SIAM J. COMPUT.
Vol. 13, No. 4, November 1984

HOW TO GENERATE CRYPTOGRAPHICALLY STRONG SEQUENCES OF PSEUDO-RANDOM BITS*

MANUEL BLUM† AND SILVIO MICALI‡

Abstract. We give a set of conditions that allow one to generate 50–50 unpredictable bits.

Based on those conditions, we present a general algorithmic scheme for constructing polynomial-time deterministic algorithms that stretch a short secret random input into a long sequence of unpredictable pseudo-random bits.

We give an implementation of our scheme and exhibit a pseudo-random bit generator for which any efficient strategy for predicting the next output bit with better than 50–50 chance is easily transformable to an "equally efficient" algorithm for solving the discrete logarithm problem. In particular: if the discrete logarithm problem cannot be solved in probabilistic polynomial time, no probabilistic polynomial-time algorithm can guess the next output bit better than by flipping a coin: if "head" guess "0", if "tail" guess "1".

Key words. randomness, pseudo-random number generation, unpredictability, random self-reducibility

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 Kolmogorov's influential definition [18]:

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

We remark that the length of a program, from a computational complexity point of view, is a rather unnatural measure. We want to investigate a more operative definition of randomness in the light of complexity theory.

A mental experiment. A and B want to play head and tail in four 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 B is still going to win with 50% frequency: before he can initialize 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. We will not say that B will always win, but we may suspect he may win 51% of the time!

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.*

* Received by the editors April 11, 1983, and in final form January 15, 1984. Supported in part by National Science Foundation grant MCS 82-04506 and by a fellowship of Consiglio Nazionale delle Ricerche-Italy. A version of this paper was presented at the AMS Conference on Probabilistic Computational Complexity, June 1982, Durham, New Hampshire and in the 23rd FOCS, November 1982, Chicago, Illinois.

† Computer Science Department, University of California, Berkeley, California 94720.

‡ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

The links between randomness and the computation model were pointed out by Michael Sipser in [31], where he defines randomness with respect to finite state automata (see also [24]). In his very nice paper [29], 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. CSPRB generators. We introduce the notion of a cryptographically strong pseudo-random bit generator (CSPRB generator) and show under which conditions it can be constructed. A CSPRB generator is a program G that, upon receiving as input a random number i (hereafter referred to as the *seed*), outputs a sequence of pseudo-random bits $b_1, b_2, b_3, \cdots$. G possesses the following properties:

1) *The bits b_k's are easy to generate.* Each b_k is output in time polynomial in the length of the seed.

2) *The bits b_k's are unpredictable.* Given the generator G and $b_1, \cdots, b_s$, the first s output bits, *but not the seed i,* it is computationally infeasible to predict the $s+1$st bit in the sequence with better than 50–50 chance. Here s is polynomial in the length of the seed.

Our generators are an improvement of Shamir's pseudo-random number generators. In [29], Shamir presents programs that from a short secret random seed, output a sequence of numbers x_i's such that the ability of predicting the next output is equivalent to inverting the RSA function [27]. The main difference between ours and Shamir's generator is:

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. As a consequence, if the numbers so generated are 100 bits long, they might not be uniformly spread in the interval $[1, 2^{100}]$.

1.3. Pseudo-random sequences and statistical tests. Passing given statistical tests is the key point for evaluating pseudo-random sequences. The classical sequence $x_{i+1} = ax_i + b \bmod n$, provides a fast way of generating pseudo-random numbers. Such a sequence is known (for a clever choice of the parameters a, b, and n) to generate "well mixed numbers" (see Knuth [17]). However it is not cryptographically strong. Plumstead [25] shows that the sequence can be inferred even when a, b and n are all unknown.

In contrast, our bit-sequences cannot be generated as fast, but cannot be inferred either. This is so because they have "embedded" some hard problem.

An analysis of a particularly simple pseudo-random number generator appears in Blum, Blum and Shub [9]. 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 in § 2.

Unpredictability of the next output bit is the key test studied in this paper. In an earlier version of this paper [10], we presented a deterministic polynomial-time algorithm that stretched a random k-bit long seed into a polynomially (in k) long bit-sequence. Any probabilistic polynomial-time algorithm, correctly predicting the next bit with probability greater than $\frac{1}{2}+\varepsilon$ in a so produced pseudo-random sequence, could be easily transformed to a probabilistic algorithm, running in expected

poly (k, ε^{-1}) time, for solving the discrete logarithm problem for a fraction ε of the primes of length k.

Though we were aware of the polynomial dependency on ε^{-1} (in fact in [10] Lemma 2 explicitly states it), our main theorem summarized our results stating that the next bit in our pseudo-random sequences could not be predicted in polynomial (in k) time with probability greater than $\frac{1}{2}+\varepsilon$, for $0<\varepsilon<1$.

Yao [33] was the first one to realize the importance of emphasizing the polynomial functional dependency on ε^{-1} and made excellent use of it (see § 1.3.2).

Indeed, without any changes in our algorithm, ε could be replaced with the smallest value that will keep the running time polynomial. Since in our case the running time is polynomial in k and ε^{-1}, we henceforth use $1/\text{poly}(k)$ for ε in this paper.

We now proceed to a formal treatment.

1.3.1. The next-bit-test. Let P be a polynomial and $S=\{S_k\}$ be a collection of multisets such that S_k contains $P(k)$-bit long sequences (the same sequence s may belong more than once to S_k). Let P_1 be a polynomial. A *predicting collection* $C=\{C_k^i\}$ is a collection of circuits such that each circuit C_k^i has less than $P_1(k)$ gates, i Boolean inputs, $i<P(k)$, and one Boolean output. On input the first i bits of a sequence s randomly selected in S_k, C_k^i will output a bit b. Let $p_{k,i}^C$ denote the probability that $b=$ the $i+1$st bit of s. We say that the collection S *passes the next-bit-test* if for all predicting collections C, all polynomials Q, all sufficiently large k and all $i<P(K)$,

$$p_{k,i}^C<\frac{1}{2}+\frac{1}{Q(k)}.$$

Ability to predict the next bit from the preceding ones is indeed a statistical test for a bit-sequence. In fact, if a bit-sequence were generated by independent coin flips, no strategy would predict the next bit with a success rate even slightly better than 50–50. This particular test is passed by the sequences produced by a CSPRB generator. Therefore CSPRB *generators produce evenly distributed numbers*: just divide the output sequences into k-bit long segments.

Subsequently, Yao [33] showed the following very interesting result.

1.3.2. Yao's statistical test. The following definition is derived from Yao [33]. As before, the collection $S=\{S_k\}$ is such that the multiset S_k contains $P(k)$-bit long sequences.

Let P_1 be a polynomial. A *polynomial-size statistical test* is a collection $C=\{C_k\}$ of circuits. Each circuit C_k has less than $P_1(k)$ gates, $P(k)$ Boolean inputs and one Boolean output. Let $p_{k,S}^C$ denote the probability that C_k outputs 1 on input a randomly selected sequence in S_k, and $p_{k,R}^C$ the probability that C_k outputs 1 on a randomly selected $P(k)$-bit long sequence. The collection S *passes all polynomial-size statistical tests* if for all polynomial-size statistical test C, for any polynomial Q, for all sufficiently large k,

$$|p_{k,S}^C-p_{k,R}^C|<\frac{1}{Q(k)}.$$

THEOREM (Yao). *A collection $S=\{S_k\}$ passes the next-bit-test if and only if it passes all polynomial-size statistical tests.*

1.3.3. Related tests for strings. Earlier definitions and theorems about tests for distinguishing strings belonging to two different sets can be found in Goldwasser and Micali [13]. They presented a probabilistic encryption scheme in which a single bit b

is, with the help of a coin, encoded by a k-bit long string β, called a *probabilistic encryption of b.* Here k is a security parameter. Both 0 and 1 will have many possible probabilistic encodings, but all of them are uniquely decodable. They defined a probabilistic encryption scheme to be *bit-secure* if for all polynomials P and Q, for all sufficiently large k, no circuit with less than $P(k)$ gates can correctly guess whether β is the encryption of 0 or 1 with probability greater than $\frac{1}{2}+1/Q(k)$. Under an intractability assumption for the quadratic residuosity problem, they present a probabilistic encryption scheme that is bit-secure.

A probabilistic encryption of an n-bit long ($n<P_1(k)$ for some polynomial P_1) string $b_1, \cdots, b_n$ is a sequence $\beta_1, \cdots, \beta_n$ where each β_i is a probabilistic encryption of b_i. Let P be a polynomial. A *separator* is defined to be a collection of circuits $C=\{C_k\}$. Each circuit C_k has less than $P(k)$ gates, kn Boolean inputs and one Boolean output. For a string $s=s_1, \cdots, s_n$ let $p^C_{s,k}$ denote the probability that C_k outputs 1 on input a probabilistic encryption of s. The encryptions of the string $x=x_1, \cdots, x_n$ are *unseparable* from the encryptions of a string $y=y_1, \cdots, y_n$ if for all separators C and for all polynomials Q, for sufficiently large k,

$$|p^C_{x,k}-p^C_{y,k}|<\frac{1}{Q(k)}.$$

The computational difficulty of separating the encryptions of polynomially long bit-sequences reduces to the one of correctly guessing the decoding of an encrypted single bit.

THEOREM (Goldwasser and Micali). *For any pair of n-bit long strings x and y, the encryptions of x and y are unseparable if and only if the encryption scheme is bit-secure.*

1.4. Instances of the CSPRB generator model. A general algorithmic scheme for constructing CSPRB generators is presented in § 2. The first instance of this scheme is based on the intractability assumption for the discrete logarithm problem and is described in § 4. Other interesting instances of the general model have subsequently been found based on the intractability assumption of various one-way functions. Yao [33] and Blum, Blum and Shub [9] found instances based on the intractability of deciding quadratic residuosity modulo composite numbers whose factorization is unknown. Yao [33] and Goldwasser, Micali and Tong [14] implemented CSPRB generators based on the intractability of factoring. Yao [33] also proves that one can obtain instances of the CSPRB generator scheme if one-way functions with a particular property exist.

1.5. Applications. Recently, a large number of cryptographic protocols for protecting private communication and business transactions have been developed [7], [8], [13], [14], [15], [20]. The security of these new protocols is based both on the security of some encryption scheme and the ability of the participants to generate large random numbers unknown to an adversary. Security vanishes if an adversary, though not able to break the encryption scheme, can successfully predict the output of the pseudo-random number generator. This is not an abstract worry as shown by Plumstead. The problem calls for the use of CSPRB generators.

In private key cryptography, one-time pads constitute the best type of cryptosystem. In practice, one-time pads are approximated by pseudo-random number generators [4]. Shamir [29] points out that "unpredictable" pseudo-random number generators may be a valid substitute for one-time pads. Therefore, CSPRB generators are particularly good substitutes for one-time pads: two partners who both possess

the same CSPRB generator and have *secretly* exchanged a random seed, are actually sharing a long bit-sequence that can be successfully used as a one-time pad.

The fact that the cryptographic strength of CSPRB generators depends only on the secrecy of the seed and not on the secrecy of the program, makes them an available tool to mathematically nonsophisticated users. In fact, it is unreasonable to assume that a business person should be able to design a cryptographically strong generator. However, anyone can buy such a program from the public market and secretly select a short random seed.

2. A general algorithmic scheme for constructing CSPRB generators. In this section we present the formal definition of a CSPRB generator and a theorem showing a set of conditions that allow one to construct such generators. In § 3 we show some results about the discrete logarithm problem; namely that there exists a Boolean predicate whose computational difficulty is equivalent to that of the discrete logarithm problem. In § 4 we show that these results make possible, under the intractability assumption for the discrete logarithm problem, to concretely implement CSPRB generators. In § 5 we explicitly define the notion of random self-reducibility that is the basis of our results.

DEFINITION. Let Q be a polynomial, I a set of inputs and $I_k \subseteq I$ the set of inputs of length k. Let A be a deterministic algorithm that, on input a seed $x \in I_k$, outputs a $Q(k)$-bit sequence s_x. Let $S_k = \{s_x | x \in I_k\}$. The algorithm A is a Q-CSPRB *generator* if the collection $S = \{S_k\}$ passes the next-bit-test.

The sequences output by a CSPRB generator will be called the CSPRB sequences. CSPRB sequences are ultimately periodic. However, for the great majority of the seeds, the corresponding CSPRB sequences do not quickly become periodic with a short period.

Let α and β be integers. We say that a bit-sequence is (α, β)-periodic if it becomes periodic, with period length less than β, after at most α bits.

THEOREM 1. *Let P_1, P_2 and P_3 be polynomials. Set $Q = P_1 + P_2 + 2P_3 + 1$ and let G be a Q-CSPRB generator. Let δ_k denote the fraction of the seeds of length k for which G generates a $(P_1(k), P_2(k))$-periodic pseudo-random sequence. Then $\delta_k < 1/P_3(k)$ for all sufficiently large k.*

Proof. Assume, for contradiction, that $\delta_k \geqq 1/P_3(k)$ for each $k \in F$ where F is infinite. Let $k \in F$. Denote by ε_i the fraction of seeds of length k for which the first $P_1(k) + P_2(k) + i$ bits in the corresponding CSPRB sequence form a $(P_1(k), P_2(k))$-periodic sequence. Then we have

$$1 = \varepsilon_0 \geqq \varepsilon_1 \geqq \cdots \geqq \varepsilon_{2 \cdot P_3(k)} \geqq \delta_k \geqq \frac{1}{P_3(k)}.$$

Let the integer $i \in [0, 2 \cdot P_3(k))$ be such that $\varepsilon_i - \varepsilon_{i+1} \leqq \frac{1}{2}(1/P_3(k))$. (Such an i exists, otherwise $\varepsilon_0 > 1$.) Consider the following algorithm A_k that predicts the $(i+1)$st bit in a CSPRB sequence $b_1, b_2, \cdots, b_{Q(k)}$ produced with a seed of length k:

> Look at $S = b_1, \cdots, b_i$. If S is not a $(P_1(k), P_2(k))$-periodic sequence, predict b_{i+1} by flipping a coin. Else, predict b_{i+1} so to preserve the $(P_1(k), P_2(k))$-periodicity.

Because of $\varepsilon_i \geqq 1/P_3(k)$ and our choice of i, A_k will predict b_{i+1} correctly with probability greater than $\frac{1}{2} + 1/(2 \cdot P_3(k))$. We have reached a contradiction as, for some polynomial P, for each $k \in F$, A_k can be transformed to a circuit C_k, with less than $P(k)$ gates, that accomplishes the same task. Q.E.D.

We just mention that we could replace the above algorithms A_k by a uniform probabilistic polynomial time algorithm that makes use of sampling.

2.1. Problems in building generators. Let B be a predicate defined in a domain D with 2^k elements, such that $B(x)=1$ for half of the x's in D. Then we could generate random bits $b_0, b_1, \cdots$ by picking x_i at random in D and outputting $b_i = B(x_i)$. The drawback of this method is that we need to use k random bits to pick each x_i, but we generate a single bit b_i. Instead, we would like to pick only the first x_0 at random in D and to select the other x_i's in a deterministic way, namely, by setting $x_{i+1} = f(x_i)$ where f is a deterministic function. The problems of this approach are illustrated by the following example. Let D consist of the integers in the interval $[0, n]$, $B(x)=1$ if $x < n/2$ and $B(x)=0$ otherwise, and let $f(x)=x+1$. With such a choice for f, we would essentially output always 0 or always 1, a not too random-looking bit-sequence! This simple example shows that the deterministic function f *may interact badly with the predicate* B spoiling the "randomness" of the output. Theorem 2 essentially shows simultaneous conditions on f and B that prevent such bad interaction. We first describe what predicates B should be used.

2.2. Unapproximable predicates. $N=\{0, 1, 2, \cdots\}$. 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. An element of D_i is always represented by n bits, the leading ones may be 0's.

Set $I_n = \{(i, x) | i \in S_n \text{ and } x \in D_i\}$. An element of I_n is called an input of size n. B is *accessible* if there are two constants c_1 and c_2 and a probabilistic algorithm A such that, on input n, A halts after n^{c_1} steps; A outputs "?" with probability $1/2^{c_2 k}$; and whenever A does not output "?" it outputs an element $(i, x) \in I_n$ with uniform probability.

Let B be a set of predicates and P be a polynomial. Let c_n^P denote the size of a minimum size circuit $C = C[\cdot, \cdot]$ that computes $B_i(x)$ correctly (i.e. $C[i, x] = B_i(x)$) for at least a fraction $\frac{1}{2} + 1/P(n)$ of the inputs (i, x) of size n. Such a circuit C will be said to $1/P(n)$*-approximate* B. B is *unapproximable* if for any polynomial P, c_n^P grows faster than any polynomial in n.

Example (Goldwasser and Micali [13]). Let S_n = set of all n-bit composite integers that are products of two distinct equal-length primes. Let D_i denote the set of all integers $x \in [1, i]$ relatively prime to i whose Jacobi symbol $(x/i) = +1$; and let $B_i(x)=1$ if x is a quadratic residue mod i, $B_i(x)=0$ otherwise. Then it is easy to show that B is accessible. Furthermore, under the quadratic residuosity assumption [13], B is also unapproximable. Another example can be found in § 4.

Remark. Note that for an unapproximable predicate B, as n goes to infinity, the fraction of I_n such that $B_i(x)=1$ ($B_i(x)=0$) goes to $\frac{1}{2}$.

2.3. Sufficient conditions for constructing CSPRB generators.

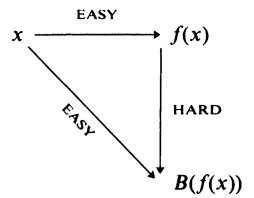

FIG. 1

THEOREM 2. *Let Q be a polynomial, $B=\{B_i: D_i \to \{0,1\} | i \in S_n, n \in N\}$, an unapproximable and accessible set of predicates and $I=\{(i, x) \in I_n | n \in N\}$ be the set of all*

inputs relative to B. Let

1) $f:(i,x)\in I\to D_i$ *be a polynomial time computable function*;

2) $f_i\equiv f(i,\cdot):D_i\to D_i$ *be a permutation for all* $i\in S_n$,

3) $h:(i,x)\in I\to B_i(f_i(x))$, *be a polynomial time computable predicate* (*see Fig.* 1).

Then it is possible to construct a Q-CSPRB *generator* G.

Proof. Let $n\in N$. As B is an accessible set of predicates, let c_1, c_2 and A be its relative constants and probabilistic algorithm. Set $c=Q(n)$, the desired length of the sequence and $n'=\lfloor n^{1/c_1}\rfloor$. The following constitutes the Q-CSPRB generator G that stretches the random n-bit seed r to a $Q(n)$-bit pseudo-random sequence.

> Run A on input n' using the bits of r as coin tosses. If A's output is "?" then generate the sequence consisting of c 0's. Else, if A has randomly selected an input $(i,x)\in I_{n'}$,
>
> generate the sequence $T_{i,x}=x,f_i(x),f_i^2(x),\cdots,f_i^c(x)$ and
>
> from right to left (!), extract one bit from each element in $T_{i,x}$ as follows:
>
> for $j=c$ to 1, output the bit $B_i(f_i^j(x))$.

For simplicity, let us assume that A never outputs "?" and $n=n'$. Then G takes the random input (i,x) and stretches it into the sequence $S_{i,x}=(s_j)_{j=1,\cdots,c}$ where $s_j=B_i(f_i^{c-j+1}(x))$.

G operates in polynomial time. The sequence $T_{i,x}$ can be constructed in Poly (n) time as the function f (and thus each function f_i) is polynomial time computable (hypothesis (1)).

Once the sequence $T_{i,x}$ is computed and stored, each bit $s_j\in S_{i,x}$ can also be computed in polynomial time: by hypothesis (3), $s_j=B_i(f_i^{c-j+1}(x))$ is easy to compute as $f_i^{c-j}(x)$ is given.

G is cryptographically strong. Let P_1 and P_2 be polynomials. We want to prove that, when n is large enough, for any k between 1 and $c-1=Q(n)$, a circuit C with less than $P_1(n)$ gates, cannot "predict" s_{k+1} with probability greater than $\frac{1}{2}+1/P_2(n)$. The proof is by contradiction. Assume that there is an infinite family of integers, F, such that for each $n\in F$ there is a circuit C_n, with less than $P_1(n)$ gates, predicting s_{k+1} correctly with probability (taken over all the possible seeds of length n) at least $\frac{1}{2}+1/P_2(n)$. Then the following poly(n) time algorithm A, making calls to the circuits C_n, $1/P_2(n)$-approximates B; i.e. $A(i,x)=B_i(x)$ for a fraction $\frac{1}{2}+1/P_2(n)$ of the $(i,x)\in I_n$. This will contradict the assumption that B is unapproximable. In fact, as all C_n have size less than $P_1(n)$, for some polynomial P_3, for each $n\in F$, A can be transformed to a circuit, with less than $P_3(n)$ gates, that $\frac{1}{2}+1/P_2(n)-$approximates B for inputs of size n.

> ALGORITHM A.
>
> For input $(i,x)\in I_n$, $n\in F$, generate the sequence of bits $(b_1,\cdots,b_{k-1},b_k)=(B_i(f_i^k(x)),\cdots,B_i(f_i^2(x)),B_i(f_i(x)))$. Input these k bits to the circuit C_n to compute a bit y. Set $A(i,x)=y$.

We now prove that A $1/P_2(n)$-approximates B for inputs of size n, $n\in F$. Notice that the bits $b_1,\cdots,b_k$ are the first k bits of the Q-CSPRB sequence $S_{i,f_i^{k-c}(x)}$. Thus $A(i,x)=B_i(x)$ if and only if C_n correctly predicts the $k+1$st bit of $S_{i,f_i^{k-c}(x)}$. But this will happen for a fraction at least $\frac{1}{2}+1/P_2(n)$ of the $(i,x)\in I_n$. In fact we are assuming that C_n correctly predicts the $k+1$st bit of the $S_{i,x}$ sequences for a fraction $\frac{1}{2}+1/P_2(n)$ of the $(i,x)\in I_n$ and we know that the function f_i^{k-c} is a permutation as, by hypothesis (2), f_i is. Q.E.D.

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 a group is denoted by Z_p^*. Let

g be a generator for Z_p^*. The function $f_{p,g}: x \in Z_p^* \to g^x \bmod p$ defines a permutation on Z_p^* computable in Poly $(|p|)$ time. The *discrete logarithm problem* (DLP) with inputs p, g and y consists in finding the $x \in Z_p^*$ such that $g^x \bmod p = y$. A circuit $C[\cdot, \cdot, \cdot]$ *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 $g^x \bmod p = y$. Such an x will be simply denoted by $\text{index}_g(y)$ whenever no ambiguity may arise about p.

3.1. Actual knowledge about the DLP. $g^x \bmod p$ seems to be a one-way function. The fastest algorithm known for the DLP is due to Adleman [1] 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 nonnegligible fraction $(1/\text{Poly}(|p|))$ of pairs (g, y), g a generator and $y \in Z_p^*$, the DLP with inputs p, g and y could be efficiently solved, then it could be solved in probabilistic 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 [26] 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 fraction of all primes [34]. No one knows how to construct "small" circuits that solve the DLP mod *even a single* prime p that is not of the above type.

3.2. The intractability assumption for the DLP. Let P be a polynomial and let c_n^P denote the size of a smallest size circuit C that solves the DLP for at least a fraction $1/P(n)$ of the n-bit primes p. Then c_n^P grows faster than any polynomial in n.

Why circuit complexity? The above intractability assumption is certainly a strong one. It implies that the CSPRB sequences, implemented using the DLP as described in § 4, resist prediction by polynomial size circuits.

For the same implementation, if we assume that the DLP cannot be solved in probabilistic polynomial time, we could prove (in essentially the same way!) that the CSPRB sequences would resist prediction by any *fixed* probabilistic polynomial time Turing machine M, i.e., for all sufficiently large seed length k, M cannot predict the next bit in a CSPRB sequence generated with a seed of length k better than at random.

This, however, is not satisfactory for the cryptographic applications mentioned in § 1.5. We would like first to choose a seed length, and then allow our adversary to choose *any* probabilistic polynomial time Turing machine for predicting our sequence! The problem calls for nonuniform complexity.

3.3. 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 mod p if and only if $T = x^2 \bmod p$ 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 $T = g^{2s} \bmod p$ for some integer $s \in [1, (p-1)/2]$. We recall that such a representation of T is unique. Moreover T has two square roots* mod p: $g^2 \bmod p$ *and* $g^{s+(p-1)/2} \bmod p$ (*e.g. see Shanks* [30]).

FACT 2. *There exists a polynomial time algorithm for testing whether an element T of Z_p^* is a quadratic residue* mod p (*e.g. see* [30]).

FACT 3 (Adleman, Manders and Miller [2], Berlekamp [5]). *Given any T, a quadratic residue* mod p, *there exists a probabilistic 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 $g^s \bmod p$ will be called the

g-principal square root of T, and $g^{s+(p-1)/2} \bmod p$ the *g-nonprincipal square root* of T. We will simply say principal square root and nonprincipal square root when no ambiguity about the generator g may arise.

Let g be a generator for Z_p^*. Notice that given T, a quadratic residue mod p, but not the index of T 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 Theorem 3 shows that deciding which square root of T is the g-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 $s \in Z_p^*$ such that $x = g^s \bmod p$, it is easy to evaluate $B_{p,g}(x)$: just check whether or not $s \leqq (p-1)/2$ and output 1 or 0 respectively.

THEOREM 3. *Let Q be a polynomial. Let $MB_Q[\cdot, \cdot, \cdot]$ (magic box) be an oracle such that, for all primes p and for all generators for Z_p^*, $MB_Q[p, g, x] = B_{p,g}(x)$ for a fraction at least $\frac{1}{2} + 1/Q(|p|)$ of the $x \in Z_p^*$. Then there is a probabilistic algorithm with oracle MB_Q that, for all primes p, solves the* DLP mod p *in expected* poly $(|p|)$ *time.*

We first establish some intermediate results. The following lemma shows that with an oracle for the principal square root problem, the DLP is solvable in polynomial time.

LEMMA 1. *Let $MB[\cdot, \cdot, \cdot]$ be an oracle such that, for all primes p, for all generators g for Z_p^* and all $x \in Z_p^*$, $MB[p, g, x] = B_{p,g}(x)$. Then there is a* poly $(|p|)$ *time algorithm with oracle MB that solves the* DLP mod p *for all primes p.*

Proof. We actually prove a stronger result: we exhibit a poly $(|p|)$ time algorithm that finds indices base $g \bmod p$ by only making use of the more restricted oracle $MB[p, g, \cdot]$.

The algorithm, given by $y \in Z_p^*$, finds $x = \text{index}_g(y)$ 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 $\text{index}_g(\textit{element})$ = the left half of x. Think of $\text{index}_g(\textit{element})$ and *index* as lists of 0's and 1's. The algorithm, abstractly, transfers the last bit of $\text{index}_g(\textit{element})$ in front of *index* until $\text{index}_g(\textit{element})$ vanishes (i.e. $\textit{element} = g^0 = 1$) and thus all of x has been reconstructed in *index*. "$\sim$" denotes the concatenation operator.

Step 0 (Initialization)
  $\textit{element} := y$; *index* := empty word.
Step 1 (check for termination condition)
  If $\textit{element} = 1$ HALT. $\textit{index} = x$.
Step 2 (find one more bit of x)
  Test whether *element* is a quadratic residue mod p. If yes $\textit{index} := 0 \sim \textit{index}$ and go to step 4 else $\textit{index} := 1 \sim \textit{index}$ and go to step 3.
Step 3 (*element* is a quadratic nonresidue, i.e. $\text{index}_g(\textit{element})$ is odd. Change the last bit of $\text{index}_g(\textit{element})$ from 1 to 0)
  $\textit{element} := g^{-1} \cdot \textit{element} \bmod p$
Step 4 (erase 0 from the tail of $\text{index}_g(\textit{element})$)
  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 Step 1.
Q.E.D.

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 to an oracle that evaluates $B_{p,g}$ only slightly better than guessing at random.

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, *even if we do not know which ones*!, into an oracle answering *any specific instance* correctly with arbitrarily high probability. Let us recall one version of the weak law of large numbers:

If $y_1, \cdots, y_k$ are k independent 0–1 *variables such that $y_i = 1$ with probability α, and $S_k = y_1 + \cdots + y_k$, then for real numbers ε and $\delta > 0, k > 1/(4\varepsilon^2\delta)$ implies that* Prob $(|S_k/k - \alpha| > \varepsilon) < \delta$.

Let us define trials (ε, δ). trials $(\varepsilon, \delta) = 1/(4\varepsilon^2\delta)$. Notice that trials (ε, δ) depends polynomially on ε^{-1} and δ^{-1}.

Let p be a prime, g a generator for Z_p^* and $t \in [1, p-1]$. Then $IS(p, g, t)$, the *t-initial segment of Z_p^** with respect to g, is defined by $IS(p, g, t) = \{g^x \bmod p | 0 \leqq x \leqq (p-1)/t\}$.

LEMMA 2. *Let $\varepsilon \in (0, \frac{1}{2})$ and $\delta \in (0, 1)$. Set $t =$* trials $(\varepsilon/2, \delta)$. *Let $MB_\varepsilon[\cdot\,,\cdot,\cdot]$ be an oracle such that for p prime, g generator for Z_p^* and $x \in Z_p^*$, $MB_\varepsilon[p, g, x] = B_{p,g}(x)$ for a fraction at least $\frac{1}{2} + \varepsilon$ of the $x \in Z_p^*$. Then, there is a probabilistic* poly $(|p|, \varepsilon^{-1}, \delta^{-1})$ *algorithm with oracle MB_ε that on input p (prime) and e (quadratic residue* mod p *belonging to $IS(p, g, t)$) selects the g-principal square root of e correctly with probability greater than $1 - \delta$.*

Proof. Let p prime and g generator for Z_p^*. Again, to find indices base g mod p we will only make use of the more restricted oracle $MB_\varepsilon[p, g, \cdot]$. As in the rest of the proof p and g will remain fixed, we write $MB_\varepsilon[x]$ instead of $MB_\varepsilon[p, g, x]$. On input $e \in IS(p, g, t)$, e quadratic residue mod p, select $r_1, \cdots, r_t$ at random in $[1, (p-1)/2]$. Compute $2r_1, \cdots, 2r_t$. Compute $e_1 = e \cdot g^{2r_1} \bmod p, \cdots, e_n = e \cdot g^{2r_t} \bmod p$. All the e_i's are quadratic residues mod p as $\text{index}_g(e_i)$ is even for all i's. In fact $\text{index}_g(e_i) = (\text{index}_g(e) + 2r_i) \bmod p-1$ and both $\text{index}_g(e)$ and $p-1$ are even. Compute the two square roots X_i and Y_i of each e_i. (Note that while both square roots 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 $MB_\varepsilon[X_i] = MB_\varepsilon[Y_i]$, randomly select, with probability $\frac{1}{2}$, one of the two square roots X_i and Y_i; call Z_i your selection and set $PSQR_i = Z_i$. 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 $(\text{index}_g(e) + 2r) \bmod p-1$, for $1 \leqq 2r \leqq p-1$. Thus, even if an *adversary* has chosen the x's for which $MB_\varepsilon(x) = B_g(x)$, setting $\alpha' =$ Prob ($PSQR_i$ is the principal square root of e_i), we have $\alpha' = \frac{1}{2} + \varepsilon$.

Notice the following fact:

Let $2s$ be the index of e, i.e. $e = g^{2s}$ mod p *and $2s \in [1, p-1]$, and let X and Y be its square roots* mod p. *Let $2s + 2r < p - 1$. Then $X \cdot g^r$* mod p *is the principal square root of $e \cdot g^{2r}$* mod p *if and only if X is the principal square root of e.*

$2s$ is unknown, but, as $e \in IS(p, g, t)$, we know that $2s \in [1, (p-1)/t]$. Therefore, if $2s + 2r_i > p - 1$, $2r_i$ must belong to the interval $[(t-1)(p-1)/t, p-1]$. This will happen with probability $= 1/t$. Assume, without loss of generality, that $PSQR_i = X_i$ and $X_i \cdot g^{r_i} \bmod p = X$. Then,

$$\alpha = \text{Prob}\,(B_{p,g}(X) = 1 | PSQR_i = X_i) \geqq \alpha' - 1/t > \tfrac{1}{2} + \varepsilon/2.$$

(Recall that $t = 1/\varepsilon^2\delta$.) We exploit this fact in the following way: initialize to "0" two counters C_X and C_Y. For each r_i if $X \cdot g^{r_i} \bmod p = PSQR_i$ then increment C_X, else increment C_Y. Upon termination, if $C_X > C_Y$ output X as the principal square root of e, or else Y. As $\alpha > \frac{1}{2} + \varepsilon/2$ let X be the principal square root of e, then by the weak law of large numbers, Prob $(|C_X/t - \alpha| > \varepsilon/2) < \delta$. Or equivalently, $C_X > C_Y$ with Probability $> 1 - \delta$. Q.E.D.

LEMMA 3. *Let Q be a polynomial and*

$$t = \text{trials}\left(\frac{1}{Q(|p|)}, \frac{1}{2|p|}\right).$$

Let MB_Q be an oracle such that, for all primes p and all generators g for Z_p^*, $MB_Q[p, g, x] = B_{p,g}(x)$ for at least a fraction $\frac{1}{2} + 1/Q(|p|)$ of the $x \in Z_p^*$. Then there is a probabilistic poly $(|p|)$ *algorithm that on input p prime and $y \in IS(p, g, t)$ finds* $\text{index}_g(y) \bmod p$ *in expected* poly $(|p|)$ *time.*

Proof. On inputs p, g, and y we will only call the more restricted oracle $MB_\varepsilon[p, g, \cdot]$. Let y be any element in $IS(p, g, t)$. We apply a modification of the algorithm in Lemma 1 to find the index of y. That algorithm, in Step 4, to select the principal square root of a quadratic residue mod p, and thus also for a quadratic residue in $IS(p, g, t)$, would call the oracle MB. Call instead MB_Q as in the algorithm of Lemma 2 setting $\varepsilon = 1/Q(|p|)$ and $\delta = 1/2|p|$. By Lemma 2, Step 4 will be performed correctly with *independent probability* equal to $1 - 1/2|p|$. Notice that if x belongs to $IS(p, g, n)$, so does $x \cdot g^{-1} \bmod p$; and that if x is a quadratic residue mod p belonging to $IS(p, g, t)$, also its principal square root will belong to $IS(p, g, t)$. Therefore, if in Step 4 the algorithm correctly selects the principal square root, the total computation will be done in the initial segment $IS(p, g, t)$. As Step 4 is executed at most $|p|$ times, the probability that the index of y will be found correctly is greater than $(1 - 1/(2|p|))^{|p|} > \frac{1}{2}$ (consider the Taylor series expansion around $x = 0$ of the function $f(x) = (1 - 1/x)^{|p|}$). It is easy to see that the whole computation is polynomial in $|p|$. Q.E.D.

We are now ready to prove Theorem 3.

Proof of Theorem 3. The following probabilistic poly $(|p|)$ time algorithm finds $\text{index}_g(y) \bmod p$ for *any* $y \in Z_p^*$ by only making calls to the oracle $MB_Q[p, g, \cdot]$. Set

$$t = \text{trials}\left(\frac{1}{Q(|p|)}, \frac{1}{2|p|}\right).$$

Recall that

$$IS(p, g, t) = \left\{ g^x \bmod p \,\middle|\, x \in \left[0, \frac{p-1}{t}\right] \right\}.$$

The algorithm makes use of the variables i, w, $index(w)$, and *candidate.*

Step 0 (Initialization)

$i := 1$

Step 1 (guess that $\text{index}(y) \in [i(p-1)/t, (i+1)(p-1)/t]$ and map y into the t-initial segment)

$w := y \cdot g^{-i(p-1)/t} \bmod p$

Step 2 (If $w \in IS(p, g, t)$, 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)

$candidate := index(w) + i(p-1)/t$; if $g^{candidate} \bmod p = y$ then HALT: *candidate* is the index of y in base g. Else continue.

Step 4 (keep on guessing)

$i := i+1$. If $i > t$ then $i := 1$ and go to Step 0; else go to Step 1. Q.E.D.

4. A concrete CSPRB generator based on the discrete logarithm problem. Let us describe an implementation, based on the intractability assumption for the DLP, of our general algorithmic scheme for constructing CSPRB generators.

We first recall a recent and powerful result due to Erich Bach [3].

LEMMA 4 (Bach [3]). *There is a probabilistic algorithm that, on input $n \in N$, selects an integer k, together with its prime factorization, with uniform probability among the n-bits integers. The algorithm runs in expected* poly (n) *time.*

THEOREM 4. *Under the intractability assumption for the* DLP, *we can construct a* CSPRB *generator.*

Proof. Let S_{2n} be the set of the $2n$-bit integers i (leading bit $= 1$) such that the first n bits of i constitute a prime p, and the next n bits (leading bit possibly 0) a generator g for Z_p^*. Let "$\sim$" denote concatenation. For $i \in S_{2n}$, $i = p \sim g$, set $D_i = Z_p^*$ and, for $x \in Z_p^*$, set $B_i(x) = B_{p,g}(x)$. We show that the set of predicates $B = \{B_i | i \in S_{2n}\}$ is an accessible, unapproximable set of predicates.

B is accessible. a) With uniform probability, we can select, among all the n-bits primes, a prime p together with the factorization of $p-1$, in probabilistic poly (n) time.

Select, with uniform probability, an n-bit integer k, together with its prime factorization, until $k+1$ is a prime. By Lemma 4, k can be selected in expected poly (n) time. $k+1$ can be tested for primality in random poly (n) time (see Solovay and Strassen [32]) and it will be a prime after expected $O(n)$ random selections of k because of the prime number theorem. If the prime $p = k+1$ has been so selected, it has been selected with uniform probability.

b) To generate a triplet (p, g, x) such that p is an n-bit prime, g a generator for Z_p^* and $x \in Z_p^*$, with uniform probability we follow the following algorithm:

(1): Generate p as in (a).

(2): Flip $2n$ fair coins; if the first n outcomes of the flips constitute a generator for Z_p^* and the second n outcomes constitute an $x \in Z_p^*$ then halt, the desired triplet has been selected, else go to (1).

As all triplets (p, g, x) so generated have the same probability of being selected it remains to show that the above algorithm runs fast. For this, note that, for all n-bit primes p, the probability of generating an $x \in Z_p^*$ is greater than $\frac{1}{2}$. Also, for all the n-bit primes p, the probability of constructing a generator for Z_p^* by flipping n fair coins is greater than $1/(12 \log_e \log_e (p-1))$. In fact, for all p, the generators for Z_p^* are $\varphi(p-1)$ (where φ is Euler totient function) and Rosser and Schoenfield [28] prove that $\varphi(k) > 1/(6 \log_e \log_e k)$ for $k > 3$. Moreover, as we have the factorization of $p-1$ as well, it is easy to check whether g is a generator for Z_p^* (see [30]).

B is unapproximable. By contradiction. Assume that there are polynomials P_1 and P_2 such that for $n \in F$, F infinite, there is a $P_1(n)$-size circuit C_n that evaluates

$B_{p,g}(x)$ correctly for a fraction of at least $\frac{1}{2}+1/P_2(n)$ of the n-bit inputs p, g, and x. Then a counting argument shows that there would be a fraction at least $1/P_2(n)$ of pairs (p, g) for which the circuit C_n evaluates $B_{p,g}(x)$ correctly for at least a fraction $\frac{1}{2}+1/(2 \cdot P_2(n))$ of the $x \in Z_p^*$. A trivial modification of Theorem 3 would then show that there is a probabilistic poly (n) time algorithm A, with oracle C_n, that for each $n \in F$ solves the DLP for at least a fraction $1/P_2(n)$ of the n-bit primes p. This would violate the intractability assumption for the DLP as, for some polynomial P_3, for each $n \in F$, A could be transformed to a circuit with less than $P_3(n)$ gates.

B satisfies the hypothesis of Theorem 2. A seed is a pair $(i = p \sim g, x)$. Define $f_i(x) = g^x \bmod p$. Note that, given $x \in Z_p^*$, it is easy to check whether $g^x \bmod p$ is a principal square root: just check whether $x \leqq (p-1)/2$. The other properties trivially hold. Q.E.D.

Theorems 2, 3 and 4 imply that it is possible to stretch a short random seed into a long pseudo-random bit sequence such that any efficient strategy to predict better than 50–50 the next output bit can be easily transformed to a "small" circuit solving the discrete logarithm problem.

5. Random self-reducibility. The purpose of this section is to single out the notion of random self-reducibility that we hope will be useful to complexity theory.

The notion of reducibility (Cook [11], Karp [16] and Levin [19]) is central to complexity theory. Conjunctive self-reducibility has also played an important role (see Berman [6], Fortune [12], Meyer and Paterson [22] and the article of Mahaney [21] proving the conjecture of Berman and Hartmanis that no NP-complete set can be reduced to a sparse set unless P = NP).

We introduce the notion of random self-reducibility by distilling the properties of the reductions in § 3. Informally, these properties guarantee that, if the majority of the instances of a decision problem (even if we do not know which ones) can be efficiently answered correctly, then every individual instance can be efficiently answered correctly with arbitrarily high probability.

In the two definitions below, $B = \{B_i : D_i \to \{0,1\} | i \in S_n$, and $n \in N\}$ is an *accessible* set of predicates. ρ is the "reduction function" and σ the "interpretation function": using r, a sequence of coin tosses, ρ randomly maps instance x into instance y; σ, given the answer for y and the sequence of coin tosses r, tells us what the answer for x should be.

DEFINITION (*strong random self-reducibility*). Let $\rho : (i \in S_n, x \in D_i, r \in D_i) \to D_i$ and $\sigma : (i \in S_N, x \in D_i, r \in D_i, b \in \{0,1\}) \to \{0,1\}$ be polynomial time computable functions. We say that B is *strongly randomly self-reducible* if for all $i \in S_N$ and all $x \in D_i$:

a) $\rho(i, x, \cdot)$ is a permutation over D_i and

b) for all $r \in D_i$, $B_i(x) = \sigma(i, x, r, B(\rho(i, x, r)))$.

DEFINITION (*weak random self-reducibility*). Let $\rho : (i \in S_n, x \in D_i, r \in D_i) \to D_i$ be, as before, a polynomially computable function and $\sigma : (i \in S_N, x \in D_i, r \in D_i, \varepsilon \in (0,1), b \in \{0,1\}) \to \{0,1\}$ be a function computable in probabilistic poly $(|i|, \varepsilon^{-1})$ time. We say that B is *weakly randomly self-reducible* if for any polynomial Q, for all sufficiently large $i \in S_N$ and all $x \in D_i$:

a) $\rho(i, x, \cdot)$ is a permutation over D_i and

b) letting r be randomly selected in D_i,

$$\text{Prob}\,(B_i(x) = \sigma(i, x, r, B_i(\rho(i, x, r)))) > \tfrac{1}{2} + 1/Q(|i|).$$

Acknowledgments. We are proud to thank many friends.

We are grateful to Shafi Goldwasser for having suggested the discrete logarithm

as a suitable one-way function for our purposes and 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 gained a great deal of concision and clarity due to the elegant result of Erich Bach [3].

We benefitted highly from the insightful comments of Lenore Blum, Steve Cook, Faith Fich, Zvi Galil, Donald Johnson, Leonid Levin, David Lichtenstein, Mike Luby, Gary Miller, Andrew Odlizko, Joan Plumstead, Charlie Rackoff, Ron Rivest, Jeff Shallit, Mike Sipser and Po Tong.

REFERENCES

[1] L. ADLEMAN, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 55–60.

[2] L. ADLEMAN, K. MANDERS AND G. MILLER, *On taking roots in finite fields*, Proc. 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 175–177.

[3] E. BACH, *How to generate random integers with known factorization*, Proc. 15th ACM Symposium on Theory of Computing, 1983.

[4] H. BEKER AND F. PIPER, *Cipher Systems*, Northwood, 1982.

[5] E. BERLEKAMP, *Factoring polynomials over large finite fields*, Math. Comp., 24 (1970), pp. 713–735.

[6] P. BERMAN, *Relationship between density and deterministic complexity of* NP*-complete languages*, 5th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, 62, Springer-Verlag, New York, 1978, pp. 63–71.

[7] M. BLUM, *How to exchange (secret) keys*, Proc. 15th ACM Symposium on Theory of Computing, 1983.

[8] ———, *Three applications of the oblivious transfer*, unpublished manuscript, 1981.

[9] L. BLUM, M. BLUM AND M. SHUB, *A simple secure pseudo-random number generator*, Proc. CRYPTO-82, Allen Gersho, ed.; this Journal, to appear.

[10] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 112–117.

[11] S. COOK, *The complexity of theorem proving procedures*, Proc. 3rd ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[12] S. FORTUNE, *A note on sparse complete sets*, this Journal, 8 (1979), pp. 431–433.

[13] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption and how to play mental poker keeping secret all partial information*, Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 365–377, Probabilistic Encryption, J. Comp. Sys. Sci., to appear.

[14] S. GOLDWASSER, S. MICALI AND P. TONG, *Why and how to establish a private code on a public network*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pt. 134–144.

[15] S. GOLDWASSER, S. MICALI AND A. YAO, *Strong signature schemes and authentication*, Proc. 15th ACM Symposium on Theory of Computing, 1983.

[16] R. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum, New York, 1972, pp. 85–103.

[17] D. KNUTH, *The Art of Computer Programming: Vol. 2 Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1981.

[18] A. KOLMOGOROV, *Three approaches to the concept of "the amount of information"*, Probl. of Inf. Transm., 1/1, 1965.

[19] L. A. LEVIN, *Universal sequential search problems*, Probl. of Inf. Transm., 9/3 (1973), pp. 265–266.

[20] M. LUBY, S. MICALI AND C. RACKOFF, *The MiRackoLus exchange of a secret bit*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983.

[21] S. MAHANEY, *Sparse complete sets for* NP*: a solution of a conjecture of Berman and Hartmanis*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1980, pp. 54–59.

[22] A. MEYER AND M. PATERSON, *With what frequency are apparently intractable problems difficult?*, Massachusetts Institute of Technology, Tech. Report, Cambridge, MA, Feb. 1979.

[23] G. MILLER, *Riemann's hypothesis and tests for primality*, J. Comp. Sys. Sci., 13 (1976), pp. 300–317.

[24] G. PETERSON, *Succint representations, random strings and complexity classes*, Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 86–95.

[25] J. PLUMSTEAD, *Inferring a sequence generated by a linear congruence*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 153–159.

[26] S. POHLIG AND M. HELLMAN, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Trans. Information Theory, IT-24 (1978), pp. 106–110.
[27] R. RIVEST, A. SHAMIR AND L. ADLEMAN, *On digital signatures and public key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.
[28] J. ROSSER AND L. SCHOENFIELD, *Approximate formulas for some functions of prime numbers*, Illinois J. Math., 6 (1962), pp. 64–94.
[29] A. SHAMIR, *On the generation of cryptographically strong pseudo-random sequences*, 8th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, 62, Springer-Verlag, New York, 1981.
[30] D. SHANKS, *Solved and Unsolved Problems in Number Theory*, Chelsea, London, 1978.
[31] M. SIPSER, *Three approaches to a definition of finite state randomness*, unpublished manuscript, 1979.
[32] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, this Journal, 6 (1977), pp. 84–85.
[33] A. YAO, *Theory and applications of trapdoor functions*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982.
[34] ANDREW ODLIZKO, private communication, 1984.