.

[17] R. Motwani and P. Raghavan, **Randomzied Algorithms**, Cambridge Univ. Press, 1995.

[18] M. Naor and O. Reingold, Synthesizers and their application to the parallel construction of pseudo-random functions, *J. of Computer and Systems Sciences*, vol. 58 (2), April 1999, pp. 336-375.

[19] M. Naor and O. Reingold, Number-Theoretic constructions of efficient pseudo-random functions, *Proc. 38th IEEE Symp. on Foundations of Computer Science*, 1997, pp. 458-467.

[20] M. Naor and O. Reingold, From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MAC's, *Advances in Cryptology - CRYPTO '98*, Lecture Notes in Computer Science, vol. 1462, Springer-Verlag, 1998, pp. 267-282.

[21] A. M. Odlyzko, *The future of integer factorization*, RSA CyptoBytes, 2(1), 1995, pp. 5–12.

[22] M. O. Rabin, Digitalized signatures and public-key functions as intractable as factorization, Technical Report, TR-212, MIT Laboratory for Computer Science, 1979.

[23] O. Reingold, Pseudo-Random Synthesizers, Functions and Permutations, *PhD Thesis, Weizmann Institute of Science*, 1998.

[24] U.V. Vazirani and V.V. Vazirani, Efficient and Secure Pseudo-Random Number Generation, *Proc. 25th IEEE Symp. on Foundations of Computer Science*, 1984, pp. 458-463.

[25] A. C. Yao, Theory and applications of trapdoor functions, *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, 1982, pp. 80-91.