How To Generate Cryptographically Strong Sequences Of Pseudo Random Bits*

Manuel Blum and Silvio Micali

Department of Elecrical Engineering and Computer Sciences
University of California - Berkeley

1. Introduction

1.1 Randomness and Complexity Theory

We introduce a new method of generating sequences of Pseudo Random Bits. Any such method implies, directly or indirectly, a definition of Randomness.

Much effort has been devoted in the second half of this century to make precise the notion of Randomness. Let us informally recall one of these definitions due to Kolmogorov [ ].

A sequence of bits $A = a_1, a_2, \ldots, a_k$ is random if the length of the minimal program outputting A is at least k.

We remark that the above definition is highly non constructive and rules out the possibility of pseudo random number generators. Also, the length of a program, from a Complexity Theory point of view, is a rather unnatural measure. A more operative definition of Randomness should be pursued in the light of modern Complexity Theory.

Let us consider the following example.

Example: A and B want to play head and tail in 4 different ways. In all of them A "fairly" flips a "fair" coin. In the first way, A asks B to bet and then flips the coin. In such a case we expect B to win with a 50% frequency. In the second way, A flips the coin and, while it is spinning in the air, she asks B to bet. We are still expecting B to win with a 50% frequency. However, in the second case the outcome of the toss is determined when B bets: in principle, he could solve the equation of the motion and win !

The third way is similar to the second one: B is allowed to bet when the coin is spinning in the air, but he is also given a pocket calculator. Nobody will doubt that in this case B is going to win with 50% frequency, as while he is still initializing any computation the coin will have come up head or tail.

The fourth way is similar to the third, except that now B is given a very powerful computer, able to take pictures of the spinning coin, and quickly compute its speed, momentum etc. In such a case we will not say that B will always win, but we may suspect he may win 51% of the time !

*Supported in part by NSF grant MCS 82-04506.

The purpose of the above example is to suggest that

The Randomness of an event is relative to a specific Model of Computation with a specified amount of computing resources.

The links between randomness and the computation model were first pointed out by Michael Sipser in [ ], where he shows that certain sequences appear random to a finite automaton. In his very nice paper [ ], Shamir considers also the factor of the computing resources, presents significant progress in this direction and points out some open problems as well.

In this paper we investigate the Randomness of k bit long sequences with respect to the computation model of **Boolean circuits with only Poly (k) gates.**

1.2 Our Generator

We show under which conditions it is possible to construct Generators of Cryptographically Strong Sequences of Pseudo Random Bits. Such a Generator is a program G that, upon receiving as input a random number s (hereafter referred to as " the seed "), outputs a sequence of Pseudo Random Bits $b_1, b_2, b_3, \ldots$

Our Generators have three main properties:

1) **The bits b_i's are polynomially many in the length of the seed.**
2) **The bits b_i's are easy to generate.** Each b_i is output in time polynomial in the length of the seed.
3) **The bits b_i's are unpredictable.** Given the Generator G and $b_1, \ldots, b_k$, the first k output bits, **but not** the seed s, it is computationally infeasible to predict the k+1st bit in the sequence with better than 50-50 chance.

1.3 Related results and applications

Our Generator is an improvement of Shamir's pseudo random number generator. In [ ], Shamir presents programs that from a short secret random seed, output a sequence of " unpredictable " numbers x_i's. The main differences between ours and Shamir's generators are:

a) **Shamir's notion of unpredictability is more restricted.** He proves that **not all** the generated x_i's can be computed from knowledge of the program and the preceding outputs, permitting that some of the x_i's **could** be so computed.

b) **Shamir's generator outputs numbers and not bits.** Such numbers could be unpredictable and yet of very special form. In particular every bit of (information about) the next number in the sequence could be heavily biased or predictable with high probability.

The classical sequence $x_{i+1} = ax_i + b \bmod n$, provides a fast way of generating pseudo random numbers. Such sequence is known to pass many statistical tests (see Knuth [ ]), however it is not Cryptographically Strong. Plumstead [ ], shows that the sequence can be inferred even when a, b and n are all unknown.

On the other hand, Yao [ ] proves a very interesting result about Cryptographically Strong Sequences of Pseudo Random Bits: they pass all Polynomial Time statistical tests. As a consequence , under the intractability assumption of the Discrete Logarithm Problem, Random Polynomial Time is contained in Deterministic Time (2^{n^ε}) for all $\varepsilon > 0$.

We finally point out the relevance of Cryptographically Strong Pseudo Random Bit Sequences to Cryptography. In **Private Key Cryptography**, one time pads constitute the simplest and safest type of Cryptosystem. Two partners who have exchanged one of our Generators and have **secretly** exchanged a random seed, are actually sharing a long bit sequence that can be used as a one time pad.

Our Generators also find applications in **Public Key Cryptography**. In [ ], Goldwasser and Micali show that, under the assumption that deciding quadratic residuosity modulo composite numbers is hard, there exist Encryption Schemes possessing the following property:

> An adversary, who knows the encryption algorithm and is given the cyphertext, cannot obtain any information about the cleartext.

Such Encryption Shemes are Probabilistic: the encoding of a message m depends on m and a sequence of coin tosses known only to the transmitter. In this context, Cryptographically Strong Pseudo Random Bits Generators are needed as an adversary might be able to decode not because he is able to efficiently decide quadratic residuosity, but because he is able to predict the random numbers used to encrypt! Such a worry is not an abstract one as shown by Plumstead.

An analysis of a particular simple pseudo random sequence generator appears in Blum, Blum, and Shub [ ]. They point out that *well-mixed sequences* in which *hard problems are embedded* can nevertheless be poor pseudo-random sequences. Something more is needed to construct good generators of pseudo random sequences; what that is is pointed out below.

2. The Generator Model

In this section we present a set of conditions that allow one to generate Cryptographically Strong Sequences of Pseudo Random Bits. In the next section we show that under the intractability assumption of the Discrete Logarithm Problem, it is possible to find a concrete implementation for the Generator Model.

Definitions. N = {0,1,2,...}. B is said to be a *set of predicates* if $B = \{B_i : D_i \to \{0,1\} \,/\, i \in S_n,\ n \in N\}$, where S_n is a subset of the n-bit integers and D_i is a subset of the integers with at most n bits.

B is an *accessible set of predicates* if for all $n \in N$ it is possible in Probabilistic Poly(n) Time to select any element in $I_n = \{ (i,x) \,/\, i \in S_n,\ x \in D_i\}$ with probability $\frac{1}{|I_n|}$.

Let B be a set of predicates. For any $\varepsilon > 0$, let $C_{n,\varepsilon}$ denote the size (number of gates) of a minimum size circuit C = C[i,x] that computes $B_i(x)$ correctly for at least a fraction $\frac{1}{2} + \varepsilon$ of the inputs $(i,x) \in I_n$. B is *input hard* if for any $\varepsilon > 0$ and any given polynomial Q, $C_{n,\varepsilon} >$ Q(n) for all sufficiently large n.

For example, suppose S_n = set of all n-bit composite integers that are products of two equal-length primes; $D_i = Z_i^*(+1)$, the set of all integers x relatively prime to i such that the Jacobi symbol (x/i) = +1; and B_i: x -> 1 if x is a quadratic residue mod i, 0 otherwise. Then it is easy to show that B is accessible. Furthermore, under the reasonable assumption that deciding quadratic residuosity modulo composite numbers is hard, B is input-hard.

Theorem 1: Let B be an input hard and accessible set of predicates. Let $\varepsilon > 0$, let Q and P be given polynomials, let $n \in N$ and $i \in S_n$, and suppose

1) the function f: i -> f_i is Poly(n) Time computable
2) $f_i : D_i \to D_i$ is a permutation computable in Poly(n) Time
3) the function $h : x \in D_i \to B_i(f_i(x))$ is Poly(n) Time computable.

Then it is possible in Poly(n) Time to compute, from initial random seeds $(i,x) \in I_n$, sequences $S_{i,x}$, each Q(n) + 1 bits long such that:

for each integer $k \in [1, Q(n)]$, for any circuit C of size less than P(n) with k Boolean inputs and one Boolean output y: if C is fed the first k bits of an $S_{i,x}$ sequence S, then Prob {y is equal to the k+1st bit of S} $< \frac{1}{2} + \varepsilon$ for all sufficiently large n. I.e. for all sufficiently large n $|\{ (i,x) \in I_n \,/\, y = \text{the k+1st bit of } S_{i,x}\}| < (\frac{1}{2} + \varepsilon)\, |I_n|$.

Proof: Let n be a natural number. As B is an accessible set of predicates, select (i,x) at random in I_n. (i,x) will be the seed of the Pseudo Random Bit Sequence. Set $c = Q(n) + 1$, the desired length of the sequence.

Generate the sequence $T_{i,x}$ = x, $f_i(x)$, $f_i^2(x)$,...., $f_i^c(x)$.

From right to left (!) , extract one bit from each element in $T_{i,x}$ in the following way: for j = c to 1, output the bit $B_i(f_i^j(x))$. (We note below that $B_i(f_i^j(x))$ is *easy* to compute because x is known, by (3)).

The above procedure constitutes the Generator that takes the random seed (i,x) and stretches it into the sequence $S_{i,x} = (s_j \mid 1 \le j \le c,\ s_j = B_i(f_i^{c-j+1}(x)))$.

We first prove that the Generator operates in Poly(n) Time. The sequence $T_{i,x}$ can be constructed in Poly(n) time as the two functions f: i -> f_i and $f_i : D_i$ -> D_i are both Poly(n) Time computable (hypothesis (1) and (2)).

Once the sequence $T_{i,x}$ is computed and stored, it is easy, by virtue of hypothesis (3), to compute each bit s_j of the $S_{i,x}$ sequence for $1 \le j \le c$.

We now prove that, when n is large enough, for any k between 1 and c, a circuit C with less than P(n) gates, cannot "predict" s_{k+1} with probability greater than $\frac{1}{2}+$ ε. The proof is by contradiction. Assume that there is a "small" circuit C predicting s_{k+1} with probability at least $\frac{1}{2}+\varepsilon$. Then we will show that the set of predicates B is not input hard. We will do this by showing that there is another "small" circuit that computes B_i(x) for a fraction bigger than $\frac{1}{2}+\varepsilon$ of the $(i,x) \in I_n$. Such a small circuit is derived by the following Poly(n) Time algorithm that makes calls to the circuit C.

> For each $(i,x) \in I_n$, generate the sequence of bits $(b_1,\ldots,b_{k-1}, b_k) = (B_i(f_i^k(x)),\ldots,B_i(f_i^2(x)), B_i(f_i(x)))$. Input these k bits to the circuit C to compute a bit y.

We reach a contradiction if we show that y equals B_i(x) for a fraction at least $\frac{1}{2}+\varepsilon$ of the $(i,x) \in I_n$. Notice that the bits $b_1,\ldots, b_k$ are the first k bits of the Pseudo Random Bit Sequence $S_{i,f_i^{k-c}(x)}$. Thus y = B_i(x) if and only if C correctly predicts the k+1st bit of $S_{i,f_i^{k-c}(x)}$. But this will happen for a fraction at least $\frac{1}{2}+\varepsilon$ of the $(i,x) \in I_n$ as the function f_i^{k-c} is bijective (as f_i is a permutation) and we are now assuming that C correctly predicts the k+1st bit of the $S_{i,x}$ sequences for at least a fraction $\frac{1}{2}+\varepsilon$ of the $(i,x) \in I_n$.

Qed

3. The Discrete Logarithm Problem

Let p be a prime. The set of integers [1, p-1] forms a cyclic group under multiplication mod p. Such group is denoted by Z_p^*. Let g be a generator for Z_p^*. The function $f_{p,g} : x \in Z_p^* \to g^x$ mod p, defines a permutation in Z_p^* computable in Poly(|p|) Time. The *Discrete Logarithm Problem* (DLP) with parameters p,g and y consists in finding the $x \in Z_p^*$ such that g^x mod p = y. A circuit C[.,.,.] *solves* the DLP mod a prime p if for any g generator for Z_p^* and any $y \in Z_p^*$, C[p,g,y] = x such that $x \in Z_p^*$ and g^x mod p = y. x will be simply denoted by $index_g$(y) whenever no ambiguity may arise about p.

3.1 Actual knowledge about the DLP

g^x mod p seems to be a one-way function. The fastest algorithm known for the DLP is due to Adleman and runs in time $O(2^{c\sqrt{\log p \ \log\log p}})$. It is easy to see that the difficulty of the DLP does not depend on the generator g or y. By this we mean that if for a non negligible fraction (1/Poly(|p|)) of pairs (g,y), g a generator and $y \in Z_p^*$, the DLP with parameters p,g and y could be efficiently solved, then it could be solved in Random Poly(|p|) Time for any g and any y. Thus our intractability assumption for the DLP will depend only on the prime p.

Pohlig and Hellman [ ] show that the DLP mod a prime p such that p-1 contains only small prime factors can be efficiently solved. However such primes constitute a negligible portion of all primes. We expect that for (nearly all) randomly selected primes p, p-1 has a large prime factor. No "small" circuits are known that solve the DLP mod a single prime p, for the primes p such that p-1 has a large prime factor (thus the DLP seems to have a higher circuit complexity than factoring: for any composite integer k **there is** a small circuit storing its factorization). In this paper we show how to generate Pseudo Random Bit Sequences under either one of the following assumptions.

Definition : A prime p is ***hard*** if p = P x + 1, where P is prime and $1 \le x \le$ Poly(|P|).

It is known (De la Vallee Poussin [ ]) that asymptotically $\frac{1}{|P|}$ of the integers of the sequence P x + 1, x=1,2,3,..., are primes.

Using efficient primality tests, there is an efficient procedure to decide if an integer, p, is a hard prime; and if so, to factor p-1.

First intractability assumption for the DLP. Let $\varepsilon > 0$ be a fixed constant and Q be a fixed polynomial. Then for all sufficiently large n, the size of any circuit that solves the DLP mod p for at least a fraction ε of the n-bits-long hard primes p, is greater than Q(n).

Second intractability assumption for the DLP. Let $\varepsilon > 0$ be a fixed constant and Q a fixed polynomial. Then for all sufficiently large n, the size of any circuit that solves the DLP for at least a fraction ε of the n-bit primes p, is greater than Q(n).

3.2 The DLP and the Principal Square Root Problem

We recall some known results about Z_p^*.

An element T of Z_p^* is called a quadratic residue if and only if T = x^2 mod p for some $x \in Z_p^*$; such an x is called a square root mod p of T.

Fact 1: Given any generator g for Z_p^*, an element T of Z_p^* is a quadratic residue mod p if and only if T = g^{2s} mod p for some $s \in [1, \frac{p-1}{2}]$. We recall that such a representation of T is unique. Moreover T has two square roots mod p: g^s mod p and $g^{s+((p-1)/2)}$ mod p. (see [ ])

Fact 2: There exists a polynomial time algorithm for testing whether an element T of Z_p^* is a quadratic residue mod p (See [ ]).

Fact 3 (Miller [ ], Adleman and Manders [ ], Berlekamp [ ]): Given any T, a quadratic residue mod p, there exists a random polynomial time algorithm to compute both square roots of T mod p.

We introduce the following basic definition.

Definition: Let g be a generator for Z_p^*, T a quadratic residue mod p and 2s the unique index of T such that $2s \in [1,p-1]$. Then g^s mod p will be called the ***principal square root*** of T, and $g^{s+((p-1)/2)}$ mod p the non principal square root of T.

Let g be a generator for Z_p^*. Notice that given T, a quadratic residue mod p, but not the index of T in base g, one can still test efficiently that T is indeed a quadratic residue and can efficiently extract its two square roots mod p, say X and Y. However the next theorem shows that deciding which square root of T is the principal one is a much harder problem. In fact, even allowing a **weak oracle** for the Principal Square Root Problem, the DLP becomes easy.

Definition: Let g be a generator for Z_p^* and $x \in Z_p^*$. The predicate $B_{p,g}(x)$ is defined to be equal to 1 if x is the principal square root of $x^2 \bmod p$ and 0 otherwise.

Remark 1: Notice that, given x, it is easy to evaluate $B_{p,g}(g^x \bmod p)$: just check whether $x < \frac{p-1}{2}$ or $x > \frac{p-1}{2}$, and output a 1 or a 0 respectively.

Theorem 2: Let $\varepsilon > 0$, p prime and g generator for Z_p^*. Then, given an oracle MB (Magic Box) such that $MB[x] = B_{p,g}(x)$ for a fraction $\geq \frac{1}{2} + \varepsilon$ of the $x \in Z_p^*$, one can construct an algorithm with oracle MB that solves the DLP mod p in Probabilistic Poly(|p|) Time.

We first establish some intermediate results.

Lemma 1: Let $\varepsilon > 0$, p prime and g generator for Z_p^*. Then, given an oracle MB such that $MB[x] = B_{p,g}(x)$ **for all** the $x \in Z_p^*$, there exist a Poly(|p|) Time Algorithm (with oracle MB) for the DLP mod p.

Proof : We will exhibit a Poly(|p|) Algorithm, making calls to MB, that finds indices mod p in base g. Such an algorithm will solve the DLP mod p as, for each generator h for Z_p^* and each $y \in Z_p^*$, $index_h(y)$ $index_g(h)$ mod p-1 = $index_g(y)$. The algorithm, given $y \in Z_p^*$, finds x = $index_g(y)$ bit by bit from right to left. In the middle of the execution, the variable *index* will contain the right half of the bits of x and the variable *element* is such that $index_g(element)$ equals the left half of x. Think of $index_g(element)$ and *index* as lists of 0's and 1's. The algorithm, abstractly, transfers the last bit of $index_g(element)$ in front of *index* until $index_g(element)$ vanishes (i.e. *element* = $'g^0$ = 1) and thus all of x has been reconstructed in *index*. " " denotes the concatenation operator.

```
Step 0  (Initialization)
    element := y;  index := empty word.
Step 1  (Check for termination condition)
    If element = 1  HALT. index  equals x.
Step 2  (find one more bit of x)
    Test whether element is a quadratic residue mod p.
    If yes index := 0 index  and go to step 4 else index
    := 1 index and go to step 3.
Step3(element  is  a  quadratic non residue, i.e.
    index_g(element) is odd. Change the last bit of
    index_g(element) from 1 to 0)
    element := $g^{-1}$ element mod p
Step 4  (Erase 0 from the tail of index_g(element) )
    element is a quadratic residue. Compute both
    square roots of element mod p. Have MB select the
    principal one. element := principal square root of
    element  and go to Step1.
```

Qed

The algorithm in lemma 1 needs, for |p| times, to select the Principal Square Root of a quadratic residue mod p. It does so by making |p| calls to the oracle MB that computes $B_{p,g}$ correctly 100% of the time.

We should ask what happens to the algorithm if it is allowed to make calls only to an oracle MB_ε that evaluates $B_{p,g}$ only slightly better than guessing at random, i.e. correctly for a fraction $\frac{1}{2} + \varepsilon$ of the $x \in Z_p^*$.

The following lemma, making use of the algebraic structure of Z_p^*, shows how to "concentrate a stochastic advantage", i.e. how to turn an oracle that answers **most** of the instances of a decision problem correctly into an oracle answering a **particular instance** correctly with arbitrarily high probability. Let us first recall the Weak Law of Large Numbers.

If $y_1, \ldots, y_k$ are k independent 0-1 variables such that $y_i = 1$ with probability α, and $S_k = y_1 + \ldots + y_k$, then for real numbers ψ and $\varphi > 0$,

$$k > \frac{1}{4\varphi\psi^2} \text{ implies that } \Pr\left(\left| \frac{S_k}{k} - \alpha \right| > \psi \right) < \varphi.$$

Let us define trials$(\psi,\varphi) = \frac{1}{4\varphi\psi^2}$. Notice that trials$(\psi,\varphi)$ is a polynomial in ψ^{-1} and φ^{-1}.

Lemma 2: Let $\varepsilon \in (0, \frac{1}{2})$, $\delta \in (0,1)$, p a prime and g a generator for Z_p^*. Set n = trials(ε,δ) and define IS, the *initial segment* of Z_p^* as follows: IS = $\{g^x \bmod p \; 1 \leq x \leq \frac{p-1}{n}\}$. Then, given an oracle MB_ε such that $MB_\varepsilon[x] = B_{p,g}(x)$ for at least a fraction $\frac{1}{2} + \varepsilon$ of the $x \in Z_p^*$, there is a Probabilistic Poly(|p|,ε^{-1}, δ^{-1}) Algorithm with oracle MB_ε that, with Probability 1-δ, correctly selects the Principal Square Root of **any** quadratic residue e mod p belonging to IS.

Proof:

Select $r_1, \ldots, r_n$ at random in $[1, \frac{p-1}{2}]$. Compute $2r_1, \ldots, 2r_n$. Compute $e_1 = eg^{2r_1} \bmod p$, $\cdots$, $e_n = eg^{2r_n} \bmod p$. All the e_i's are quadratic residues mod p as $index_g(e_i)$ is even for all i's. In fact $index_g(e_i) = (index_g(e) + 2r_i)$ mod p-1 and both $index_g(e)$ and p-1 are even. Compute the two square roots X_i and Y_i of each e_i. (Note that while these can be computed, it is not (yet) clear which of X_i and Y_i is principal.) For each e_i select $PSQR_i$, your guess for the principal square root of e_i, in the following way: if $MB_\varepsilon[X_i] = MB_\varepsilon[Y_i]$, set $PSQR_i$ = one of $MB_\varepsilon[X_i]$, $MB_\varepsilon[Y_i]$ selected at random with probability 1/2. Otherwise, if $MB_\varepsilon[X_i] = 1$, set $PSQR_i = X_i$; else set $PSQR_i = Y_i$. Notice that the e_i's have been drawn at random with uniform probability among the quadratic residues mod p: in fact every even index between 1 and p-1 can be uniquely written in the form $(index_g(e) + 2r)$ mod p-1, for $1 \leq 2r \leq p-1$. Thus, even if an **adversary** had chosen the x's for which $MB_\varepsilon[x] = B_g(x)$, the Weak Law of Large Numbers guarantees that with Probability 1-δ, $|\frac{S_n}{n} - (\frac{1}{2} + \varepsilon)| < \frac{\varepsilon}{2}$. I.e., with probability 1 - δ, we have selected the principal square root of the e_i's more correctly than incorrectly. We exploit this fact in the following way.

Initialize to 0 two counters C_X and C_Y. Compute a square root of e, call it X. For each r_i compute $S_i = Xg^{r_i}$ mod p.

If S_i = $PSQR_i$ then increment the counter C_X, else increment the counter C_Y.

Notice the following fact:

Let e = g^{2s} mod p ($2s \in [1,p-1]$) be a quadratic residue mod p and let X and Y be its square roots mod p. Let 2s + 2r < p-1. Then Xg^r mod p is the principal square root of eg^{2r} mod p if and only if X is the principal square root of e.

Without loss of generality, let $C_X > C_Y$ and let 2s be the index of e in base g. If **for all** the r_i's, $2s + 2r_i < p-1$; then with probability 1 - δ, X will be the Principal Square Root of e.

2s is unknown, but we know that $2s \in [1, \frac{p-1}{n}]$. Thus all r_i's for which $2s + 2r_i > p-1$ must belong to the interval $[(n-1)\frac{p-1}{n}, p-1]$. But the $2r_i$'s are n even integers drawn at random with uniform probability in [1,p-1]; thus each $2r_i$ has the same probability to belong to each of the n sub intervals $[k\frac{p-1}{n}, (k+1)\frac{p-1}{n}]$. Let t be the number of r_i's belonging to the dangerous interval $[(n-1)\frac{p-1}{n}, p-1]$. This t will be so small that also C_X - t will be greater than C_Y. Thus **still with probability** 1 - δ, X will be the Principal Square Root of e.

Qed

Lemma 3: Let $\varepsilon \in (0,\frac{1}{2})$ and $\varphi \in (0,1)$, p prime and g generator for Z_p^*. Set n = trials$(\varepsilon, \frac{1}{2|p|})$ and define IS = $\{ g^x \bmod p \mid x \in [1,\frac{p-1}{n}]\}$. Then, given an oracle MB_ε such that $MB_\varepsilon[x] = B_{p,g}(x)$ for at least a fraction $\frac{1}{2} + \varepsilon$ of the $x \in Z_p^*$, there is a Probabilistic Algorithm that finds indices of **any** $y \in IS$ in Expected Poly(|p|) Time.

Proof : Let y be any element in IS. Apply the algorithm in lemma 1 to find the index of y. In Step 4, to select the principal square root of a quadratic residue in IS, instead of calling MB, apply the algorithm in lemma 2 with $\delta = \frac{1}{2|p|}$. In view of lemma 2, Step 4 will be performed correctly with **independent** probability equal to $1 - \frac{1}{2|p|}$. Notice that if x belongs to IS, so does xg^{-1} mod p; and that if x is a quadratic residue mod p belonging to IS, also its principal square root will belong to IS. Therefore, if in Step 4 the algorithm correctly selects the principal square root, the total computation will be done in the initial segment IS. As Step 4 is executed at most |p| times, the probability that the index of y will be found correctly is greater than $(1-\frac{1}{2|p|})^{|p|} > \frac{1}{2}$. It is easy to see that the whole computation is polynomial in ε^{-1} and |p|, thus polynomial in |p| for sufficiently large p.

Qed

Proof of Theorem 2 : The following Probabilistic Poly(|p|) Time Algorithm finds $index_g(y)$ for **any** $y \in Z_p^*$. Set n = trials $(\varepsilon, \frac{1}{2|p|})$ and define IS = $\{ g^x \bmod p \mid x \in [1,\frac{p-1}{n}]\}$.

```
Step 0  (Initialization)
    i:=1
Step 1  (guess that $y \in [i\frac{p-1}{k}, (i+1)\frac{p-1}{k}]$ and map y into IS)
    w:= $yg^{-i\frac{p-1}{k}}$ mod p
Step 2  (find the index of w)
    Apply the algorithm in Lemma 3 to find the index of w.   index(w) := the index of w.
Step 3  (check whether the index of y has been found)
    candidate := index(w) + i $\frac{p-1}{k}$; if $g^{candidate}$ mod p = y then HALT: candidate is the index of y in base g. Else continue.
Step 4  (keep on guessing)
    i := i+1. If i > k then i:=1 and go to Step 0; else go to Step 1.
```

Qed

4. A concrete implementation of the General Model

We merely sketch the proofs that will appear in the final paper.

4.1 First Implementation

This implementation is more efficient than the second one. It assumes the first intractability assumption for the DLP **and** the constructability of the hard primes (suggested, but not implied, by the De La Vallee Poussin Theorem, which is an asymptotic result).

Let n $\in$ N. Let S_{2n} be the set of 2n-bit long integers i such that the first n bits of i constitute a hard prime p, and the next n bits a generator g for Z_p^*. For $i \in S_{2n}$, i = p g, set $D_i = Z_p^*$ and, for x an n-bit integer, set $B_i(x) = B_{p,g}(x)$. Then the set of predicates B = $\{B_i \mid i \in S_{2n}\}$ is an accessible, input hard set of predicates.

B is accessible : Flip 3n coins. An element $(i,x) \in I_{2n}$ has been obtained if

1) The first n bits constitute a hard prime p. This will happen in n^2 expected trials (Prime theorem & De La Vallee Poussin Theorem). Moreover, success can be easily detected by means of fast primality tests.
2) The next n bits constitute a generator for Z_p^*. This will happen in a low expected number of trials as the fraction of generators for Z_p^* is asymptotically greater than $\frac{1}{6 \log \log (p)}$. Also notice that as we easily have the complete factorization of p-1, it is easy to check whether g is a generator for Z_p^*.
3) The last n bits constitute an integer $x \in [1,p-1]$.

If the 3n flips have not generated a complete element of I_{2n}, flip 3n coins again.

B is input hard : If there were a circuit C, of size less than Q(n) for some fixed polynomial Q, that evaluates correctly $B_{p,g}(x)$ for a fraction of at least $\frac{1}{2} + \varepsilon$ of the n-bit inputs p,g, and x, then a counting argument shows that there would be a fraction of pairs (p,g) for which the circuit guesses $B_{p,g}(x)$ correctly for at least a fraction $\frac{1}{2} + \varepsilon$ of the $x \in Z_p^*$. By the results in the previous section, using C as an oracle, there would be a Probabilistic Poly(n) Time Algorithm for solving the DLP for a fixed fraction of the hard primes of n bits. As the size of C is bounded by Q(n) and any Probabilistic Poly Algorithm is easily seen to admit small circuits, the first intractability assumption for the DLP has been violated.

B satisfies the hypothesis of Theorem 1 : Define $f_i(x) = g^x$ mod p.

4.2 Second Implementation

Assume that we can pick a prime p with uniform probability, among those of a given size, so that the factorization of p-1 is known. Then, set S_{2n} equal to the set of 2n bit integers i such that the first n bits of i constitute a prime p and the second n bits a generator g for Z_p^*. Set $D_i = Z_p^*$. $f_i(x) = g^x$ mod p. Then as in the previous section, $\{B_i : x \rightarrow B_{p,g}(x) \mid i \in S_{2n}\}$ is an accessible set of predicates satisfying hypothesis (1), (2) and (3) of Theorem 1. However we do not know how to pick at random a prime p so that the factorization of p-1 is known. So, after having picked a prime p we would have trouble picking a generator for Z_p^*, as no way is known of proving that $x \in Z_p^*$ is a generator without having the factorization of p-1. However there is an "abundance" of generators in Z_p^*: one out of 6 log log (p) elements is a generator. Thus having picked at random k = log (p) elements $x_1, \ldots, x_k$ in Z_p^*, with probability greater than any fixed ε one of the x_i's will be a generator. Consider each x_i to be a generator for Z_p^*; and implement k Pseudo Random Bit Generators $G_1, \ldots, G_k$ as above. We now make use of the "exclusive or" function in a way similar to Yao [ ]. Construct the following new Pseudo Random Bit Generator G: generate the ith bit by outputting the ith bit for $G_1, G_2, \ldots, G_k$ and take their "exclusive or". It is easy to see that, if at least one of the G_i's is Cryptographically Strong so is G.

Acknowledgements

We are proud to thank many friends.

We are grateful to Shafi Goldwasser for numerous valuable discussions, to Richard Karp for his precious gift of setting the context and making vague ideas precise, and to Andy Yao for having brought to light hidden potentials.

This work has benefitted highly from the insightful comments of Erich Bach, Lenore Blum, Faith Fich, Donald Johnson, Donald Knuth, Leonid Levin, David Lichtenstein, Mike Luby, Gary Miller, Joan Plumstead, Ron Rivest, Jeff Shallit, Mike Sipser, Po Tong, Umesh Vazirani, Vijay Vazirani, and Frances Yao.

References:

[1] L. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," 20th FOCS (1979), 55-60.

[2] L. Blum, M. Blum, and M. Shub, "A Simple Secure Pseudo-Random Number Generator," in Proc. CRYPTO-82, ed. Allen Gersho.

[3] S. Goldwasser and S. Micali, "Probabilistic Encryption and How to Play Mental Poker Keeping Secret all Partial Information," 14th STOC (1982), 365-377.

[4] D. Knuth, "The Art of Computer Programming: Seminumerical Algorithms," Vol. 2, Addison-Wesley Pub. Co., 1981.

[5] J. Plumstead, "Inferring a Sequence Generated by a Linear Congruence," submitted to FOCS 1982.

[6] S. Pohlig and M. Hellman, "An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance," IEEE Trans. on Info. Theory, Vol. It-24, No. 1, (1978), 106-110.

[7] R. Rivest, A. Shamir, and L. Adleman, "On Digital Signatures and Public Key Cryptosystems," Commun. ACM, vol. 21 (Feb. 1978), 120-126.

[8] A. Shamir, "On the Generation of Cryptographically Strong Pseudo-random Sequences," ICALP 1981.

[9] M. Sipser, "Three Approaches to a Definition of Finite State Randomness," unpublished manuscript.

[10] A. Yao, "A Relation Between Random Polynomial Time and Deterministic Polynomial Time," submitted to FOCS 1982.