The Discrete Log is Very Discreet

A. W. Schrift and A. Shamir
Department of Applied Mathematics
Weizmann Institute of Science
Rehovot 76100, Israel

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, almost all its bits are individually hard, and half 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 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 quardratic 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 every one-way function has at least a logarithmic number of hard bits. 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

© 1990 ACM 089791-361-2/90/0005/0405 $1.50

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, almost all its bits are individually hard, and the lower half 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. We also quote the recent improvements of Johan Hastad [Ha] who extended our techniques to prove that indeed all the bits of $f_{g,N}$ are hard and that the upper half of the bits are also simultaneously hard.

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 composite N.

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, $N = P \cdot Q$, and g a randomly chosen element in $\mathbf{Z}_N^*$, then:

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

The proof will appear in the full version of this paper.

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

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

Its inverse, the *discrete log modulo 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.

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 $Y = g^N \;(\mathrm{mod} N) = f_{g,N}(N)$. Let $S = f^{-1}_{g,N}(Y) = N - d \cdot ord_N(g)$, 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:

$$|S| \leq n/2 + O(1)$$

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 $f^{-1}_{g,N}(Z)$ 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 B be the set containing the polynomial fraction of the Blum integers for which C computes $f^{-1}_{g,N}(Z)$ successfully on a non-negligible fraction of $g \in \mathbf{Z}^*_N$ and $Z \in G$. Given $N \in B$ we use C to compute S by applying standard randomization techniques, and subsequently try to factor N using S in one of the following methods:

Method 1: For g such that $ord_N(g) > P + Q - 1$: $S = (P + Q - 1)$. Hence, by solving the two equations: $S = P + Q - 1$ and $N = P \cdot Q$ we get the full factorization of N.

Method 2: Let g_0 be a random element in $\mathbf{Z}^*_N$, and let $g = g_0^2$. Let k be the largest integer such that $2^k|(N-S)$. As g is an admissible generator, $g^{(N-S)/2^k} = 1 \;\;(\mathrm{mod} N)$ and therefore $g_0^{(N-S)/2^k}$ is a square root of 1 modulo N. With probability 1/2 $g_0^{(N-S)/2^k} \neq \pm 1$, which enables the factorization of N. Note that while method 1 succeeds for almost all $N \in B$, method 2 works for every $N \in B$ with arbitrarily high probability, but requires the knowledge of a square root of g. □

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

For a number U let $u_n ... u_1$ 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^k_j.

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 $f^{-1}_{g,N}(Z)$ 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 $f^{-1}_{g,N}$ (as in [LW]).

Theorem 3:
For every $1 \leq i \leq (1-\varepsilon)n$, with ε an arbitrarily small constant, the i-th bit of $f_{g,N}$ is hard.

Our result left open the question of the individual bit security of the extreme left bits ($i > \varepsilon n$). This was recently solved by Johan Hastad [Ha] who proved:

Theorem 4:
For every i : $n/2 \leq i < n - O(\log n)$ the i-th bit of $f_{g,N}$ is hard.

Proving the individual (as well as simultaneous) security of the $O(\log n)$ most significant bits of $f_{g,N}$ calls for a new definition of security for bits that are a-priori known to be biased (and therefore can be trivially predicted with probability greater that 1/2). Following the work in [SS] we can define this notion and prove the individual bit security of all the bits.

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.

Definition H.2: x_i is *hard* if for any family C of polynomial-size Boolean circuits that is given a random admissible triplet (g, N, Z), and for any polynomial $poly(n)$:

$$\frac{1}{b(i)} \cdot \Pr(C = x_i | C = 0) +$$

$$+ \frac{1}{1 - b(i)} \cdot \Pr(C = x_i | C = 1) < 2 + \frac{1}{poly(n)}$$

Theorem 5:
The $O(\log n)$ most significant bits of $f_{g,N}$ are hard.

We now present the full proof of Theorem 3 which contains most of the new techniques and procedures that are needed to obtain the above results.

Proof of Theorem 3:

Overview:

Suppose that for a certain i, the i-th bit is not hard. Then, there exists a polynomial-size oracle (circuit) $C: (g, N, Z) \rightarrow \{0,1\}$, (where (g, N, Z) is an admissible triplet) that succeeds with probability exceeding $1/2 + 1/n^k$ in predicting the i-th bit of $f_{g,N}^{-1}(Z)$, for some constant k. 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 one of the reductions of Theorem 2.

Intuitively, we can regard the oracle 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. 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 the $O(\log n)$ 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 three possible cases and show:

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

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

3. Every non-extreme bit to the left of the middle $(n/2 + O(\log n) \leq i \leq (1-\varepsilon)n$ for any $\varepsilon)$ is hard (Proposition 3.3).

The actual extraction of S in this theorem involves the basic Forward-Extract procedure, where the unknown bits of S are computed from the least significant to the most significant. We describe the procedure and its use in detail while proving the propositions. In the appendix we present a more complicated Backward-Extract procedure, where the bits are discovered from the most significant to the least significant. The Backward-Extract procedure is essential to the proofs of Theorems 4 and 5 (and later 8). It can also be used instead of the Forward-Extract procedure in propositions 3.1 and 3.3 but in that context it has no advantage.

We shall henceforth assume that the randomly chosen g is of high order and perform our analysis accordingly. The small probability that g is not of high order (Proposition 1) is taken into consideration in our final error estimation of deriving an incorrect value for S.

Main Techniques:

Let $V = f_{g,N}(U)$, for $U \leq ord_N(g)$. Let m be the location of the leftmost non-zero bit in the binary representation of $ord_N(g)$, $o_n..o_1$, i.e.: $o_j = 0$ for $m+1 \leq j \leq n$, but $o_m = 1$. Note that for high order g: $n - O(\log n) \leq m \leq n$. Note also 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

$(ZR_{m-1}(g, V))^2$ (mod$N$). 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 (a similar note applies to the randomization technique).

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 $\sqrt{ZR_1(V)}$ (mod$N$), 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 $Y' = f_{g',N}(N)$.

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 $g_j = g_{j-1}^2$ (mod$N$), 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.

Note that assumptions a.2 and a.3 are needed only to enable the right shifts, and can be dropped whenever right shifts are not performed.

Randomization Technique:
We perform the randomization by querying the oracle on polynomially many 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 $u_{i-1}...u_{i-O(\log n)} = 000...0$, we could discard the possibility of a carry except in the low probability event that $r_{i-1}...r_{i-O(\log n)} = 111...1$. As the actual values of $u_{i-1}...u_{i-O(\log n)}$ 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. The order of g, $ord_N(g)$, is unknown, and cannot be computed in polynomial time. We can guess an approximation $e = e_m...e_1$ of $ord_N(g)$ (which enables a sufficiently random choice of R) by the following three stages:
   1. Pick $0 \leq n - m \leq O(\log n)$ (using $O(\log\log n)$ random bits),
   2. Pick a value for $e_{m-1}...e_{m-O(\log n)}$,
   3. Set all the lower order bits of e to zero.

□

Proposition 3.1:
For every $i : n/2 - O(\log n) \leq i \leq n/2 + O(\log n)$ the i-th bit of $f_{g,N}$ is hard.
Proof:
To simplify the presentation, assume that we are given an oracle for the $n/2$-th bit. It is easy to see that the same reasoning applies to all $i : n/2 - O(\log n) \leq i \leq n/2 + O(\log n)$. We extract the half-sized secret S by using the following method:
The Forward-Extract Procedure:

1. Shift S $n/2 - O(\log n)$ bits to the left. This will not cause a wraparound for high order g.

2. Guess the $O(\log n)$ least significant bits of S which have not passed under the oracle's location, and then 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 sufficiently many random multiples $Y' \cdot g^R$ (mod$N$), for $R < e$ (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 whenever S is located to the left of its original location. (See the note following the proof of Proposition 3.3 for further discussion).

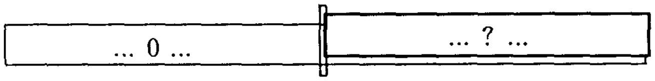
5. Repeat stages 3-4, to extract all bits, s_j, $O(\log n) \leq j \leq n/2 + O(1)$.

6. Repeat previous stages for each of the (polynomial number) of initial guesses at stage 2.

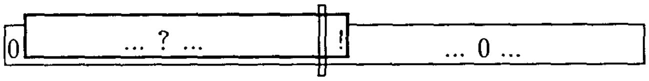
The correct value of S among the $n^{O(1)}$ resulting candidates from the procedure is chosen by trying to factor N with each computed S.

The following scheme illustrates the position of S during the procedure. We denote an unknown value of a bit by a question mark. All bits that are known apriori 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 $n/2$-th bit, where the oracle is located, is indicated by a box.

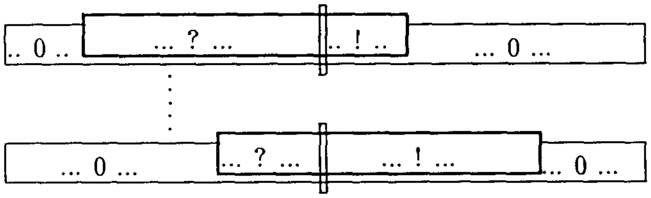
Before the procedure begins:



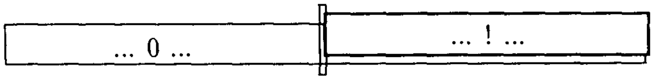
After stages 1,2: $O(\log n)$



During the procedure (stages 3-5):



Finally:



It is not difficult to show that:

Claim 1.1: The procedure yields at most $n^{O(1)}$ possible values for S.

Claim 1.2: With non-negligible probability one of the values is the correct value of S, where the probability is over the possible values of g, N and R.

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. □

Proposition 3.2:

For every $i : 1 \leq i \leq n/2 - O(\log n)$, the i-th bit of $f_{g,N}$ is hard.

Proof

Assume that for some $1 \leq i \leq n/2 - O(\log n)$, the i-th bit is not hard. In that case we use a simplified version of procedure Forward-Extract. As we shift S i bits to the left (stage 1 of Forward-Extract, where i substitutes the $n/2$-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, $1 \leq j \leq n/2$. In this simplified version we need not guess the least significant bits of S (stage 2), whereas in Proposition 3.1 some of the bits of S remain to the oracle's right after the initial shift so that stage 2 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 $n/2 - i$ of the bits of S unknown) we must use the general right shift technique. We choose a random g_0, create $\{g_i\}_{i=0}^{n/2+1}$ with $g_{i+1} = g_i^2$ (mod$N$) and use $g = g_{n/2+1}$ 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.

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

Claim 2.2: With non-negligible probability the value is the correct value of S, where the probability is over the possible values of g, N and R.

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. □

Proposition 3.3:

For every $i : n/2 \leq i \leq (1-\varepsilon)n$, where ε is an arbitrary small constant, the i-th bit of $f_{g,N}$ is hard.

Proof:

Assume that for some $i = (1-\alpha)n$, where $\varepsilon \leq \alpha \leq 1/2$, the i-th bit is not hard. The main problem we face here, is that S can be shifted at most $n/2 - O(\log n)$ bits to the left to avoid a wraparound, leaving $n/2 - \alpha n + O(\log n)$ of the bits of S right to the oracle's location. Such a large number of bits cannot be guessed efficiently. To solve that we use procedure Forward-Extract as an internal routine to extract blocks of αn consecutive bits of S, and repeat the use of the routine $1/\ \alpha$ times to fully extract S. While the bits of each block are discovered from the rightmost bit to the leftmost one, we start with the block that contain the most significant bits of S and finish with the block of the least significant bits, each time using the knowledge of the already extracted bits to move S further to the left and place more unknown bits under the oracle.

We begin by shifting S $n/2 - \alpha n$ bits to the left, having $s_{n/2}$ located at the i-th bit. We then use procedure Forward-Extract to extract the possible values of $s_{n/2} \ldots s_{n/2-\alpha n}$, where at stage 2 we guess the values of $s_{n-\alpha n+O(\log n)}, \ldots, s_{n-\alpha n}$. We cannot discard any of the possibilities yet, since we can try to factor

N only when most of the bits of S are known. We can, however, try each of the results for $s_{n/2}...s_{n/2-\alpha n}$ and zero it to enable an additional shift of S up to αn bits further to the left without causing a wraparound, so that we can perform yet another Forward-Extract procedure for $s_{n/2-\alpha n}...s_{n/2-2\alpha n}$. In general, at the j-th stage ($1 \leq j \leq 1/\alpha$) of the extraction process we zero each of the recently discovered values for $s_{n-(j-1)\alpha n}...s_{n-j\alpha n}$ to enable further left shifts of S and to discover $s_{n-j\alpha n}...s_{n-(j+1)\alpha n}$. The process increases the number of final candidates for S from $poly(n)$ to $poly^{1/\alpha}(n)$, but for non-extreme locations this remains polynomial. Therefore:

Claim 3.1: For any fixed α the above process yields at most $n^{O(1)}$ possible values for S.

Claim 3.2: With non-negligible probability one of the values is the correct value of S, where the probability is over the possible values of g, N and R.

Claim 3.3: The above process 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. □

Note: The shifts to the right in propositions 3.1 and 3.3 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. As a result Propositions 3.1 and 3.3 can be altered to hold for any high order g without assumptions a.2 and a.3. For such generators the extraction of S remains unchanged and the factorization of N (now not necessarily a Blum integer) is still possible via method 1.

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 with non-negligible probability 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: 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 indistiguishable from $(x_j^k,\ Z)$ for randomly chosen admissible $(g,\ N,\ Z)$ and a random $(k-j+1)$-bit string x_j^k.

Definition: The 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 $i-k$ $(j-i)$ 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 $poly(n)$.

Proposition 6:
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.

The proof of this equivalence involves a careful application of the techniques implemented in Yao's well known proof of the universality of the next bit test [Y].

Theorem 7:
The $n/2$ right hand bits of $f_{g,N}$ are simultaneously hard.

The techniques used to prove this theorem were recently extended by Johan Hastad [Ha] to yield:

Theorem 8:
The $n/2$ left hand bits of $f_{g,N}$ are simultaneously hard.

Proof of Theorem 7:
By Proposition 6 it suffices to show that every right hand bit of $f_{g,N}$ is relatively hard to the right. 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 $1 \leq i \leq n/2$ there exists an oracle $C(g,\ N,\ Z,\ x_1^{i-1})$ (for admissible triplets) that succeeds in predicting x_i with probability $1/2 + 1/n^k$, for some constant k. 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 $f^{-1}_{g,N}(Y')$ 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. □

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 single bits. 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 $n/2$ right hand bits of $f_{g,N}$ implies that $f_{g,N}$ hides $n/2$ 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 and construct a simple multi-bit commitment scheme accordingly.

Definition: A probability distribution function (pdf) over the n-bit variable X is *right non-uniform* if:

1. $x_n \ldots x_{n/2+1}$ are uniformly distributed, and
2. $x_{n/2} \ldots x_1$ are arbitrarily distributed.

Let $D(g,\ N,\ Z)$ denote any pdf of admissible triplets in which:

1. g and N are uniformly distributed.
2. the distribution of Z is induced by a right non-uniform pdf, $P(X)$, of $X = f^{-1}_{g,N}(Z)$.

Definition: The i-th bit of the function $f_{g,N}$ is *D-hard* if the conditions of definition H.2 hold under the probability distribution $D(g,\ N,\ Z)$.

Let p_j^k be the function defined in section 4.

Definition: k right most bits of $f_{g,N}$ are *simultaneously D-hard,* if $(p_1^k(f^{-1}_{g,N}(Z)),\ Z)$ is polynomially indistiguishable from $(x_1^k,\ Z)$, for D-distributed admissible $(g,\ N,\ Z)$ and P-distributed k-bit x_1^k.

Theorem 9:

The $n/2$ right hand bits of $f_{g,N}$ are simultaneously D-hard. In particular for every $1 \leq i \leq n/2$ the i-th bit of $f_{g,N}$ is D-hard.

Sketch of Proof:

The proof of the theorem is essentially a non-uniform version of the proof of Theorem 7. If the theorem is false then in particular there exists a certain assignment, A, for the $n/2$ right hand bits of $f^{-1}_{g,N}$ such that $(A,\ f_{g,N}(R \circ A))$ is polynomially distinguishable from $(A,\ Z)$, where $(g,\ N,\ Z)$ is a D-distributed admissible triplet, and R is a randomly chosen $(n/2 - O(\log n))$-bit string, s.t. $R \circ A < ord_N(g)$. However the proof of Theorem 7 can be strengthen to show that for any specific $n/2$-bit message, A, $(A,\ f_{g,N}(R \circ A))$ is polynomially indistiguishable 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 D-distributed.

Note that the theorem can be proven in the uniform complexity model under the additional assumption that the distribution P (and therefore D) is polynomially samplable (as in [ILL]). □

By Theorem 9 it is possible to commit to a $n/2$-bit value M by choosing randomly N and g, picking a uniformly distributed $(n/2 - O(\log n))$-bit R s.t. $R \circ M < ord_N(g)$ and sending $Z = f_{g,N}(R \circ M)$, 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 9.1:

Let $M_0,\ M_1 \in \{0,1\}^{n/2}$, be any pair of $n/2$-bit messages. Let $Z_i = f_{g,N}(R \circ M_i)$, $i = 0,1$, with R a uniformly distributed $(n/2 - O(\log n))$-bit string such that $R \circ M_j < ord_N(g)$. Then, $(M_i,\ Z_i)$ and $(M_i,\ Z_{1-i})$ are polynomially indistiguishable.

In order to perform the above commitment in practice it is necessary not only to verify that a randomly chosen generator has high order (which happens with high probability), 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 stronger 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.

5.2 Pseudo-Random Bit Generation

Any one-way function can be used for the construction of a pseudo-random bit generator, due to a recent result of [ILL]. However, this general technique is 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 $O(n)$ bits of $f_{g,N}$ to output as many bits as possible in every stage of the generation.

Using the results [ILL] and [IZ] we present a construction of an extender $G : \{0,1\}^l \rightarrow \{0,1\}^{l+O(n)}$ where $l = O(n)$. The pseudo-random bit generation is achieved through repeated applications of the extender to a random seed.

Let $N = P \cdot Q$ be a Blum integer of size n and let g be an admissible high order generator. Let $n - O(\log n) \leq m \leq n - 2$ be an integer such that $2^{m-1} < ord_N(g) \leq 2^m$. (As before, hard primes must be used to find m in practice.) Let $H_{n,t}$ be a family of universal hash functions, where $t = m - 4 \cdot \log^2 n$. In [IZ] some simple constructions are demonstrated, where $O(n)$ bits suffice to define a unique function $h \in H_{n,t}$. Let h be a randomly chosen function in $H_{n,t}$ and let X be a random m-bit string. Let $x_1^{n/2}$ denote the $n/2$ right hand bits of X and let $\circ$ denote concatenation. *The extender* G is:

$$G(h \circ X) = h \circ h(f_{g,N}(X)) \circ x_1^{n/2}.$$

Note: **1.** 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 G. Applying the hash function causes a $O(\log^2 n)$-bit loss in the length of G's output. The final $O(n)$ extension is possible only because of the many simultaneously secure bits, which more than compensate for this loss.
2. The values that are made public in G's construction are g, N and also m. This does not detract from the perfectness of G since m can be guessed in polynomial time (we used this fact in our shifting and randomization techniques).
Using the Leftover Hash Lemma of [IZ] combined with our proof of the simultaneous security of the bits of $f_{g,N}$ it is easy to show:

Theorem 10:
G is a perfect extender.

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. For example,we can show that for every $1 \leq j \leq n/2$ the rightmost j bits together with the leftmost $n/2-j$ bits are simultaneously secure, and in particular the rightmost $n/4$ bits together with the leftmost $n/4$ 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 $n/3$ bits of P or Q are given [RS]. Assume that the factorization of Blum integers remains computationally hard even when we are given the $n/4$ most significant 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 $n/4$ instead of $n/2$.

3. Let F denote the set of all composites N which are the products of a small number of large primes. Assume 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). 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 benefitted 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.*

[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.*

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

[Ha] Hastad, J., *private communication.*

[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", *Proc. 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.*

[SS] Schrift, A.W., Shamir. A. "On the Universality of the Next Bit Test", *unpublished manuscript, 1989.*

[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.*

Appendix: Procedure Backward-Extract

In this appendix we present an alternative extraction procedure for S, named 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 extreme left bits and the simultaneous security of the left half of the bits, is that at any stage of its application all the bits left to the oracle's location are zero. The procedure is applicable only when the oracle's location is left to the middle. To simplify our presentation assume that the oracle's location is exactly in the middle. Following is a description of the

Backward-Extract procedure, with $\circ$ denoting concatenation:

The Backward-Extract Procedure:

1. Find $s_{n/2}$ by guessing a value k of $s_{n/2-1}...s_{n/2-O(\log n)}$, zeroing these guessed bits (with the original Y transformed to Y_k^1) and querying the oracle on sufficiently many random multiples $Y_k^1 \cdot g^R$ (mod$N$), for $R < e$. Denote this current guess for the most significant bits of S by $CS_k^1 = s_{n/2} \circ k$.

2. Repeat stage 1 for all possible guesses $k = 0, ..., n^{O(1)}$ of the bits $s_{n/2-1}...s_{n/2-O(\log n)}$, thus creating $n^{O(1)}$ candidate values for the most significant bits of S: $CS_0^1, ..., CS_{n^{O(1)}}^1$.

3. Let $s_{n/2-j}$ be the bit that is currently evaluated. Let CS_k^j be the candidate value for the left j bits of S that we use for the evaluation. Guess a value b for $s_{n/2-j-O(\log n)}$. Let $v = k_1^{O(\log n)-1} \circ b$ be the current guess for $s_{n/2-j-1}, ..., s_{n/2-j-O(\log n)}$, where $k_1^{O(\log n)}$ denotes the value assigned by k to $s_{n/2-j-1}, ..., s_{n/2-j-O(\log n)+1}$. Shift S j bits to the left with the left j bits of S zeroed according to CS_k^j, placing $s_{n/2-j}$ at the middle. Zero $s_{n/2-j-1}, ..., s_{n/2-j-O(\log n)}$ according to v. Let Y_v^j denote the resulting Y.

4. Deduce $s_{n/2-j}$ by querying the oracle on sufficiently many random multiples $Y_v^j \cdot g^R$ (mod$N$), for $R < e$.

5. Check whether the resulting value of $s_{n/2-j}$ equals $k_{O(\log n)}$, i.e. the value that has been assigned to that bit while creating CS_k^j. If so, update the current guess for the $j+1$ left most bits of S to $CS_v^{j+1} = CS_k^j \circ b$.

6. Repeat stages 3-5 for the other possible guess of $s_{n/2-j-O(\log n)}$.

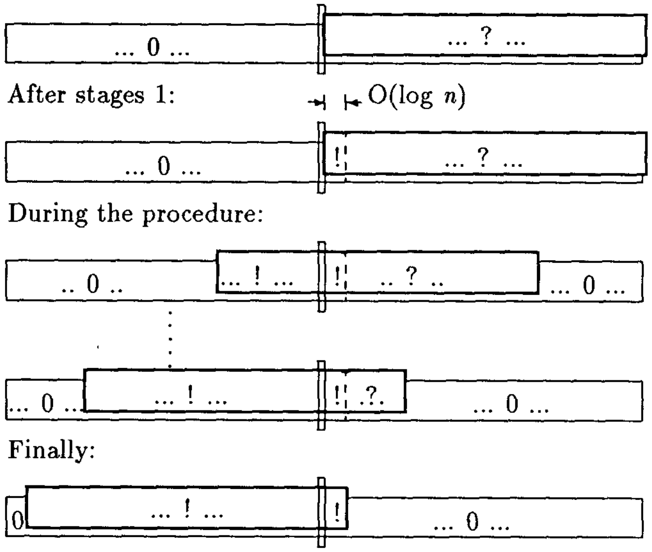
7. Repeat stages 3-6 for all existing candidate values for the j left bits of S.

8. Repeat previous stages to extract all bits.

In the above process we initially create $n^{O(1)}$ possible candidates for the $O(\log n)$ most significant bits of S. As we proceed, a certain candidate CS_v^{j+1} can be generated either from candidate value CS_k^j with $k_{O(\log n)} = 1$ or from a candidate value with $k_{O(\log n)} = 0$ but not from both, since we use the guess v to determine explicitly the value of $s_{n/2-j}$ and thus evaluate the guess k. Therefore, there are still at most $n^{O(1)}$ candidate values at every stage of the process. The correct value of S among the $n^{O(1)}$ resulting candidates is chosen by trying to factor N with each computed S.

The following scheme illustrates the position of S during the procedure. 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 $n/2$-th bit, where the oracle is located, is indicated by a box.

Before the procedure begins:


After stages 1:

During the procedure:

Finally: