# The Discrete Logarithm Modulo a Composite Hides O(n) Bits

J. Håstad,[*]A. W. Schrift[†]and A. Shamir[†]

**Abstract**

In this paper we consider the one-way function $f_{g,N}(X) = g^X \pmod{N}$, where $N$ is a Blum integer. We prove that under the commonly assumed intractability of factoring Blum integers, all its bits are individually hard, and the lower as well as upper halves of them are simultaneously hard. As a result, $f_{g,N}$ can be used in efficient pseudo-random bit generators and multi-bit commitment schemes, where messages can be drawn according to arbitrary probability distributions.

## 1 Introduction

A function $f(x)$ is one-way if it is easy to compute but hard to invert. One-way functions have numerous cryptographic applications in public-key cryptosystems, pseudo-random bit generation, commitment schemes and so on. Several explicit constructions of one-way functions have been suggested under some plausible number-theoretic assumptions. One such candidate is the exponentiation function $f_{g,P}(X) = g^X \pmod{P}$, where $P$ is a prime and $g$

[*]Royal Institute of Technology, Stockholm, Sweden, supported by Swedish board for technical development.

[†]Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel.

is a generator of $\mathbf{Z}_P^*$ ([BM]). Its inverse is the discrete logarithm function, for which no efficient algorithms have been found. Another problem that is considered to be highly intractable is that of factoring a number which is the product of two large primes. Among the one-way functions that are based on the difficulty of factoring are the RSA / Rabin functions ([RSA], [Ra]), as well as the quadratic residuosity problem and its related root extracting function ([BBS]).

An interesting property of one-way functions is the existence of *hard bits* in the argument which cannot be computed by any family of polynomial-size Boolean circuits with *1/2+1/poly* probability of success. This notion was extensively investigated in the early 1980's, culminating in proofs that some specific bits in these number theoretic functions (usually the most significant or the least significant $O(\log n)$ bits of the $n$-bit argument) are individually hard ([BM], [ACGS], [BBS]), and that those $O(\log n)$ bits are also simultaneously hard ([LW], [ACGS], [VV]). All the subsequent efforts to extend the techniques to prove the individual or simultaneous security of $O(n)$ bits in these number theoretic functions failed.

Goldreich and Levin [GL] have shown that for every one-way function there is a logarithmic number of one-bit predicates that are hard, given the value of the function. Extending their result to prove that more bits are hard without imposing any assumptions on the one-way function is conjectured to be impossible, since a function may be one-way and still depend only on a small fraction of its bits. Explicit constructions of one-way functions for which all the bits are secure do exist, but they rely on the composition of hard bits from many one-way functions (rather than on a single application of a natural function, e.g. in the probabilistic encryption functions of [GM], [BG]).

Besides its theoretical significance, proving a one-way function to have many simultaneously hard bits can improve the efficiency of many cryptographic schemes. Very recently Impagliazzo and Naor ([IN]) have introduced an efficient pseudo-random bit generator based on the combinatorial one-way function corresponding to the subset sum problem. Their novel construction makes it possible to obtain $O(n)$ pseudo-random *output* bits from each application of the function on random inputs, but does not necessarily imply that the *input* bits of the function are individually or simultaneously hard, leaving the problem of constructing a natural function with $O(n)$ secure bits open.

In this paper we consider the well known one-way function $f_{g,N}(X) = g^X \pmod{N}$, where $N$ is a Blum integer. We prove that under the sole assumption that factoring Blum integers is difficult, all its bits are individually hard, and the lower and upper halves of them are simultaneously hard. As a result, $f_{g,N}$ can be used in efficient pseudo-random bit generators with $O(n)$-bit output per stage and in multi-bit commitment schemes, in which the messages can be drawn according to arbitrary probability distributions.

The paper is organized as follows: In section 2 we give the various definitions and assumptions used. In section 3 we deal with the individual bits security of $f_{g,N}$ and in section 4 with the simultaneous bit security. We present some applications of our enhanced security results in section 5 and discuss several extensions of our work in section 6.

## 2 Preliminaries

Let $N = P \cdot Q$, where $P$, $Q$ are distinct odd primes, and let $n$ be the binary size of $N$. Let $\mathbf{Z}_N^*$ be the multiplicative group containing the elements in $[1, N]$ that are relatively prime to $N$. The order of an element $g \in \mathbf{Z}_N^*$, $ord_N(g)$, is the smallest $c \geq 1$ such that $g^c = 1 \pmod{N}$. We denote $\max_{g \in \mathbf{Z}_N^*}\{ord_N(g)\}$ by $O_N$. Clearly:

$$O_N = \mathrm{lcm}(P-1, Q-1) \leq \frac{(P-1)(Q-1)}{2}.$$

We refer to any $g$ as a *generator* despite the fact that no $g$ can generate all the elements in $\mathbf{Z}_N^*$ for $N$ which is the product of two odd primes.

**Definition**: For a given $g$ let $G \subset \mathbf{Z}_N^*$ be the set of elements generated by it, i.e.:

$$G = \{Z | \text{there exists } X \in \mathbf{Z}_N^* \text{ s.t. } Z = g^X \pmod{N}\}$$

Note that the number of elements in $G$ equals $ord_N(g)$.

**Definition**: Fix a constant $k$. A *high order* $g$ is an element for which:

$$ord_N(g) \geq \frac{1}{n^k} \cdot (P-1)(Q-1).$$

A careful counting argument, for which we gratefully acknowledge Noga Alon, shows that a substantial fraction of the elements in $\mathbf{Z}_N^*$ have high order:

**Proposition 1**:
Let $P$ and $Q$ be randomly chosen primes of equal size, $N = P \cdot Q$ with binary size $n$, and $g$ a randomly chosen element in $\mathbf{Z}_N^*$ , then:

$$\Pr\left(ord_N(g) < \frac{1}{n^k} \cdot (P-1)(Q-1)\right) \leq O\left(\frac{1}{n^{(k-4)/3}}\right)$$

**Proof**:

$$\Pr\left(ord_N(g) < \frac{1}{n^k} \cdot (P-1)(Q-1)\right) \leq$$
$$\Pr\left(O_N < \frac{1}{n^{k'}} \cdot (P-1)(Q-1)\right) + \Pr\left(ord_N(g) < \frac{1}{n^{k-k'}} \cdot O_N\right), \text{ for every } k' \leq k.$$

We next use the following two combinatorial lemmas and by choosing $k' = (k+5)/3$ the proposition follows.

**Lemma 1.1**: For randomly chosen primes of equal size, $P$ and $Q$, let $N = P \cdot Q$ and $n = \lceil \log N \rceil$.

$$\Pr\left(O_N < \frac{1}{n^l} \cdot (P-1)(Q-1)\right) \leq O\left(\frac{1}{n^{l-3}}\right)$$

**Lemma 1.2**: Let P be a randomly chosen prime of size $m$.

$$\Pr\left(ord_P(g) < \frac{1}{m^l} \cdot (P-1)\right) \leq O\left(\frac{1}{m^{(l-1)/2}}\right)$$

$\square$

**Proof of Lemma 1.1**: $O_N = \text{lcm}(P-1, Q-1) = (P-1)(Q-1)/\gcd(P-1, Q-1)$. Hence to prove the lemma we need to show that:

$$\Pr(\gcd(P-1, Q-1) > n^l) \leq O\left(\frac{1}{n^{l-3}}\right)$$

Let $m = \lceil n/2 \rceil$ denote the binary size of $P$ and $Q$. Let $X = 2^m - 1$. We shall first examine the sum of the gcd of all pairs of $m$-bit numbers: $\sum_{X/2 \leq R_1, R_2 \leq X} \gcd(R_1, R_2)$. Obviously:

$$\sum_{X/2 \leq R_1, R_2 \leq X} \gcd(R_1, R_2) \leq \sum_{b=2}^{X} \frac{1}{b} \cdot \frac{X^2}{4} \leq \log X \cdot \frac{X^2}{4}$$

We are interested in $R_1$, $R_2$ of the form $P-1$, $Q-1$. As the density of primes in the interval $[X/2, X]$ is known to be $O(X/\log X)$ (by an extension of Heath-Brown to the Prime Numbers Theorem [GK]), the lemma follows. $\square$

**Proof of Lemma 1.2**: Let $g$ be an element in $\mathbf{Z}_P^*$. It is well known that the order of any element of a group divides the order of the group, and that the number of elements of order $d$ is exactly $\phi(d)$, where $\phi(\cdot)$ is the Euler function. Thus, for any prime $P$ of binary size $m$:

$$|\{g : ord_P(g) < \frac{1}{m^l} \cdot (P-1)\}| = \sum_{d|P-1;\ d<\frac{1}{m^l}\cdot(P-1)} \phi(d) \le \sum_{d|P-1;\ d<\frac{1}{m^l}\cdot(P-1)} d$$

For any $m$-bit number $Y$, let $F(Y)$ denote the sum of all the divisors of $Y$ that are smaller than $1/m^l \cdot Y$, i.e. $F(Y) = \sum_{d|Y, d<1/m^l\cdot Y} d$. We shall first bound the sum of $F(Y)$ over all $m$-bit numbers: $\sum_{Y=X/2}^{X} F(Y)$, with $X = 2^m - 1$. It is easy to see that any number $d < X/m^l$ is summed up as a divisor at most $1 + X/(2d)$ times. Hence:

$$\sum_{Y=X/2}^{X} F(Y) \le \sum_{d \le \frac{1}{m^l}\cdot X} \left(\frac{X}{2d} + 1\right) \cdot d = O\left(\frac{X^2}{m^l}\right)$$

Note that for any $P \in [X/2, X]$:

$$\Pr\left(ord_P(g) < \frac{1}{m^l}\cdot(P-1)\right) \le \Pr\left(F(P-1) \ge \frac{1}{m^{l'}}\cdot X\right) + \frac{1}{m^{l'}}, \quad \text{for every } l' \le l.$$

To estimate $\Pr\left(F(P-1) \ge 1/m^{l'} \cdot X\right)$ we use our bound on the sum of $F(Y)$ together with the fact that $O(X/\log X)$ of the $Y'$s in the interval are of the form $P-1$. It is easy to see that:

$$\Pr\left(F(P-1) \ge \frac{1}{m^{l'}}\cdot X\right) \le O(m^{l'-l+1})$$

By choosing $l' = (l-1)/2$, the lemma follows. $\square$

**Definition**: Let $g \in \mathbf{Z}_N^*$ . The *exponentiation modulo a composite* function is defined by:

$$f_{g,N}(X) = g^X \pmod{N}.$$

Its inverse, the *discrete logarithm modulo a composite*, is defined only for $Z \in G$ by:

$$f_{g,N}^{-1}(Z) = X,$$

for the unique $X \leq ord_N(g)$ s.t. $Z = f_{g,N}(X)$.
Note that while the values of $f_{g,N}$ range from 1 to $N$, $f_{g,N}^{-1}$ outputs only values up to $ord_N(g)$ which is strictly smaller than $N$.

Following is a list of the assumptions that are used throughout this paper. Unless otherwise mentioned, we shall assume that all these assumptions hold, even though some of our results can be derived without some of them.
**Assumption a.1**: $P$ and $Q$ are of equal size.
This assumption is commonly used in cryptography, and is believed to strengthen the intractability of factorization.
**Assumption a.2**: $P = Q = 3 \pmod 4$.
If the assumption holds, every square in $\mathbf{Z}_N^*$ has exactly one square root that is also a square. Hence, squaring is a permutation of the quadratic residues. The numbers $N = P \cdot Q$ for which both assumptions hold are called *Blum integers*.
**Assumption a.3**: $g$ is a quadratic residue.
We refer to any $g$ for which assumption a.3 holds as an *admissible generator*. Note that Proposition 1 holds even if we restrict $N$ to be a Blum integer and $g$ to be an admissible generator.
**Intractability assumption** [Y]: No family of polynomial-size Boolean circuits can factor a polynomial fraction of the Blum integers.
**Note**: All our assumptions and results use the non-uniform approach to complexity, i.e. are stated in terms of polynomial-size Boolean circuits. However, most of them can be stated, without any changes, in terms of probabilistic polynomial-time algorithms. The only case, where an adjustment is required, is in the proof of Theorem 7, as indicated there.
**Definition**: An *admissible triplet* $(g,\ N,\ Z)$ is such that:
1. $N$ is a Blum integer.
2. $g$ is an admissible generator.
3. $Z \in G$.

The collection of admissible triplets can be efficiently sampled, i.e. it is possible to pick a random admissible triplet using a polynomial amount of resources (time, random bits).

A well known result ([Ba], [Ch]) is:

**Theorem 2**:
Under the intractability assumption, the exponentiation modulo a Blum integer, $f_{g,N}(X)$, is a one-way function.
**Proof**:
We present the simple proof of this theorem as it demonstrates some of the basic techniques that are crucial for our results. We establish that it is possible to plant a short yet hard secret inside the argument of $f_{g,N}$, and use that fact extensively in the sequel.

Define $Y = g^N \pmod{N} = f_{g,N}(N)$. Let $S = f_{g,N}^{-1}(Y) = N - d \cdot ord_N(g)$, where $d$ is the largest multiple of $ord_N(g)$ for which $S$ is non-negative. Let $|S|$ denote the binary size of $S$. The following key lemma proves that $S$ is extremely small:
**Lemma 2.1**:

$$|S| \leq \left\lceil \frac{n}{2} \right\rceil + 1$$

**Proof**: It is well known that for any $g \in \mathbf{Z}_N^* : ord_N(g)|O_N$, and therefore $ord_N(g)|(P-1)(Q-1)$. Assume now that $ord_N(g) > P+Q-1 \approx 2\sqrt{N}$. In that case it is easy to see that $(P-1)(Q-1)$ is the largest multiple of $ord_N(g)$, which is still smaller than $N$: $(P-1)(Q-1) < N$, but $(P-1)(Q-1) + ord_N(g) = N - (P+Q-1) + ord_N(g) > N$. Therefore by definition: $S = N - d \cdot ord_N(g) = N - (P-1)(Q-1) = (P+Q-1)$. For $g$ such that $ord_N(g) \leq P+Q-1$, we get $S < ord_N(g) \leq P+Q-1$, which completes the proof of the lemma.

Assume that $f_{g,N}$ is not a one-way function, i.e. there exists a family $C$ of polynomial-size Boolean circuits that computes $f_{g,N}^{-1}(Z)$ successfully on a non-negligible fraction of the Blum integers $N$, the generators $g \in \mathbf{Z}_N^*$ and the elements $Z \in G$. We use $C$ to factor a non-negligible fraction of the Blum integers, thus contradicting the intractability assumption. Let $N$ be some random Blum integer and $g$ a random generator in $\mathbf{Z}_N^*$ for which $C$ computes $f_{g,N}^{-1}(Z)$ successfully on a non-negligible fraction of $Z \in G$. For such $N$ and $g$ we use $C$ to compute $S$ by applying standard randomization techniques on $Y$ (i.e. trying out sufficiently many inputs to $C$ of the form $Y \cdot g^R$ for an appropriate choice of a random $R$). Subsequently we try to factor $N$ using $S$: If $ord_N(g) > P+Q-1$ (which by Proposition 1 is likely to happen), then $S = (P+Q-1)$. Hence, by solving the two equations: $S = P+Q-1$ and $N = P \cdot Q$ we get the full factorization of $N$. $\square$

**Notations**: For a number $U$ let $u_n \ldots u_1$ denote the binary representation of

$U$, with $u_n$ being the most significant bit and $u_1$ being the least significant bit. Note that most significant bit always refers to the $n$-th bit in the binary representation, even when $U$ ranges over a smaller interval of possible values. A substring $u_k ... u_j$ of $u_n ... u_1$ ($1 \leq j < k \leq n$) will be denoted by $u_j^k$. We use the notation $f < O(\nu(n))$ for any function that vanishes faster than any polynomial, i.e. for every polynomial $poly(n)$ and $n$ large enough $f < 1/poly(n)$.

## 3 The Hard Bits of $f_{g,N}(x)$

**Definition H.1**: The $i$-th bit of the function $f_{g,N}$ is *hard* if no family of polynomial-size Boolean circuits can, given a random admissible triplet ($g$, $N$, $Z$), compute the $i$-th bit of $f_{g,N}^{-1}(Z)$ with probability of success greater than $1/2 + 1/poly(n)$, for any polynomial $poly(n)$.
Note that we use the direct definition of hardness (as in [BM]) rather than defining a bit to be hard if its approximation is as hard as computing $f_{g,N}^{-1}$ (as in [LW]).

The above well known definition of hardness is valid only for unbiased bits. However, proving the security of the $O(\log n)$ most significant bits of $f_{g,N}$ calls for a new definition of hardness for bits that are a-priori known to be biased (and therefore can be trivially predicted with probability greater that $1/2$). Let $x_i$ be the $i$-th input bit of the function $f_{g,N}$ and denote its bias towards 0 by $b(i)$. Note that only for $i \geq n - O(\log n)$ the bias is significantly greater than $1/2$, yet the definition we give is valid for any bias. In particular it implies definition H.1 for non-biased bits ($b(i) = 1/2$). Following the work in [SS1] we define:
**Definition**: The *weighted success rate* of any family $C$ of polynomial-size Boolean circuits in predicting $x_i$ is:

$$ws(C, x_i) = \frac{1}{b(i)} \cdot \Pr(C(g, N, Z) = x_i | C = 0) + \frac{1}{1 - b(i)} \cdot \Pr(C(g, N, Z) = x_i | C = 1)$$

To make the conditional probabilities well defined we must require $C$ to be *non-constant*, i.e. output 0 and 1 with probabilities greater than 0. As is explained in [SS1] this does not detract from the generality of the definition, since a constant circuit can only discover deviations from the overall bias.

**Definition H.2**: The $i$-th bit of the function $f_{g,N}$, $x_i$, is *hard* if for every family $C$ of polynomial-size Boolean circuits that is given a random admissible triplet $(g,\ N,\ Z)$:

$$ws(C, x_i) < 2 + O(\nu(n))$$

**Theorem 3**:
Under the intractability assumption, for every $1 \leq i \leq n$ the $i$-th bit of $f_{g,N}$ is hard.
**Proof**:
**Overview**:
Suppose that for a certain $i$, the $i$-th bit is not hard. For non-biased bits $(i < n - O(\log n))$ by definition H.1 there exists a polynomial-size oracle (circuit) $C:\ (g, N, Z) \to \{0, 1\}$, (where $(g,\ N,\ Z)$ is an admissible triplet) that succeeds with probability exceeding $1/2 + 1/n^c$, for some constant $c$, in predicting the $i$-th bit of $f_{g,N}^{-1}(Z)$. As in Theorem 2 let $Y = g^N \pmod{N}$. We use the oracle to factor $N$, by computing all the bits of $S = f_{g,N}^{-1}(Y)$ and following the reduction of Theorem 2. In the following we discuss the general techniques and procedures that are implemented in performing the extraction of $S$ using $C$. Dealing with the most significant (biased) bits of $f_{g,N}$, imposes several additional difficulties, that are of a less general nature. This analysis of the hardness of the biased bits is given in full in the proof of Proposition 3.4.

Intuitively, we can regard an oracle for the $i$-th bit as a one-bit window into the $i$-th position in a long unknown sequence of bits. By moving the sequence underneath the window, we can see everything in it. We therefore need a method to shift the unknown $S$ to the right and to the left, by operating on the known $Y$. We should be careful not to cause a wraparound (i.e. a reduction of the shifted $S$ modulo the unknown $ord_N(g)$), by zeroing some known bits of $S$ while operating on $Y$. The shifts to the left result essentially from squaring $Y$. We cannot perform the shifts to the right by extracting square roots of $Y$, since that cannot be done in polynomial time when the factorization of $N$ is unknown. Instead we develop a special technique by which the right shifts result from changing the base $g$ of the exponentiation function, and using the fact that squaring modulo a Blum integer is a permutation over the (randomly chosen) admissible generators.

As the oracle may err, one peek through the window in not enough. We 'collect votes' on the value of the $i$-th bit by querying the oracle on polynomially many random multiples of the original input, and use a majority vote to decide the value. To perform this randomization we have to guess an estimate of the unknown $ord_N(g)$ as an upper bound on the random choices, thus preventing the occurrence of a wraparound. Since the multiplication involves the addition of the known exponent of the random value with the unknown argument of $f_{g,N}$, we should handle with care the unknown carry into the $i$-th bit position from the addition of their least significant $i-1$ bits. We solve the problem by guessing the value of a logarithmic number of bits right to the $i$-th bit and zeroing them. A straightforward implementation of this guessing strategy for each bit position leads to an exponential algorithm, but a more careful implementation can make sure that only a polynomial number of candidates for the value of $S$ exist.

We begin the proof with a detailed description of the bit-zeroing, shifting and randomization techniques, which provide us with the necessary tools for extracting $S$. We then separate the proof into four possible cases and show:

1. The middle bits ($\lceil n/2 \rceil - O(\log n) \leq i \leq \lceil n/2 \rceil + O(\log n)$) are hard (Proposition 3.1).

2. Every bit to the right of the middle ($1 \leq i \leq \lceil n/2 \rceil - O(\log n)$) is hard (Proposition 3.2).

3. Every non-biased bit to the left of the middle ($\lceil n/2 \rceil + O(\log n) \leq i \leq n - O(\log n)$ is hard (Proposition 3.3).

4. The $O(\log n)$ most significant bits of $f_{g,N}$ are hard (Proposition 3.4).

The actual extraction of $S$ in this theorem involves two possible procedures. The simpler is the Forward-Extract procedure, where the unknown bits of $S$ are computed from the right (least significant) to the left. The more complicated is the Backward-Extract procedure, where the bits are discovered from the left to the right. We describe both procedures in detail while proving Proposition 3.1. We use a simplified version of procedure Forward-Extract in the proof of Proposition 3.2. A very careful application of procedure Backward-Extract is required for the proofs of Propositions 3.3 and 3.4.

Assumptions a.2 and a.3 on $P$, $Q$ and $g$ are required to enable performing the general right shifts technique that we develop. However, as we shall demonstrate, the use of this technique is necessary only when the oracle is located to the right of the middle. As a result Propositions 3.1, 3.3 and 3.4 can be strengthened to hold without assumptions a.2 and a.3.

We shall henceforth assume that the randomly chosen $g$ is of high order and perform our analysis accordingly. More specifically we assume that:

$$ord_N(g) \geq \frac{32}{n^{4c+4}} \cdot (P-1)(Q-1)$$

The probability that $g$ is not of high order is by Proposition 1 $O(1/n^{4c/3}))$ and since the oracle has an overall $1/n^c$ advantage in predicting correctly the $i$-th bit of $f_{g,N}$ (for any $g$), by our choice of the order it has at least a $1/2n^c$ advantage in predicting correctly the $i$-th bit for high order $g$'s. For notational simplicity let $\alpha = 4c+4$. We express most quantities in terms of $\alpha$, even when smaller quantities can be used.

In many of our procedures we need a fairly accurate guess on the order of $g$. Since all we need is a logarithmic number of significant digits we can try all possibilities and still remain polynomial time. Let $o_n ... o_1$ be the binary representation of $ord_N(g)$ and let $o_m$ be the leftmost non-zero bit of $ord_N(g)$. We assume from now on the we know the value of $m$ and of $o_{m-j}$ for $j = 1, \ldots, \lceil \alpha \log n \rceil$.

**Main Techniques**:

Let $V = f_{g,N}(U)$, for $U \leq ord_N(g)$. Note that $u_j = 0$ for $m+1 \leq j \leq n$.

**Bit-Zeroing Technique**:

The operation of zeroing a known $j$-th bit of $U$ (while operating on $V$) is denoted by $ZR_j(g, V)$. It is easy to see that:

$$ZR_j(g,V) = V \cdot g^{-u_j \cdot 2^{j-1}} \pmod{N}$$

**Shifting Techniques**:

**Shifting to the left**: Assume we are guaranteed that $u_m = 0$, and we know $u_{m-1}$. We shift the sequence of bits $u_{m-1} ... u_1$ one bit to the left, while zeroing the new $m$-th bit of the shifted $U$, by using the knowledge of $m$ and $u_{m-1}$ to transform $V$ into $(ZR_{m-1}(g,V))^2 \pmod{N}$. We cancel $u_{m-1}$ to prevent the shifted value of $U$ from becoming greater than $ord_N(g)$ and causing an overflow, which will entirely change the value of $U$ by subtracting from it $ord_N(g)$. As $ord_N(g)$ and therefore $m$ are unknown, we have to guess the value of $m$ as indicated above.

**Shifting to the right**: We can shift the sequence of bits representing $U$ one bit to the right, with the known least significant bit falling off, by transforming $V$ into $\sqrt{ZR_1(V)} \pmod{N}$, under an appropriate choice of one of the four possible square roots. However, square roots modulo $N$ cannot be efficiently computed without knowledge of the factorization of $N$, so we have to compute it in an indirect way.

Assume now that $g$ was not arbitrarily chosen, but created by squaring mod $N$ another admissible generator $g'$. Let $V' = f_{g',N}(U)$. Using the knowledge of $V'$ and of the least significant bit of $U$ we get:

$$\text{shifted } U = f_{g,N}^{-1}(ZR_1(g',\ V'))$$

As $V'$ depends on $U$, if $U$ is unknown $V'$ is also unknown. However, since we only use the technique to obtain shifts of $S = f_{g,N}^{-1}(Y)$ for $Y = f_{g,N}(N)$, it is easy to derive $Y' = f_{g',N}(S)$ via $Y' = f_{g',N}(N)$.

Observe that we only use $g'$ to calculate square roots. All the queries to the oracle are with respect to the same original $g$.

We can use this method to perform a bounded number of shifts to the right. In order to perform at most $k$ shifts to the right, we prepare in advance the sequence: $\{g_j\}_{j=0}^{k+1}$, where $g_j = g_{j-1}^2 \pmod{N}$, and use $g = g_{k+1}$ as the base of the exponentiation function. Since squaring is a permutation of the quadratic residues modulo a Blum integer $N$, a random choice of $g_0$ will produce a random admissible $g$ for any $k$.

**Randomization Technique**:
We perform the randomization by querying the oracle on $t(n) = n^{2c+3}$ inputs of the form: $(g,\ N,\ V \cdot g^R)$ for randomly chosen $n$-bit $R = r_n ... r_1$ such that $0 \leq R < ord_N(g)$. We then determine the value of $u_i$ by a majority vote.
Two main problems arise:

1. Despite our knowledge of $R$, we cannot know whether a carry from the addition of the $i-1$ least significant bits of the known $R$ and the unknown $U$ effects the $i$-th bit of the sum, and thus we cannot infer $u_i$ from the answers of the oracle for the $i$-th bit. If, on the other hand, we were guaranteed that $u_{i-1} ... u_{i-\lceil \alpha \log n \rceil} = 000...0$, we could discard the possibility of a carry except in the low probability event that $r_{i-1} ... r_{i-\lceil \alpha \log n \rceil} = 111...1$ (whose probability is at most $1/n^{\alpha}$). As the actual values of $u_{i-1} ... u_{i-\lceil \alpha \log n \rceil}$ are unknown, we try out all their (polynomial number of) possible values. For each value we act as if it was

the correct value, zero it and compute the unknown bit $u_i$ accordingly. Our procedures for the extraction of $S$ make sure that the ambiguity concerning its value remains polynomial, so that an exhaustive search can find the correct value.

2. As the exact value of $ord_N(g)$ is unknown and cannot be computed in polynomial time, we use an approximation $e = e_n ... e_1$ of $ord_N(g)$ to enable a sufficiently random choice of $R$:

$$e_j = \begin{cases} o_j & \text{for } m - \lceil \alpha \log n \rceil \leq j \leq n \\ 0 & \text{otherwise} \end{cases}$$

**Proposition 3.1**:
Under the intractability assumption, for every $i : \lceil n/2 \rceil - O(\log n) \leq i \leq \lceil n/2 \rceil + O(\log n)$ the $i$-th bit of $f_{g,N}$ is hard.
**Proof**:
Assume that for some $\lceil n/2 \rceil - O(\log n) \leq i \leq \lceil n/2 \rceil + O(\log n)$, the $i$-th bit is not hard. We extract the half-sized secret $S$ by using the following method:
**The Forward-Extract Procedure:**

1. Shift $S$ $\lceil n/2 \rceil - \lceil \alpha \log n \rceil$ bits to the left. This will not cause a wraparound by our assumption on the order of $g$. For each of the (polynomial number of) guesses of the $i - \lceil n/2 \rceil + \lceil \alpha \log n \rceil$ least significant bits of $S$, which have not passed under the oracle's location, do the following stages. (For $i < \lceil n/2 \rceil - \lceil \alpha \log n \rceil$ there is no need to guess any bits. See the proof of Proposition 3.2 for further discussion).

2. Zero these guessed bits to prevent any carry into the oracle's location during the randomization.

3. Let $Y'$ denote $Y$ after the transformations of previous stages. Let $s_j$ be the bit that is currently at the oracle's location. Deduce $s_j$ by querying the oracle on $t(n)$ random multiples $Y' \cdot g^R \pmod{N}$, for random $R < e$ (see previous discussion of randomization technique), and zero it.

4. Shift $S$ one bit back to the right, placing $s_{j+1}$ at the oracle's location. The right shifts may be performed directly since $S$ is located to the left of its original location. (See the following note.)

5. Repeat stages 3-4 until $S$ reaches its initial position, to extract all bits, $s_j$, for
   $\max\{i - \lceil n/2 \rceil + \lceil \alpha \log n \rceil, 1\} \leq j \leq i$.

6. For $i < \lceil n/2 \rceil + 1$ guess the $\lceil n/2 \rceil + 1 - i$ leftmost non-zero bits of $S$ that have not passed under the oracle's location.

7. Output the values obtained for $S$.

The correct value of $S$ among the resulting candidates from the procedure is chosen by trying to factor $N$ with each computed $S$.
**Note**: The shifts to the right in the above procedure move $S$ back at most to its initial position but not further to the right. Therefore it is possible to perform the right shifts directly without using the general shift to the right technique. An efficient implementation of these right shifts involves saving the intermediate results of the initial shifts to the left and reusing them. The same holds also for all the other procedures that are used when the oracle is located left to the middle. For this reason, Propositions 3.1, 3.3 and 3.4 can be strengthened to hold without assumptions a.2 and a.3, as we have already indicated.

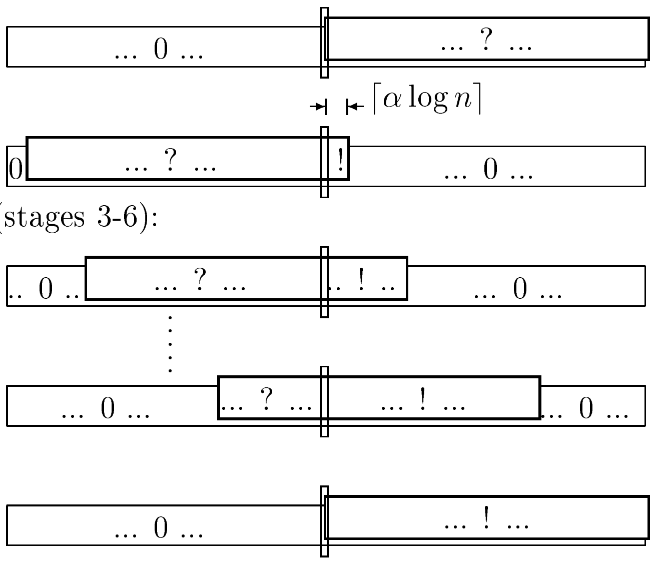
The following scheme illustrates the position of $S$ during the procedure, given an oracle for the $(\lceil n/2 \rceil + 1)$-th bit. We denote an unknown value of a bit by a question mark. All bits that are known a-priori to be zero are denoted by a zero. Bits of $S$ that were discovered or assigned values and subsequently zeroed are denoted by an exclamation mark. The oracle's location is indicated by a box.

Before the procedure begins:

After stages 1,2:

During the procedure (stages 3-6):

Finally:



It is not difficult to show that:

**Claim 1.1**: The procedure yields at most $2^{|i-\lceil n/2 \rceil+1|} \cdot 2^{\lceil \alpha \log n \rceil} = n^{O(1)}$ possible values for $S$.

This follows since each set of guesses for the least significant bits of $S$ (and at times some left non-zero bits of $S$) yields a unique value for $S$.

**Claim 1.2**: With probability greater than $\Omega(1/n^c)$ one of the output values is the correct value of $S$.

**Proof**: Consider the following two conditions:

1. For the particular choice of $N$ and $g$, the probability (over $R$) that the oracle gives the correct answer is at least $1/2 + 1/2n^c$.

2. The order of $g$ is at least $N/n^\alpha$.

We claim that whenever these two conditions are satisfied a correct value of $S$ will be derived with a probability that is exponentially close to 1. We leave this routine verification to the reader.

Now observe that since the overall probability that the oracle is correct is $1/2 + 1/n^c$ the probability (over $g$ and $N$) that 1. will happen is at least $1/2n^c$. On the other hand by Proposition 1 we know that the probability of 2. not happening is at most $1/n^{4c/3}$ and the claim follows.

**Claim 1.3**: The above procedure can be used to factor a non-negligible fraction of the Blum integers with an overwhelming probability of success, by trying random admissible $g$'s.

This follows from the two other claims.

We now present the Backward-Extract procedure. In this procedure the bits of $S$ are discovered from the most significant bit to the least significant bit. The main property of the procedure, which makes it essential for the proofs dealing with the left bits, is that at any stage of its application all the bits left to the oracle's location are known. The current procedure is applicable only when the oracle's location is logarithmically close to the middle but can be modified to work for all bits to the left of the middle (see the proofs of Propositions 3.3 and 3.4).

Let $\bar{i} = \min\{\lceil n/2 \rceil + 1, i\}$, and let $\circ$ denote concatenation. Following is a description of the Backward-Extract procedure:

**The Backward-Extract Procedure**:

1. For $i > \lceil n/2 \rceil + 1$ shift $S$ $i - \lceil n/2 \rceil - 1$ bits to the left.

2. Create a list $L_1$ of the possible candidates for $s_{\bar{i}}$ based on any guess of the $\lceil \alpha \log n \rceil$ right bits that are adjacent to it. For all possible guesses $k = 0, ..., 2^{\lceil \alpha \log n \rceil}$ of the bits $s_{\bar{i}-1} ... s_{\bar{i}-\lceil \alpha \log n \rceil}$, do:

   (a) Zero these guessed bits (with the original $Y$ transformed to $Y_k^0$)

   (b) Query the oracle on $t(n)$ random multiples $Y_k^0 \cdot g^R \pmod{N}$, for random $R < e$.

   (c) Use a majority vote to deduce the current guess for $s_{\bar{i}}$ and denote it by $CS_k^1$.

3. Let $s_{\bar{i}-j}$ be the bit that is currently evaluated. Shift $S$ $j$ bits to the left placing $s_{\bar{i}-j}$ at the oracle's location. Unlike the general shift to the left technique, perform the left shifts without zeroing the discovered bits. Let $Y^j$ denote the resulting $Y$.

4. Let $L_j$ be the list containing the candidates values for the $j$ left bits of $S$: $CS_0^j, ..., CS_{2^{\lceil \alpha \log n \rceil}}^j$. Create a new list $L_{j+1}$, which will include candidate values for $s_{\bar{i}-j}$ too. For all possible values $k = 0, ..., 2^{\lceil \alpha \log n \rceil}$ do:

(a) If $CS_k^j$ exists in the list $L_j$, then in $CS_k^j$ $s_{\bar{i}-j+1}$ was determined according to the guess $k = k_1^{\lceil \alpha \log n \rceil}$ of $s_{\bar{i}-j} ... s_{\bar{i}-j-\lceil \alpha \log n \rceil+1}$. Try a value b for $s_{\bar{i}-j-\lceil \alpha \log n \rceil}$. Let $v = k_1^{\lceil \alpha \log n \rceil - 1} \circ b$ be the current guess for $s_{\bar{i}-j-1}, ..., s_{\bar{i}-j-\lceil \alpha \log n \rceil}$.

(b) Zero $s_{\bar{i}-j-1}, ..., s_{\bar{i}-j-\lceil \alpha \log n \rceil}$ according to $v$, transforming $Y^j$ into $Y_v^j$. Deduce $s_{\bar{i}-j}$ by querying the oracle on $t(n)$ random multiples $Y_v^j \cdot g^R \pmod N$, for $R < e$.

(c) Check whether the resulting value of $s_{\bar{i}-j}$ equals $k_{\lceil \alpha \log n \rceil}$, i.e. the value that has been assigned to that bit while creating $CS_k^j$. If so, enter the value $CS_v^{j+1} = CS_k^j \circ s_{\bar{i}-j}$ into $L_{j+1}$.

(d) Repeat previous stages with $s_{\bar{i}-j-\lceil \alpha \log n \rceil} = 1 - b$.

5. Repeat stages 3-4 to extract all bits, $s_{\bar{i}-j}$, $\max\{i + \lceil \alpha \log n \rceil - \lceil n/2 \rceil - 1, 1\} \leq \bar{i} - j \leq \bar{i}$.

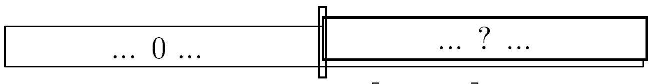
6. For $i < \lceil n/2 \rceil + 1$ guess the $\lceil n/2 \rceil + 1 - i$ leftmost non-zero bits of $S$ that have not passed under the oracle's location. For $i > \lceil n/2 \rceil + 1 - \lceil \alpha \log n \rceil$ guess the $i + \lceil \alpha \log n \rceil - \lceil n/2 \rceil - 1$ rightmost bits of $S$ that have not passed under the oracle's location.

In the above process we initially create a list $L_1$ of the possible candidates for $s_{\bar{i}}$ based on any guess of the $\lceil \alpha \log n \rceil$ right bits that are adjacent to it. As we proceed, we create a new list $L_j$ containing the candidate values for the $j$ left bits of $S$ out of the existing list $L_{j-1}$, by following stage 4. From a rough analysis of the procedure it seems possible for each candidate value $CS_k^j$ to be extended in two different ways (by concatenating both $s_{\bar{i}-j} = 0$ and $s_{\bar{i}-j} = 1$ to its right). This seemingly doubles the length of the list in every stage and causes an exponential blow-up in the number of candidate values. In fact every candidate value $CS_k^j$ is uniquely defined in terms of $j$ and $k$: We perform the left shifts without zeroing the bits that were discovered, and all the different trials for the evaluation of $s_{\bar{i}-j}$ are done on the same transformed value of $Y$, $Y^j$. Thus a certain candidate $CS_v^{j+1}$ (with $v = k_1^{\lceil \alpha \log n \rceil - 1} \circ b$) can be generated either from candidate value $CS_k^j$ with $k_{\lceil \alpha \log n \rceil} = 1$ or from a candidate value with $k_{\lceil \alpha \log n \rceil} = 0$ but not from both, since we use the guess $v$ to determine explicitly the value of $k_{\lceil \alpha \log n \rceil} = s_{\bar{i}-j}$ and thus evaluate the guess $k$. Repeating this argument we see that each string $CS_k^j$ has a unique extension to the right. It is therefore easy to see that claims 1.1-1.3 hold for the Backward-Extract procedure as well.

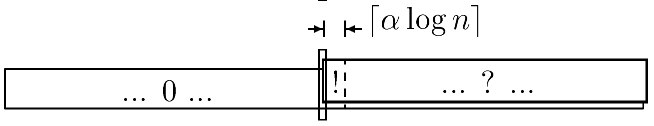
Observe that it is crucial to this uniqueness argument that we do not zero the bits that have passed under the oracle's location.

The following scheme illustrates the position of $S$ during the procedure, given an oracle for the $(\lceil n/2 \rceil + 1)$-th bit (i.e. $\bar{i} = \lceil n/2 \rceil + 1$). We denote an unknown value of a bit by a question mark. All bits that are known a-priori to be zero are denoted by a zero. Bits of $S$ that were discovered or assigned values are denoted by an exclamation mark. The oracle's location is indicated by a box.

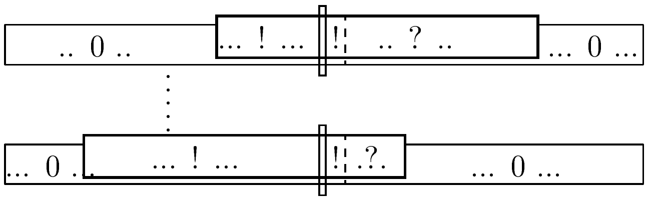
Before the procedure begins:



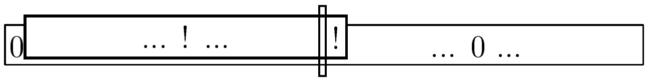
After stages 2:



During the procedure:



Finally:



$\square$

**Proposition 3.2**:
Under the intractability assumption, for every $i : 1 \leq i \leq \lceil n/2 \rceil - O(\log n)$, the $i$-th bit of $f_{g,N}$ is hard.

**Proof**
Assume that for some $1 \leq i \leq \lceil n/2 \rceil - O(\log n)$, the $i$-th bit is not hard. For all cases where $i < \lceil n/2 \rceil - \lceil \alpha \log n \rceil$ we use a simplified version of procedure Forward-Extract. (Otherwise Proposition 3.1 still holds). As we shift $S$ $i$ bits to the left (stage 1 of Forward-Extract, where $i$ substitutes the $(\lceil n/2 \rceil - \lceil \alpha \log n \rceil)$-shift), we know that all the $i-1$ least significant bits are 0.

We can therefore extract the successive bits of $S$, by repeatedly performing stages 3-4 of procedure Forward-Extract for all bits $s_j$, $1 \leq j \leq \lceil n/2 \rceil + 1$. In this simplified version we need not try out all possible values of the least significant bits of $S$, whereas in Proposition 3.1 some of the bits of $S$ remain to the oracle's right after the initial shift so that an exhaustive search cannot be avoided.

To make the right shifts possible (after the first $i$ shifts which merely move $S$ back to its original position, but leave $\lceil n/2 \rceil - i + 1$ of the bits of $S$ unknown) we must use the general right shift technique. We choose a random $g_0$, create $\{g_i\}_{i=0}^{\lceil n/2 \rceil - i + 2}$ with $g_{i+1} = g_i^2 \pmod{N}$ and use $g = g_{\lceil n/2 \rceil - i + 2}$ as the base of the exponentiation function. Since by assumption a.2 squaring is a permutation over the admissible generators, randomly choosing $g_0$ will result in a random $g$, thus ensuring that the oracle is correct for $g$ with a non-negligible probability. By the same arguments as before we get:

**Claim 2.1**: The Simplified Forward-Extract procedure yields a single value for $S$.

**Claim 2.2**: With probability at least $\Omega(1/n^c)$ the value output is the correct value of $S$.

**Claim 2.3**: The procedure can be used to factor a non-negligible fraction of the Blum integers with an overwhelming probability of success, by trying random admissible $g_0$'s. $\square$

**Proposition 3.3**:
Under the intractability assumption, for every $i : \lceil n/2 \rceil + O(\log n) \leq i \leq n - O(\log n)$ the $i$-th bit of $f_{g,N}$ is hard.

**Proof**:
Assume that for some $\lceil n/2 \rceil + O(\log n) \leq i < n - O(\log n)$, the $i$-th bit is not hard. Again we use the answers of an oracle for that $i$-th bit to derive $S$ and subsequently factor $N$. For non-extreme left bits (i.e. for $\lceil n/2 \rceil \leq i \leq (1-\varepsilon)n$, for any constant $\varepsilon$) it is possible to use the Forward-Extract procedure to extract successive $\varepsilon n$-bit blocks of $S$, as demonstrated in [SS2]. This method cannot be extended to arbitrary left $i$'s. Instead we must use the Backward-Extract procedure.

Our first presentation of the Backward-Extract procedure was for an oracle located at the middle. We performed the left shifts without zeroing the recently discovered bits and ensured (by limiting the amount of left shifts that were performed) that this did not cause an overflow. In some cases this limitation required us to guess a logarithmic number of bits of $S$. In this proposition the oracle is located in an arbitrary left location so that limiting the amount of left shifts (as done in Proposition 3.1) will leave many (too many to guess) bits of $S$ unseen by the oracle. Here we must perform the left shifts as in the general shift to the left technique and zero some bits. This makes it no longer clear that the length of the lists $L_j$, containing the candidate values for the $j$ left bits of $S$, does not grow exponentially, and hence we need a new way of trimming the list. Let $l_1, ..., l_\xi$ denote the elements of the list $L_j$ ordered from the largest $l_1$ to the smallest $l_\xi$. (Note that each number in the list is smaller than $2^j - 1$.) We shall demonstrate how to handle the list in such a way that at every stage $j$ of the Backward-Extract procedure it is still the case that $l_1 - l_\xi \leq 2^{\lceil \alpha \log n \rceil}$. To do this we show how to trim $L_j$ whenever $2^{\lceil \alpha \log n \rceil} < l_1 - l_\xi \leq 2^{\lceil \alpha \log n \rceil + 1}$.

The key idea of the trimming is to check a certain bit of another secret $S'$, which is defined by:

$$S' = \left\lceil \frac{2^{2\lceil \alpha \log n \rceil}}{l_1 - l_\xi} \right\rceil \cdot (S - l_\xi \cdot 2^{\lceil n/2 \rceil + 1 - j})$$

Define the crucial position by $cp = \lceil n/2 \rceil - j + 2\lceil \alpha \log n \rceil + 1$ (its importance will soon be evident). We now make two observations, which lead us to the following rule by which we trim $L_j$:

**Trimming Rule:** Shift $S'$ $i - cp$ bits to the left. Deduce $s'_{cp}$ by querying the oracle on $t(n)$ random multiples $g^{\text{shifted} S'} \cdot g^R \pmod{N}$, for $R < e$. If $s'_{cp} = 1$, discard the candidate value $l_\xi$ from the list $L_j$. Otherwise discard $l_1$ from that list. Repeat until you reach a list in which $l_1 - l_\xi \leq 2^{\lceil \alpha \log n \rceil}$.

**Observations:**

1. If the candidate value $l_\xi$ indeed contains the $j$ leftmost non-zero bits of $S$, then $0 \leq S' \leq 2^{\lceil \alpha \log n \rceil} \cdot 2^{\lceil n/2 \rceil + 1 - j}$. In particular $s'_{cp} = 0$, and also $s'_{cp-1} ... s'_{cp-\lceil \alpha \log n \rceil} = 0...0$.

2. If the candidate value $l_1$ indeed contains the $j$ leftmost non-zero bits of $S$, then $S' = 2^{2\lceil \alpha \log n \rceil + \lceil n/2 \rceil + 1 - j} + \delta$ where $0 \leq \delta \leq 2^{\lceil \alpha \log n \rceil} \cdot 2^{\lceil n/2 \rceil + 1 - j}$. Therefore: $s'_{cp} = 1$, $s'_q = 0$ for $q > cp$, and also $s'_{cp-1} ... s'_{cp-\lceil \alpha \log n \rceil} = 0...0$.

These two observations together with a standard sampling argument imply that with high probability we will never discard a correct value and hence by similar arguments as before we can prove:

**Claim 3.1**: Procedure Backward-Extract combined with the Trimming Rule yields at most $2^{\lceil \alpha \log n \rceil}$ possible values for $S$.

**Claim 3.2**: With probability greater than $\Omega(1/n^c)$ one of the output values is the correct value of $S$.

**Claim 3.3**: Procedure Backward-Extract combined with the Trimming Rule can be used to factor a non-negligible fraction of the Blum integers with an overwhelming probability of success, by trying random admissible $g$'s. $\square$

**Proposition 3.4**:
Under the intractability assumption, the $O(\log n)$ most significant bits of $f_{g,N}$ are hard.

**Proof**:
First let us observe that if the bias of the bit in question is at most $\frac{1}{2}n^{-c}$ (i.e. the probability of the bit being 1 is in the interval $[\frac{1-n^{-c}}{2}, \frac{1+n^{-c}}{2}]$) then the methods of Proposition 3.3 will work. (We leave this verification to the reader). Thus we will assume that the bias is at least $\frac{1}{2}n^{-c}$ and in particular $m - i \leq c \log n + 1$, where $m$ is the location of the leftmost non-zero bit in $ord_N(g)$. Also note that when the bias of the bit is exponentially close to 1 (which is the case for $i > m$), this bit is trivially hard by definition H.2.

Assume that for some $m - c \log n - 1 \leq i \leq m$ the $i$-th bit is not hard by definition H.2. We will use the same outline of the reconstruction of $S$ as in Proposition 3.3, i.e. we will use the oracle to determine if the shifted $S'$ is a number of the form $\delta$ or $2^{i-1} + \delta$ where $0 < \delta \leq 2^{i-1-\lceil \alpha \log n \rceil}$ (if it is of neither form we do not care what happens). Problems arise since it is no longer true that asking the oracle questions about numbers of the form $R +$ *shifted* $S'$

for a randomly chosen $R$ between 0 and $e$ will allow us to distinguish the two cases.

By [SS1] the fact that there is an oracle for which the weighted success rate is significantly greater than 2 (proving the bit to be weak by definition H.2) implies that there is another oracle for which the probabilities of correct 1-answers and of erroneous 1-answers significantly differ. For this oracle $C$ there exists some constant $c$ such that:

$$|\Pr(C = 1|x_i = 1) - \Pr(C = 1|x_i = 0)| \geq \frac{1}{n^c}$$

Let $I$ be the entire interval $[0, e]$ and for any interval $J$ let $P_1(J)$ the fraction of 1-answers the oracle gives on $J$. Let $J + a$ denote the interval $J$ shifted $a$. For natural reasons we will be working modulo $e$. We will need the following key lemma:

**Lemma 4.1:** If we know an interval $J$ of length at least $d$ such that $|P_1(J) - P_1(J + 2^{i-1})| \geq 20d^{-1}2^{i-1-\lceil\alpha \log n\rceil}$ then we can recover $S$.

**Proof:** Assume without loss of generality that $P_1(J) - P_1(J + 2^{i-1}) \geq 20d^{-1}2^{i-1-\lceil\alpha \log n\rceil}$. We will use the same procedure as in Proposition 3.3. The only difference is that instead of choosing a random $R < e$ we will choose a random $R$ in $J$. We need only to check that the oracle will be able to distinguish the two extreme cases. If $l_1$ contains the correct left-most non-zero bits of $S$ then the shifted $S'$ is of the form $2^{i-1} + \delta$ where $0 \leq \delta \leq 2^{i-1-\lceil\alpha \log n\rceil}$. Thus when we choose $R$ from $J$, $R +$ *shifted* $S'$ lies in an interval whose symmetric difference with $J + 2^{i-1}$ is of size at most $2^{i-\lceil\alpha \log n\rceil}$. Thus the expected fraction of 1-answers in this case is at most $P_1(J + 2^{i-1}) + \frac{1}{d}2^{i-\lceil\alpha \log n\rceil}$. Similarly we get that if $l_\xi$ contains the correct value for the left bits of $S$ then we will get the answer 1 with probability at least $P_1(J) - \frac{1}{d}2^{i-\lceil\alpha \log n\rceil}$. By assumption we can tell these two cases apart if we sample $n \cdot (P_1(J) - P_1(J + 2^{i-1}))^{-2}$ times. □

The proof will in fact proceed by identifying a polynomial number of intervals such that Lemma 4.1 will be true for one of these intervals. This interval can then be identified by sampling and then used in the Trimming Rule.

Divide $I$ into $K = \lfloor\frac{e}{2^{i-1}}\rfloor + 1$ intervals $I_1, I_2 \ldots I_K$ where $I_j = [(j - 1)2^{i-1}, j2^{i-1}]$ for $j < K$ and $I_K = [(K - 1)2^{i-1}, e]$. We have two cases:

1. There is a $j < K-1$ such that $|P_1(I_j) - P_1(I_{j+1})| \geq \frac{n^{-c}}{20K}$.

2. There is no such $j$.

In case 1 we are done by Lemma 4.1 (since $K$ is only polynomially large) and we only have to worry about case 2. The failure of 1 implies that there is a $q$ such that for all $j < K$, $|P_1(I_j) - q| \leq \frac{n^{-c}}{20}$. To get the stated difference in the overall behavior of the oracle we need $|P_1(I_K) - q| \geq \frac{4 \cdot 2^{i-1}}{5|I_K|n^c}$ (the sign depending on the parity of $K$). Observe that this implies $|I_K| \geq \frac{4 \cdot 2^{i-1}}{5n^c}$. Now consider the intervals $I_K^j = I_K + j2^{i-1}$ for $0 \leq j \leq T$, where $T = \lceil \frac{en^{2c}}{|I_K|} \rceil$. We shall now prove that these intervals cover $I$ very nicely. In particular $I_K^{2(K-1)}$ has a common border with $I_K^{K-1}$ which on its other side has $I_K^0 = I_K$: Let $f_a$ be the number of intervals of the form $I_K^j$, $0 \leq j \leq T$ that a point $a$ belongs to.
**Lemma 4.2:** For any two points $a_1$ and $a_2$ we have $|f_{a_1} - f_{a_2}| \leq 2$.
**Proof**: Since we are working modulo $e$ we are basically on the circle and we have no problems with the border. The size of the circle is $e$ which by definition equals $(K-1)2^{i-1} + |I_K|$. We establish the lemma by using the following two claims:
**Claim 4.3:** If $a_1 \equiv a_2 \bmod 2^{i-1}$ then $|f_{a_1} - f_{a_2}| \leq 1$.
**Claim 4.4:** If $a_2 < a_1 < a_2 + 2^{i-1}$ then $f_{a_1} - f_{a_2} \leq 1$.
The lemma clearly follows from these two claims.

To see the first claim assume that $a_1 = a_2 + k2^{i-1}$ where $0 < k < K-1$. Then it is true that $a_2 \in I_K^{j_0}$ iff $a_1 \in I_K^{j_0+k}$ and since it is not hard to see that neither of the points is in the intermediate intervals the claim follows.

For the second claim, suppose that $a_1 \in I_K^{j_0}$. Then we claim that $a_2 \in I_K^{j_0+k}$ for some $k$ of the form $l(K-1)$ and that $a_1 \notin I_K^j$ for $j_0 < j < j_0 + k$. This is clearly sufficient to establish the original claim. To see this later claim, observe that $I_K^{j_0+K-1}$ has its right endpoint coinciding with the left endpoint of $I_K^{j_0}$ and in general $I_K^{j_0+l(K-1)}$ has its right endpoint coinciding with the left endpoint of $I_K^{j_0+(l-1)(K-1)}$. Thus the first $l$ for which the left endpoint of $I_K^{j_0+l(K-1)}$ is to the left of $a_2$ is the desired interval. To see that $a_1$ does not belong to any of the intermediate intervals, observe that the only $I_K^j$ to which it could belong are of the form $j = j_0 + l(K-1) + 1$. However if $a_1$ belongs to $I_K^j$ then the left endpoint of $I_K^{j-1}$ is already to the left of $a_2$ and hence we have established the claim. $\square$

Examining the new division of $I$ into $T$ intervals $I_K^j$, that we have defined, we again have two cases:

1. There is a $j$ such that $j < T$ such that $|P_1(I_K^j) - P_1(I_K^{j+1})| \geq \frac{n^{-c}}{20T}$.
2. There is no such $j$.

In case 1 we are again done by Lemma 4.1 (T is large but still polynomially bounded) and all that remains is to prove that 2 cannot happen. The fact that no $j$ satisfying 1 exists implies that $|P_1(I_K^j) - P_1(I_K)| \leq n^{-c}/20$ for all $j$. Now consider $P_1(I)$: It is a weighted average of the $P_1(I_j)$ and thus $|(P_1(I) - q| \leq \frac{1}{2}|P_1(I_K) - q| + n^{-c}/20$ (using the assumption that $|I_K| \leq \frac{1}{2}|I|$). On the other hand by Lemma 4.2 if we pick a random $j$ and then pick a random point from $I_K^j$ then each point is picked with a probability that is within a factor $(1 + \frac{2}{n^{2c}})$ of the probability it would have been picked by the uniform distribution. This fact implies

$$|P_1(I) - \frac{1}{T}\sum_{j=0}^{T-1} P_1(I_K^j)| \leq \frac{2}{n^{2c}}$$

which together with the assumption that we are in case 2 implies

$$|P_1(I) - q| \geq |P_1(I_K) - q| - \frac{2}{n^{2c}} - \frac{1}{20n^c}$$

and we have reached a contradiction which completes the proof of Proposition 3.4. $\square$

## 4 The Simultaneously Hard Bits of $f_{g.N}$

In the following section we define the strong notion of simultaneous security, which states that it is computationally hard to succeed in computing any information whatsoever about groups of bits of $f_{g,N}$. We then show that $f_{g,N}$ is indeed secure in that sense.
**Definition**: $p_j^k : [1, N] \to \{0,1\}^{k-j+1}$ is the function $p_j^k(U) = u_k ... u_j$, with $k > j$.
**Definition H.3**: The bits of $f_{g,N}$ at locations $j \leq i \leq k$ are *simultaneously hard*, if $(p_j^k(f_{g,N}^{-1}(Z)),\ Z)$ is polynomially indistinguishable from $(r_j^k,\ Z)$ for randomly chosen admissible $(g,\ N,\ Z)$ and a random $R = r_1^n$.

**Definition H.4**: A non-biased $i$-th bit, $j \leq i \leq k$, of the function $f_{g,N}$ is *relatively hard to the right* (*to the left*) if no family of polynomial-size Boolean circuits can, given a random admissible triplet ($g$, $N$, $Z$) and in addition the $i-k$ ($j-i$ respectively) bits of $f_{g,N}^{-1}(Z)$ to its right (left), compute the $i$-th bit of $f_{g,N}^{-1}(Z)$ with probability of success greater than $1/2 + 1/poly(n)$, for any polynomial $poly(n)$.

**Definition H.5**: The $i$-th bit, $j \leq i \leq k$, of the function $f_{g,N}$, $x_i$, is *relatively hard to the right* (*to the left*) if for every family $C$ of polynomial-size Boolean circuits which is given a random admissible triplet ($g$, $N$, $Z$) and in addition the $i-k$ ($j-i$ respectively) bits of $f_{g,N}^{-1}(Z)$ to its right (left): $ws(C, x_i) < 2 + O(\nu(n))$.

**Proposition 4**:

The following conditions are equivalent:

1. The bits of $f_{g,N}$ at locations $j \leq i \leq k$ are simultaneously hard.
2. Each bit $j \leq i \leq k$ of $f_{g,N}^{-1}(Z)$ is relatively hard to the right.
3. Each bit $j \leq i \leq k$ of $f_{g,N}^{-1}(Z)$ is relatively hard to the left.

**Proof**:

The proof of this equivalence for non-biased bits is basically the well known proof of the universality of the next bit test [Y]. Let us give an outline of this proof. Assume that the bits are not simultaneously hard. Then, in particular there exists a distinguisher, $D$, for which (w.l.o.g.):

$$\Pr(D(p_j^k(f_{g,N}^{-1}(Z)), g, N, Z) = 1) - \Pr(D(r_j^k, g, N, Z) = 1) \geq \varepsilon,$$

for some non-negligible $\varepsilon$, with the probability taken over the random choices of $g$, $N$, $Z$ and $r_j^k$. Let:

$$p_i = \Pr(D(p_j^{j+i-1}(f_{g,N}^{-1}(Z)) \circ r_{j+i}^k, g, N, Z) = 1)$$

$$p^i = \Pr(D(r_j^{k-i} \circ p_{k-i+1}^k(f_{g,N}^{-1}(Z)), g, N, Z) = 1)$$

Where strings of negative length are null.

Clearly:

$$p_{k-j+1} = p^{k-j+1} = \Pr(D(p_j^k(f_{g,N}^{-1}(Z)), g, N, Z) = 1)$$

Also:

$$p_0 = p^0 = \Pr(D(r_j^k, g, N, Z) = 1)$$

By the pigeonhole principle there exist $i_1$ and $i_2$ for which $p_{i_1} - p_{i_1-1} \geq \varepsilon/n$ and $p^{i_2} - p^{i_2-1} \geq \varepsilon/n$. For each case we construct an oracle which proves the corresponding bit to be relatively weak either to the right or to the left. The well known technique of constructing these oracles is explained in [BH] and is therefore omitted.

For biased bits, where definition H.5 should be used, the proof of this equivalence is given in [SS1] and is also omitted. $\square$

**Theorem 5**:
Under the intractability assumption, the $\lceil n/2 \rceil + O(\log n)$ right hand bits of $f_{g,N}$ are simultaneously hard.

**Theorem 6**:
Under the intractability assumption, the $\lceil n/2 \rceil + O(\log n)$ left hand bits of $f_{g,N}$ are simultaneously hard.

**Proof of Theorem 5**:
It suffices to show that every right hand bit of $f_{g,N}$ is relatively hard to the right by definition H.4. In general, even if each bit is individually hard, it does not immediately imply the simultaneous hardness of all bits: In order to use an oracle for a relatively weak to the right $i$-th bit, all the $i-1$ least significant bits of the unknown value must be supplied too, a very hard task in general. However, careful analysis of the Forward-Extract procedure shows that such a task is possible.

Let $X = f_{g,N}^{-1}(Z)$. Assume that the theorem is false, i.e. for some $1 \leq i \leq \lceil n/2 \rceil + O(\log n)$ there exists an oracle $C(g,\ N,\ Z,\ x_1^{i-1})$ (for admissible triplets) that succeeds in predicting $x_i$ with probability $1/2+\varepsilon$, for some non-negligible $\varepsilon$. We extract the bits of $S = f_{g,N}^{-1}(Y)$, where $Y = g^N \pmod{N}$ using procedure Forward-Extract in exactly the same way as in Theorem 3 (either directly or in its simplified version, according to the location of $i$). The only difference is in the queries to the oracle, where we have to supply the $i-1$ least significant bits of the argument. By examining both versions, it is easy to see that after the initial shift to the left and for each subsequent right shift and bit-zeroing of $S$ (corresponding to a certain transformed value $Y'$ of $Y$) the $i-1$ least significant bits of $f_{g,N}^{-1}(Y')$ are zero. Therefore the $i-1$ least significant bits of $Y' \cdot g^R$ are the known bits of $R$, $r_1^{i-1}$, which can be given to the oracle. $\square$

**Proof of Theorem 6**:
We first deal with the non-biased left hand bit of $f_{g,N}$, (up to the $O(\log n)$ most significant bits). As in Theorem 5, it suffices to show that every non-biased left hand bit of $f_{g,N}$ is relatively hard to the left by definition H.4. Let $X = f_{g,N}^{-1}(Z)$. Assume that the theorem is false, i.e. for some $\lceil n/2 \rceil - O(\log n) \leq i \leq n - O(\log n)$ there exists an oracle $C(g,\ N,\ Z,\ x_{i+1}^n)$ (for admissible triplets) that succeeds in predicting $x_i$ with probability $1/2+\varepsilon$, for some non-negligible $\varepsilon$. By examining the proof of Proposition 3.3 it seems plausible to use the Backward-Extract procedure combined with the Trimming Rule, since that ensures that before the randomization is performed all the bits left to the oracle's location are zero. Therefore it seems that the left bits that should be supplied to the oracle in every query are simply the left bits of the exponent of the random multiplier. Unfortunately that is not true! During the randomization we ensure that with high probability no carry reaches the oracle's location. However there may be a carry from the $i$-th bit into the $(i+1)$-th bit (which is given as input to the oracle), and its existence depends on the value of the unknown bit of $S$ (or $S'$ in the Trimming Rule).

We solve this problem by performing the randomization with random values $R < e$ such that $r_i = 0$. In that case we are indeed guaranteed that in every query the bits left of the oracle's location are those of the random $R$. However now we cannot determine the value of the $i$-th bit simply by taking a majority vote, since it might not be the case that the oracle has any advantage over $1/2$ in correctly predicting the $i$-th bit (based on the bits to its left) on half of the possible values as determined by our randomization. Instead we use the fact that if the oracle has probability significantly greater than $1/2$ of correctly predicting the $i$-th bit (or in other words its probability of correct predictions is significantly greater than that of erroneous predictions) then its probability of correct 1-answers significantly differs from that that of erroneous 1-answers:

$$\Pr(C(g,N,Z,x_{i+1}^n) = 1 | x_i = 1) - \Pr(C(g,N,Z,x_{i+1}^n) = 1 | x_i = 0) \geq 2\varepsilon$$

It is possible to perform a-priori tests on $C$ (using inputs with known $x_i^n$) to determine the corresponding two probabilities. Thus instead of using the answers of the oracle to perform a majority vote, we estimate the relative frequency of the 1-answers with accuracy greater than $\varepsilon$. (It is again easy to see that $t(n)$ queries are sufficient). By that we can derive the unknown value of $x_i$.

Note that, as we have already observed, when performing the randomization for the Trimming Rule if none of the extreme values in the list ($l_1$ and $l_\xi$) are the correct candidates, then we are not guaranteed that the $\lceil \alpha \log n \rceil$ bits right to $s'_{cp}$ are 0 and a carry may reach the oracle's location. We may therefore get that the frequency of 1-answers is altogether different from the two a-priori measured probabilities. In that case both values can be discarded from the list.

To extend the proof to the biased bits (using definition H.5) we have to be slightly careful. Using the notations of the proof of Proposition 3.4 we argue as follows:

If the oracle behaves significantly differently on $I_{2j+1}$ and $I_{2j+2}$ for any $j$ when the correct bits for the left $n-i$ positions are given to the oracle (they are basically $j$ with some zeroes in front) we are done by the same reasoning as in Lemma 4.1 (there are only polynomially many possibilities so we can try them all). The only time we can get into trouble is when $K$ is even and most of the oracles advantage is when the left $n-i$ bits take their maximal value. Now create a "new" oracle $C'$ by always feeding this maximal value of the $n-i$ bits to our original oracle (no matter if they are correct or not). Note that by assumption $P'_1(I_K)$ and $P'_1(I_{K-1})$ are substantially different.

Looking more closely at the proof of Proposition 3.4 one can make the following statement (with substantially different interpreted as "at least $1/poly(n)$ for a suitable polynomial $poly(n)$".):

For any oracle $C$ such that $P_1(I_{K-1})$ and $P_1(I_K)$ are substantially different one of the following statements is true:

1. There is a $j < K$ such that $P_1(I_{j-1})$ and $P_1(I_j)$ are substantially different.

2. There is a $j < T$ such that $P_1(I_K^{j-1})$ and $P_1(I_K^j)$ are substantially different.

Using $C' = C$ in this statement makes it straightforward to prove the theorem also in this case. $\square$

# 5 Applications

## 5.1 Commitment Schemes

Several cryptographic schemes require a party to commit to a certain message without revealing any information on the content of the message. The message is drawn out of an arbitrary collection, which may be very sparse. Most known commitment schemes are designed to hide a single bit. Multi-bit commitment improves the efficiency of existing protocols as presented in [KMO]. Recently Naor has presented a multi-bit commitment scheme [Na] using any pseudo-random bit generator. We construct a different scheme that uses $f_{g,N}$ directly.

The simultaneous security of the $\lceil n/2 \rceil$ right hand bits of $f_{g,N}$ implies that $f_{g,N}$ hides $\lceil n/2 \rceil$ uniformly distributed bits. To use $f_{g,N}$ in a multi-bit commitment scheme, it should be proven that $f_{g,N}$ hides $O(n)$ arbitrarily distributed bits in a polynomially secure manner. We now formally define the notion of simultaneous security with respect to non-uniform probability distributions, prove that most of our results still hold under these distributions and construct a simple multi-bit commitment scheme accordingly.

**Definition**: $NU(g,\ N,\ Z)$ denotes any probability distribution function of admissible triplets in which:

1. $g$ and $N$ are uniformly distributed.
2. Let $X = f_{g,N}^{-1}(Z)$. The distribution of $Z$ is induced by any probability distribution $P(X)$ in which:
   (a) $x_{\lceil n/2 \rceil} \ldots x_1$ are arbitrarily distributed, and
   (b) $x_n \ldots x_{\lceil n/2 \rceil + 1}$ are uniformly distributed (in the range determined by $X \le ord_N(g)$).

**Definition H.6**: The $i$-th bit, $1 \le i \le \lceil n/2 \rceil$, of the function $f_{g,N}$ is *non-uniformly hard* (NU-hard) if no family of polynomial-size Boolean circuits can, given a random $NU$-distributed admissible triplet $(g,\ N,\ Z)$, compute the $i$-th bit of $f_{g,N}^{-1}(Z)$ with probability of success greater than $1/2 + 1/poly(n)$, for any polynomial $poly(n)$.

**Definition H.7**: The $k$ right most bits of $f_{g,N}$ are *simultaneously NU-hard*, if $(p_1^k(f_{g,N}^{-1}(Z)),\ Z)$ is polynomially indistinguishable from $(x_1^k,\ Z)$, for any $NU$-distributed admissible $(g,\ N,\ Z)$ and $P$-distributed $X = x_1^n$.

**Theorem 7**:
Under the intractability assumption, the $\lceil n/2 \rceil$ right hand bits of $f_{g,N}$ are simultaneously $NU$-hard. In particular for every $1 \leq i \leq \lceil n/2 \rceil$ the $i$-th bit of $f_{g,N}$ is $NU$-hard.
**Proof**:
The proof of the theorem is essentially a non-uniform version of the proof of Theorem 5. If the theorem is false then in particular there exists a certain assignment, $A$, for the $\lceil n/2 \rceil$ right hand bits of $f_{g,N}$ such that $(A,\ f_{g,N}(r^n_{\lceil n/2 \rceil+1} \circ A))$ is polynomially distinguishable from $(A,\ Z)$, where $(g,\ N,\ Z)$ is a $NU$-distributed admissible triplet, and $R$ is a randomly chosen string, s.t. $r^n_{\lceil n/2 \rceil+1} \circ A < ord_N(g)$. As we shall prove Theorem 5 can be strengthen to show that for any specific $\lceil n/2 \rceil$-bit message, $A$, $(A,\ f_{g,N}(r^n_{\lceil n/2 \rceil+1} \circ A))$ is polynomially indistinguishable from $(A,\ Z)$ for uniformly distributed admissible $(g,\ N,\ Z)$ and a random $R$ which is defined as above. Exploiting once again the fact that we work in the non-uniform complexity model leads to the conclusion that the same holds when the admissible $(g,\ N,\ Z)$ is $NU$-distributed.

To prove the strengthened version of Theorem 5, assume that there exists an assignment, $A$, for which $(A,\ f_{g,N}(r^n_{\lceil n/2 \rceil+1} \circ A))$ is polynomially distinguishable from $(A,\ Z)$, for the above defined arguments. Using the hybrid technique as in Proposition 4 it is possible to show that if this distinguishability holds then there exists a certain $1 \leq i \leq \lceil n/2 \rceil$ such that $(A,\ f_{g,N}(r^n_{i+1} \circ a^i_1))$ is polynomially distinguishable from $(A,\ f_{g,N}(r^n_i \circ a^{i-1}_1))$, where $a^i_1$ denotes the $i$ rightmost bit of $A$. In other words there exists a distinguisher $D$ such that (w.l.o.g.):

$$\Pr\Big(D(g,N,A,f_{g,N}(r^n_{i+1} \circ a^i_1) = 1\Big) - \Pr\Big(D(g,N,A,f_{g,N}(r^n_i \circ a^{i-1}_1) = 1\Big) \geq \varepsilon$$

for some non-negligible $\varepsilon$.

Using $D$ it is possible to construct an oracle $C(g,N,A,f_{g,N}(r^n_i \circ a^{i-1}_1))$ that predicts the $i$-th bit of the random (and therefore unknown) $R$ with probability greater than $1/2 + \varepsilon$ [BH]. We have used such a construction in Proposition 4. We complete the proof by using the oracle $C$ to extract the bits of $S = f^{-1}_{g,N}(Y)$ with $Y = f_{g,N}(N)$. For that we use the Forward-Extract procedure directly or in its simplified version (according to $i$, as in Propositions 3.1 and 3.2) with special attention to combining $A$ in the queries to the oracle. Both versions ensure that after the initial shift and guess all the bits right to the $i$-th bit in the argument of the transformed $Y$ (denoted $Y'$) are 0. The oracle requires, however, those bits to be the right bits of $A$. Our queries to the oracle will therefore be: $(g,\ N,\ A,\ Y' \cdot g^{r_i^n} \cdot g^{a_1^{i-1}} \pmod{N})$, for random $R < e$. It is easy to see that indeed this procedure can be used to factor a non-negligible fraction of the Blum integers with an overwhelming probability of success, which by the intractability assumption leads to the desired contradiction.

Note that unlike previous proofs, this proof relies on the use of the non-uniform complexity model. However even this theorem can be proven in the uniform complexity model under the additional assumption that the distribution $P(X)$ (and therefore $NU(g,\ N,\ Z)$) is polynomially samplable (as in [ILL]). Note also that under the appropriate definition of $NU(g,\ N,\ Z)$, Theorem 7 is the exact analogue of Theorem 5 (with $\lceil n/2 \rceil + O(\log n)$ simultaneously $NU$-hard bits). We refrained from giving this version for notational simplicity. $\square$

By Theorem 7 it is possible to commit to a $\lceil n/2 \rceil$-bit value $M$ by choosing randomly $N$ and $g$, picking a uniformly distributed $R$ s.t. $r^n_{\lceil n/2 \rceil+1} \circ M < ord_N(g)$ and sending $Z = f_{g,N}(r^n_{\lceil n/2 \rceil+1} \circ M)$, where $\circ$ denotes concatenation. In particular the theorem implies that the existence of even a single pair of messages (chosen by the opponent) whose committed values can be efficiently distinguished will lead to the factorization of $N$:

**Corollary 7.1**:
Let $M_0,\ M_1 \in \{0,1\}^{\lceil n/2 \rceil}$, be any pair of $\lceil n/2 \rceil$-bit messages. Let $Z_j = f_{g,N}(r^n_{\lceil n/2 \rceil+1} \circ M_j)$, $j = 0, 1$, with $R$ a uniformly distributed string such that $r^n_{\lceil n/2 \rceil+1} \circ M_j < ord_N(g)$. Then, $(M_j,\ Z_j)$ and $(M_j,\ Z_{1-j})$ are polynomially indistinguishable.

Note that the conditions of practical commitment schemes are somewhat different from the underlining conditions of our previous proofs: In these proofs we can try out various values for $g$ and ensure that at least one of them is of high order and that at least one of our guesses for its order is correct. However when we use $f_{g,N}$ in practice, we must guarantee that a randomly chosen generator $g$ has high order (which happens with high probability), if we are to count on the security of $f_{g,N}$. Furthermore it is necessary not only to verify that $g$ is of high order, but to know the exact order of the generator (to ensure and prove that $R$ has been chosen correctly). Recall that $ord_N(g)$ divides $(P-1)(Q-1)$. Thus in practice the factorization of

$(P-1)(Q-1)$ must be known to the party that chooses the commitment scheme, by carefully choosing the primes.
**Definition** [BM]: A prime $P$ of size $n$ is *hard* if $P = tP' + 1$, where $P'$ is a prime and $1 < t < poly(n)$.

Since hard primes have an asymptotically polynomial density among the integers of the sequence $tP' + 1$ [BM] hard primes can be found efficiently. The commitment protocol will be performed in practice using a Blum integer which is the product of randomly chosen hard primes, and its security will rely on a somewhat different assumption, namely that no family of polynomial-size Boolean circuits can factor a polynomial fraction of the Blum integers that are the product of hard primes. This might at first sound like a stronger assumption, but in fact it is weaker, since if we can factor a non-negligible fraction of Blum integers which are the product of hard primes then we can factor a non-negligible fraction of all Blum integers, while the converse is not clearly true.

## 5.2 Pseudo-Random Bit Generation

Any one-way function can be used for the construction of a pseudo-random bit generator, due to the recent results of [ILL] and [H]. However, the general techniques are very inefficient. The simple construction of [BM] is inapplicable to $f_{g,N}$, since for composite $N$ it is not one to one. $f_{g,N}$ is also not regular (i.e. not every possible value has the same number of preimages), hence even the (inefficient) construction of [GKL] cannot be used. We are interested in an efficient construction, using the simultaneous security of $\lceil n/2 \rceil$ bits of $f_{g,N}$ to output as many bits as possible in every stage of the generation.

Using the Leftover Hash Lemma, presented in [ILL] and [IZ], we give a construction of an extender $E : \{0,1\}^{3n} \to \{0,1\}^{3.5n - O(\log^2 n)}$. The pseudo-random bit generation is achieved through repeated applications of the extender to a random seed (as demonstrated in [BH]).

Let $N = P \cdot Q$ be a Blum integer of size $n$ and let $g$ be an admissible high order generator. Let $n - O(\log n) \le m \le n - 2$ be an integer such that $2^m \le ord_N(g) < 2^{m+1}$. As before, hard primes must be used to find $m$ and $ord_N(g)$ in practice. Let $H_{n,t}$ be a family of universal hash functions, where $t = m - \log^2 n$. It is well known that $2n$ bits suffice to define a unique function $h \in H_{n,t}$ (see for example [IZ], where some simple constructions are demonstrated). Let $h$ be a randomly chosen function in $H_{n,t}$ and let $X$

be a random $n$-bit string. Let $x_1^{\lceil n/2 \rceil}$ denote the $\lceil n/2 \rceil$ right hand bits of $X(\mathrm{mod} ord_N(g))$ and let $\circ$ denote concatenation. *The extender* $E$ is:

$$E(h \circ X) = h \circ h\left(f_{g,N}(X)\right) \circ x_1^{\lceil n/2 \rceil}.$$

**Note**: The fact that $O(n)$ bits of $f_{g,N}$ are simultaneously secure and not just $O(\log n)$ is crucial for the construction of $E$. Applying the hash function causes a $\log^2 n$-bit loss in the length of $E$'s output. The final $O(n)$ extension is possible only because of the many simultaneously secure bits, which more than compensate for this loss.
**Theorem 8**:
$E$ is a perfect extender.

**Proof**:
The proof is a direct result of the following two lemmas.
**Lemma 8.1**: For randomly chosen $h \circ X$, $h \circ h\,(f_{g,N}(X))$ is polynomially indistinguishable from a randomly chosen $(2n+t)$-bit string.
**Proof**: The hash functions are defined on the set $G$, of elements in $\mathbf{Z}_N^*$ that can be expressed as powers of $g$. In our construction the distribution of the elements in $G$ is induced by a uniform probability distribution of $X$. Clearly, by definition of $ord_N(g)$:

$$\min_{Z \in G}\{-\log(\Pr(Z))\} = \log(ord_N(g)) \geq m$$

The lemma is then a straightforward application of the Leftover Hash Lemma ([ILL], [IZ]).
**Lemma 8.2**: $x_1^{\lceil n/2 \rceil}$ are simultaneous secure given $h \circ h\,(f_{g,N}(X))$.
**Proof**: Identical to the proof of Theorem 5. The only difference results from the fact that the inputs to the oracle are not admissible triplets but $g$, $N$, $h$, $h(f_{g,N}(X))$. Note that the fact that $m$ is made public (through the publication of the range of $h$) does not detract from the security of $x_1^{\lceil n/2 \rceil}$ since it can be guessed in polynomial time (indeed we guess $m$ in our shifting and randomization techniques). $\square$

## 6 Discussion

In this paper we have explored some of the unique properties of exponentiation modulo a Blum integer, which make it the first number theoretic function all of whose bits are proven to be individually hard and half of whose bits are proven to be simultaneously hard. The results presented in this paper can be extended in several directions:

1. It is interesting to see which mixed groups of bits from the right and left half of $f_{g,N}$ can be proven to be simultaneously secure. We can show that the bits of the complement of every window of length $\lfloor n/2 \rfloor$ are simultaneously secure, i.e. for every $1 \leq j \leq \lceil n/2 \rceil$ the rightmost $j$ bits together with the leftmost $\lceil n/2 \rceil - j$ bits are simultaneously secure. In particular the rightmost $\lceil n/4 \rceil$ bits together with the leftmost $\lceil n/4 \rceil$ bits are simultaneously secure.

2. The factorization of Blum integers may remain intractable even if some of the bits of $P$ and $Q$ are known. Efficient factorization techniques are known only when at least $\lceil n/3 \rceil$ bits of $P$ or $Q$ are given [RS]. Assume that the factorization of Blum integers remains computationally hard even when we are given the $\lfloor n/4 \rfloor$ leftmost non-zero bits of $P$ or $Q$. Under this strengthened intractability assumption it is easy to show that three quarters of the bits of $f_{g,N}$ are simultaneously secure, as the length of the unknown part of $S$ is now only $\lceil n/4 \rceil + 1$ instead of $\lceil n/2 \rceil + 1$.

3. Let $F$ denote any set of composites for which it is assumed that it is computationally hard to distinguish Blum integers from the numbers in $F$ (and thus in particular it is difficult to factor these numbers). Such a set is, for example, the set of all composites $N$ which are the products of a small number of large primes. Under this strengthened assumption our results hold not only for Blum integers but for all $F$ as well, even though our proof techniques are not directly applicable to numbers in $F$. This generalization was first observed by Silvio Micali (personal communication).

## Acknowledgements

We would like to thank Noga Alon for his help in the proof of Proposition 1. Our paper greatly benefited from the helpful insight of Uriel Feige, Oded Goldreich, Rafi Heiman and Moshe Tenenholtz.

## References

[ACGS] Alexi, W., Chor, B., Goldreich, O., Schnorr, C.P., "RSA/Rabin bits are 1/2+1/*poly(log N)* secure", *Proc. 25th FOCS, 1984, pp. 449-457.*

[Ba] Bach, E., "Discrete Logarithms and Factoring", *Report No. UCB/CSD 84/186, Univ. of California, 1984.*

[BBS] Blum, L., Blum, M., Shub, M., "A Simple Secure Pseudo-Random Number Generator", *SIAM J. on Computing, Vol. 15, No. 2, 1986, pp. 364-383.*

[BG] Blum, M., Goldwasser, S., "An Efficient Probabilistic Public Key Encryption Scheme which Hides All Partial Information", *Proc. CRYPTO 84, pp. 289-302.*

[BM] Blum, M., Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM J. Computing, Vol. 13, No. 4, 1984, pp. 850-864.*

[BH] Boppana, R.B., Hirschfeld, R., "Pseudorandom Generators and Complexity Classes", *to appear in Advances in Computer Research, vol. 5, editor: S. Micali, JAI Press.*

[Ch] Chor, B., *Two Issues in Public Key Cryptography: RSA Bit Security and a New Knapsack Type System, MIT Press, 1986.*

[GKL] Goldreich, O., Krawczyk, H., Luby, M., "On the Existence of Pseudorandom Generators", *Proc. 29th FOCS, 1988, pp. 12-24.*

[GL] Goldreich, O., Levin, L.A., "A Hard-Core Predicate for all One-Way Functions, *Proc. 21st STOC, 1989, pp. 25-32.*

[GK] Goldwasser, S., Kilian, J., "Almost All Primes Can be Quickly Certified", *Proc. 18th STOC, 1986, pp. 316-329.*

[GM] Goldwasser, S., Micali, S., "Probabilistic Encryption", *JCSS, Vol. 28, 1984, pp. 270-299.*

[H] Håstad, J., "Pseudo-Random generators under Uniform Assumptions", *Proc. 22nd STOC, 1990, pp.395-404.*

[ILL] Impagliazzo, R., Levin, L.A., Luby, M., "Pseudo-Random Generation from One-Way Functions", *Proc. 20th STOC, 1988, pp. 12-24.*

[IN] Impagliazzo, R., Naor, M., "Efficient Cryptographic Schemes Provably as Secure as Subset Sum", *Proc. 30th FOCS, 1989, pp. 236-241.*

[IZ] Impagliazzo, R., Zuckerman, D., "How to Recycle Random Bits", *Proc. 30th FOCS, 1989, pp. 248-254.*

[KMO] Kilian, J., Micali, S., Ostrovsky, R., "Minimum Resource Zero-Knowledge Proofs", *Proc. 30th FOCS, 1989, pp. 474-479.*

[LW] Long, D.L., Wigderson, A., "The Discrete Logarithm Hides O(log n) Bits", *SIAM J. Computing, Vol. 17, No. 2, 1988, pp. 363-372.* Also: "How discreet is the Discrete Log?" *Proc. 15th STOC, 1983, pp. 413-420.*

[Na] Naor, M., "Bit Commitment Using Pseudo-Randomness", *Presented in CRYPTO 89.*

[Ra] Rabin, M.O., "Digital Signature and Public Key Cryptosystems as Intractable as Factoring", *Technical Report, MIT LCS TR-212, 1979.*

[RS] Rivest, R.L., Shamir, A., "Efficient Factoring Based on Partial Information", *Proc. Eurocrypt 85, pp. 31-34.*

[RSA] Rivest, R.L., Shamir, A., Adleman, L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM 21:120-126, 1978.*

[SS1] Schrift, A.W., Shamir. A. "On the Universality of the Next Bit Test", *Presented in Crypto 90.*

[SS2] Schrift, A.W., Shamir. A. "The Discrete Log is Very Discreet", *Proc. 22nd STOC, 1990, pp. 405-415.*

[VV] Vazirani, U.V., Vazirani, V.V., "Efficient and Secure Pseudo-Random Number Generator", *Proc. 25th FOCS, 1984, pp. 458-463.*

[Y] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. 23rd FOCS, 1982, pp. 80-91.*