MIT/LCS/TR-336

SOME IMPLICATIONS OF COMPLEXITY THEORY ON PSEUDO-RANDOM BIT GENERATION

Stephen Trilling

SOME IMPLICATIONS OF COMPLEXITY THEORY ON PSEUDO-RANDOM BIT GENERATION

by

STEPHEN TRILLING

B.S., Yale University
(1980)

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the
Requirements for the
Degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1985

Signature of Author____
Department of Electrical Engineering and Computer Science
January 18, 1985

Certified by____
Prof. Michael Sipser
Thesis Supervisor

Accepted by____
Prof. Arthur C. Smith
Chairman, Departmental Graduate Committee

SOME IMPLICATIONS OF COMPLEXITY THEORY ON PSEUDO-RANDOM BIT GENERATION

by

STEPHEN TRILLING

Submitted to the Department of Electrical Engineering and Computer Science
on January 18, 1985 in partial fulfillment of the
requirements for the Degree of Master of Science

ABSTRACT

A recent area of interest in theoretical computer science has been in the construction of so-called pseudo-random bit generators. These generators "stretch" a short sequence of truly random bits into a longer sequence of "pseudo-random" bits. These bits are sufficiently indistinguishable from truly random bits to be useful in deterministic simulation of probabilistic computation.

Let us say, informally, that a function is *one-way* if it can be computed in polynomial time but no family of polynomial-size circuits can invert it with high probability. Yao [Y] has recently proven that if any such function exists, then it can be used to construct a pseudo-random bit generator. Furthermore, the existence of this generator implies that $\mathcal{R} \subseteq \bigcap_{\epsilon>0} DTIME(2^{n^{\epsilon}})$.

No proofs of the results have previously appeared in print. In this thesis, we present proofs of these results. In addition, we consider two other types of one-way function. The first type is much weaker than Yao's and we show that if such a function exists, it can be used to generate a somewhat less powerful pseudo-random bit generator. We then consider a second, much stronger type of one-way function and show that if such a function exists, then pseudo-random bit generators can be constructed which imply $\mathcal{R} \subseteq \mathcal{P}$.

Keywords: randomness, probabilistic computation, one-way function

Thesis Supervisor: Prof. Michael Sipser

Title: Associate Professor of Mathematics

ACKNOWLEDGEMENTS

I feel quite certain that not one word of this thesis would ever have been written without the support and guidance I received from Mike Sipser, my advisor. Mike never lost faith in me, even when I felt like giving up. Shafi Goldwasser was generous with both her time and insights in a number of helpful conversations. I also wish to thank Albert Meyer for his help during all of my time as a graduate student.

Ray Hirschfeld was a frequent late night companion and also an invaluable aid to me in unraveling the many mysteries of the system. Thang Bui and Miller Maley provided timely advice on using the typesetting system.

Finally, I thank my parents and brother and sister for their constant love, encouragement and understanding through the years.

TABLE OF CONTENTS

Chapter 1

Introduction

A major goal of theoretical computer science is to find "efficient" algorithms for solving various problems. In this context, "efficient" generally means that the algorithm runs in time bounded by some polynomial in the length of the problem instance. Unfortunately, finding such algorithms has so far been very difficult to accomplish in the case of many practical problems. Equally useful to the theoretician, although perhaps not the programmer, is a proof that *no* efficient algorithm exists for a given problem. Success in this area has been even less common.

The theory of NP-completeness has provided an alternative in the latter case. Hundreds of practical problems have been shown to be NP-complete and this is generally taken as pervasive evidence that they are intractable.

Recently, an alternative to finding provably efficient methods for solving given problems has also been considered. Instead of looking for algorithms which *always* give the correct solution to a given problem, one tries to find algorithms which give the correct solution with a very high probability. These "random" or "probabilistic" methods have been the focus of much research.

A probabilistic algorithm relies on coin flips in order to make certain decisions and its behavior on any given input is therefore not deterministic. Adleman[A] was the first to explicitly define the class $\mathcal{R}$ ("random polynomial time") of those problems efficiently solvable by probabilistic algorithms. Informally, a problem is in $\mathcal{R}$ if there is a polynomial time algorithm which solves it with only a very small chance of error. The chance that a solution given by such an algorithm is *incorrect* is just the probability that some unlikely sequence of coin flips is produced during its execution.

Note that any sequence of K coin flips is easily represented as a sequence of K bits. Thus, any probabilistic computation can be simulated by a deterministic machine which simply runs the probabilistic computation over and over again, trying all possible bit sequences of coin flips. Any polynomial time random algorithm can therefore be simulated deterministically in exponential time.

Unfortunately, problems which require exponential time to solve are beyond the reach of even today's fastest computers. As a result, effort has been focused on methods for "stretching" a short sequence of truly random bits generated by coin flips into a longer sequence of "pseudo-random" bits, without performing any additional coin flips. Ideally, such "pseudo-random" bit sequences should be sufficiently indistinguishable from truly random sequencces to be useful in simulating probabilistic compuations.

So far, every such pseudo-random bit generator which has been proposed is based on an unproved assumption that some given problem is intractable. Blum and Micali were the first to demonstrate such a generator. Theirs is based on the assumed difficulty of solving the so-called "discrete logarithm problem". Subsequent generators have been based on the difficulty of factoring [G,GMT,Y] and the quadratic residuosity problem [BBS].

Recently, Yao [Y] has proven a much more general theorem concerning pseudo-random bit generators. Let us say, informally, that a function is *one-way* if it can be computed in polynomial time, but no family of polynomial-sized circuits can invert it with high probability. Yao states that if *any* such one-way function exists, it can be used to construct a pseudo-random bit generator. Furthermore, the sequences produced by such a generator are indeed useful for simulating general probabilistic computation;. they can be used to simulate any polynomial time probabilistic algorithm deterministically in sub-exponential time. It remains as a major open problem to demonstrate that any such one-way function actually exists.

This thesis presents proofs of Yao's results in Chapter 2. Proofs of the results have not previously appeared in print. In Chapter 3, we consider two other types of one-way functions. The first type, which is weaker than those of Yao's, cannot be inverted with high probability by any circuit of some *fixed* polynomial size. We show that if such a function exists, a somewhat less powerful pseudo-random bit generator can be constructed. We then consider a second type of one-way function which is very much stronger: no circuit of some fixed sub-exponential size can invert it without being mistaken frequently. We then show that if such a function exists it is possible to construct a bit generator which can be used to simulate any polynomial time probabilistic algorithm deterministically in polynomial time.

Chapter 2

Yao's Theorems

In this chapter, we discuss conditions under which good pseudo-random bit generators can be constructed. We will then show that such generators can be used to simulate probabilistic computations deterministically. The results are all due to Yao[Y].

2.1 Building Generators From One-Way Functions

Suppose G is a deterministic program which, given some k-bit sequence x (a "seed") as input, outputs some bit sequence $b_1, b_2, \ldots, b_{P(k)}$ where $P(k)$ is a polynomial. In order for G to serve our purposes as a useful pseudo-random bit generator we will require the following:

(1) G **is efficient.** The sequence $b_1, b_2, \ldots, b_{P(k)}$ is output in time polynomial in k.

(2) **The output of** G **is unpredictable.** Given the generator G, and the first i output bits $b_1, b_2, \ldots, b_i$ generated from some seed x *but not the seed x itself,* it is computationally infeasible to predict the $i+1$st bit in the sequence with a better than 50-50 chance.

This definition was first proposed by Blum and Micali [BM].

Let us make the notion of "unpredictability" mentioned in condition (2) above more formal:

Definition Let P be a polynomial, S_k a multiset consisting of $P(k)$-long bit sequences and $S = \bigcup_k S_k$. A *polynomial-size next-bit test* for S is a family of circuits $C = \{C_k^i\}$. Each circuit C_k^i has i Boolean inputs where $i < P(k)$, one Boolean output and size polynomial in k. On input the first i bits of a sequence s randomly selected from 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 S *passes the test* C if for every polynomial Q, all sufficiently large k and all $i < P(k)$:

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

We will refer to both C and C_k^i as next-bit tests.

Now we can state condition (2) of our definition of pseudo-random bit generators as follows:

(2) Let S_k be the multiset of sequences output by G on all k-bit seeds. Then S_k passes all polynomial-size next-bit tests. (Note that S_k may be a multiset since two different seeds might cause G to output the same sequence.)

We formally define a *Cryptographically strong pseudo-random bit generator* following Blum and Micali[BM]:

Definition Let P be a polynomial, I_k the set of all strings of length k, and $D_k \subseteq I_k$ a set of inputs (or "seeds") of length k. Let G be a deterministic algorithm which, on input a seed $x \in D_k$, outputs a $P(k)$-long bit sequence s_x in $Poly(k)$ time. Let $S_k = \{s_x | x \in D_k\}$. The algorithm G is a *Cryptographically strong pseudo-random bit generator* (a *P-CSB* generator) if the multiset $S = \bigcup_k S_k$ passes all next-bit tests.

Every explicit CSB generator which has so far been proposed is based on an unproved assumption that some given problem is intractable. Blum and Micali [BM] were the first to show such a generator. They based it on the assumed difficulty of solving the discrete logarithm problem. For further details on the discrete logarithm problem, see [AL].

Definition Let p be a prime. The set of integers [1,p-1] forms a cyclic group Z_p^* under multiplication mod p. Given a prime p, a generator g for Z_p^*, and $y \in Z_p^*$, the *Discrete Logarithm Problem* (DLP) is to find the unique $x \in Z_p^*$ such that $y = g^x$ (mod p). This x is often denoted $index_{p,g}(y)$.

Let $C = \{C_n\}$ be a family of circuits such that C_n has $3n$ Boolean inputs and size $P(n)$ for some polynomial P. Think of these inputs as consisting of an n-bit prime p, an n-bit generator g for Z_p^*, and an n-bit $y \in Z_p^*$. Blum and Micali are able to construct a CSB generator under the (unproved) assumption that for every such family of circuits $C = \{C_n\}$ and all polynomials P and Q: for all sufficiently large n, $C_n(p, g, y) \neq index_{p,g}(y)$ for at least a fraction $1/Q(k)$ of the n-bit primes p. Some subsequent generators have been based on the difficulty of factoring [G,GMT,Y] and the quadratic residuosity problem [BBS].

Although the Discrete Logarithm Problem seems difficult, the inverse of the DLP is easily solved in polynomial time. Given a prime p, a generator g for Z_p^*, and $x \in Z_p^*$ the function POWER$(g, x, p) = g^x$ (mod p) can be calculated by successive squaring in time polynomial in the length of g, x, and p. Thus POWER can be thought of as a "one-way" function–easy to compute but difficult to invert. Yao[Y] has subsequently shown that given *any* "one-way" function, it is possible to construct a CSB generator. Let us make this notion of "one-way" more precise.

Definition(Yao[Y]) Let I_k be the set of all k-bit strings, let $D_k \subseteq I_k$ and let $f_k : D_k \mapsto D_k$ be a sequence of permutations. We will write $D = \bigcup D_k$ and $f = \{f_k\}$. Then, f is a *weak* one-way function if the following properties are satisfied:

(1) The domain is accessible: there exists a probabilistic polynomial-time algorithm which, on input k, selects an $x \in D_k$ with uniform probability.

(2) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $f_k(x)$

(3) There exists a polynomial Q such that the following holds. Let $C = \{C_k\}$ be any family of polynomial-size circuits where each C_k has k inputs. Then for all sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{Q(k)} \text{ of the } x \in D_k.$$

We can now state an important result of Yao's precisely.

Theorem 1 (Yao[Y]) Given any weak one-way function f, it is possible to construct a P-CSB generator for any polyomial P.

Note the generality of this theorem. It says that given *any* weak one-way function, we can construct a generator which "stretches" a k-bit seed into a $P(k)$-bit pseudorandom sequence, for *any* polynomial P. Thus we can decide beforehand, up to a polynomial, how much "stretching" we want our generator to do.

Note also one slight difference between the difficulty assumed about inverting a general one-way function and the difficulty which Blum and Micali assume about inverting the POWER function. In the general definition, one assumes that there exists *some* polynomial Q such that any family of polynomial-size circuits will fail to invert a one-way function. with probability at least $1/Q(k)$. Blum and Micali assume that this holds for *every* polynomial Q. It is, of course, possible to weaken the Blum-Micali assumption to conform with the more general case. The resulting generator is somewhat less efficient than that in the Blum-Micali paper.

Proof of Theorem 1 We first show the following lemma, due to Blum and Micali[BM], which provides a set of sufficient conditions for constructing CSB generators. We then show that, given *any* weak one-way function, it is possible to satisfy these conditions.

Lemma 1.1(Blum,Micali[BM]) Let I_k be the set of k-bit strings and let $D_k \subseteq I_k$. Let $g_k: D_k \mapsto D_k$ be a sequence of permutations and let $B_k: D_k \mapsto \{0,1\}$ be a sequence of predicates. We will write $D = \bigcup_k D_k$, $g = \{g_k\}$ and $B = \{B_k\}$. If the following set of conditions hold, then it is possible to construct a P-CSB generator, for any polynomial P.

(1) The domain is accessible: there exists a probabilistic polynomial-time algorithm which, on input k, chooses $x \in D_k$ with uniform probability.

(2) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $g_k(x)$.

(3) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $B_k(g_k(x))$.

(4) Let $C = \{C_k\}$ be any family of polynomial-size circuits such that each C_k has k inputs and let Q be any polynomial. Then, for all sufficiently large k:

$$C_k(x) \neq B_k(x) \text{ for at least a fraction } \frac{1}{2} - \frac{1}{Q(k)} \text{ of the } x \in D_k.$$

Proof of Lemma 1.1 First we construct the P-CSB generator and then prove that its outputs must pass all next-bit tests.

Choose an appropriate value of k to be the seed length and choose in probabilistic polynomial time a random $x \in D_k$ to be used as the seed. Set $c = P(k)$, the desired length of the output sequence, and generate the bits:

$$B_k(g_k(x)), B_k(g_k^2(x)), \ldots, B_k(g_k^c(x)).$$

The notation g_k^j indicates the j-fold composition of g_k. Now, output these bits in reverse order, i.e.:

$$B_k(g_k^c(x)), B_k(g_k^{c-1}(x)), \ldots, B_k(g_k(x)).$$

It should be clear that all of this can be accomplished in polynomial time by conditions (2) and (3).

It remains to show that the sequences output by this generator pass all next-bit tests. Suppose that this is not true. Then there exists a polynomial Q and a family of $Poly(k)$ size circuits $C = \{C_k^i\}$ where each C_k^i has $i < P(k)$ inputs and the following holds. For each of infinitely many values of k there exists some i such that:

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

where $p_{k,i}^C$ is the probability that the circuit C_k^i outputs the correct $i+1$st bit of a sequence when given the first i bits as input.

We now construct a polynomial-size family of circuits $A = \{A_k\}$ where each A_k has k inputs and such that for infinitely many values of k, A_k correctly computes the predicate $B_k(x)$ with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$. This contradicts condition (4).

Choose one of the infinitely many values of k such that for some $i < P(k)$:

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

The circuit A_k uses C_k^i as a "subroutine". On input $x \in D_k$, the circuit A_k first generates the i-bit sequence:

$$B_k(g_k^i(x)), B_k(g_k^{i-1}(x)), \ldots, B_k(g_k(x))$$

and inputs this sequence to the next-bit test circuit C_k^i. Note that this can be accomplished with a polynomial number of circuit gates by conditions (2) and (3). The circuit A_k then outputs whatever value C_k^i outputs on these bits.

Note that the bits:

$$B_k(g_k^i(x)), B_k(g_k^{i-1}(x)), \ldots, B_k(g_k(x))$$

are the first i bits of the CSB sequence:

$$B_k(g_k^i(x)), \ldots, B_k(g_k(x)), B_k(x), \ldots, B_k(g_k^{i-c}(x)).$$

Since the $i+1$st bit of this sequence is $B_k(x)$, the circuit A_k will correctly compute $B_k(x)$ whenever the circuit C_k^i correctly outputs the $i+1$st bit. Furthermore, the seed of this sequence is $g_k^{i-c-1}(x)$ and since g_k is, by assumption, a permutation, so is g_k^{i-c-1}. Thus, g_k^{i-c-1} generates all possible seeds $x \in D_k$ which means that the next-bit circuit C_k^i will correctly output the $i+1$st bit of this particular sequence for a fraction at least $\frac{1}{2}+\frac{1}{Q(k)}$. Thus, the circuit A_k computes $B_k(x)$ for a fraction at least $\frac{1}{2}+\frac{1}{Q(k)}$ of the $x \in D_k$ which contradicts condition (4). This completes the proof of the lemma.

It remains to show that, given some arbitrary one-way function $f = \{f_k\}$ over an accessible domain $E = \bigcup_k E_k$, it is possible to construct a new domain $D = \bigcup_k D_k$, a function $g = \{g_k\}$ and a predicate $B = \{B_k\}$ which satisfy the four conditions of lemma 1.1. This is proven in the following lemma due to Yao.

Lemma 1.2(Yao[Y]) Let $f = \{f_k\}$ be a weak one-way function over an accessible domain $E = \bigcup_k E_k$. Then, by definition, for each k, given any $x \in D_k$ it is possible to compute $f_k(x)$ in polynomial time and there exists some constant d with the following property. Given any family of polynomial-size circuits $C = \{C_k\}$ where C_k has k inputs, we have, for all sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{k^d} \text{ of the } x \in D_k.$$

The following construction satisfies conditions (1)-(4) of lemma 1.1:

(a) Set D_k to be the cartesian product of ck copies of E_k where c is a constant which depends on k and the constant d mentioned in the statement of this lemma. The exact value of c will be determined later in the proof. Suffice for now to say that it is polynomial function of k. Formally:

$$D_k = \{\, (x_1, x_2, \ldots, x_{ck}) \mid x_1 \in E_k, \ldots, x_{ck} \in E_k \,\}.$$

(b) Let $g_k(\,(x_1, x_2, \ldots, x_{ck})\,) = (\,f_k(x_1), f_k(x_2), \ldots, f_k(x_{ck})\,)$ where each $x_j \in E_k$.

(c1) Let $B_k^i(x) =$ the ith bit of $f_k^{-1}(x)$ where $x \in E_k$.

(c2) Let

$$B_k(\,(x_1, x_2, \ldots, x_{ck})\,) = \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{c} B_k^i(x_{c(i-1)+j})$$

where each $x_j \in D_k$ and "$\oplus$" denotes the "exclusive-or" operation.

Proof of Lemma 1.2 It should be fairly obvious that conditions (1), (2) and (3) of lemma 1.1 are satisfied by this construction. The domain D_k is certainly accessible since, by assumption, E_k is accessible. Given any $(x_1, x_2, \ldots, x_{ck}) \in E_k$, the function $g_k(\,(x_1, x_2, \ldots, x_{ck})\,)$ can be computed in polynomial time since, by assumption, each $f_k(x_j)$ is computable in polynomial time and, as mentioned, the constant c is polynomial in k. Finally, the predicate:

$$\begin{aligned} B_k(\, g_k(x_1, x_2, \ldots, x_{ck})\,) &= B_k(\,(f_k(x_1), f_k(x_2), \ldots, f_k(x_{ck}))\,) \\ &= \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{c} B_k^i(\, f_k(x_{c(i-1)+j})\,) \end{aligned}$$

can easily be computed in polynomial time since for any $x \in D_k$:

$$\begin{aligned} B_k^i(f_k(x)) &= \text{the } i\text{th bit of } f_k^{-1}(f_k(x)) \\ &= \text{the } i\text{th bit of } x \end{aligned}$$

The more difficult task is to prove that, as constructed, the predicate B_k satisfies condition (4) of lemma 1.1. Let $C = \{C_k\}$ be a family of polynomial-size circuits where each circuit C_k takes inputs from the domain D_k constructed in (a) above. For *every* such family of circuits $\{C_k\}$ and *every* polynomial $Q(k)$, we will show, for all sufficiently large k:

$$C_k((x_1, x_2, \ldots, x_{ck})) \neq B_k((x_1, x_2, \ldots, x_{ck}))$$
$$\text{for at least a fraction } \frac{1}{2} - \frac{1}{Q(k)} \text{ of the } (x_1, x_2, \ldots, x_{ck}) \in D_k.$$

Actually, we will choose to formulate this problem slightly differently. For every family of circuits $\{C_k\}$ and every polynomial $Q(k)$, we will show, for all sufficiently large k:

$$\text{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}$$

where $(x_1, x_2, \ldots, x_{ck})$ is a randomly chosen element of D_k. This will be proven through a sequence of three lemmas.

Note one small point. In order to be absolutely consistent, we should actually show that the probability of any circuit C_k correctly solving the predicate B_k is less than $\frac{1}{2} + \frac{1}{Q(ck^2)}$ since the length of each element in D_k is ck^2 bits. However, since $Q(ck^2)$ is polynomial in $Q(k)$, it suffices to prove the result as stated.

Lemma 1.2.1 Suppose that there exists a polynomial-size family of circuits $C = \{C_k\}$ where each circuit C_k takes inputs from the domain D_k and such that for *all* k and *all* $(x_1, x_2, \ldots, x_{ck}) \in D_k$ we have:

$$C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck})).$$

Then, there exists a polynomial-size family of circuits $A = \{A_k\}$ where each A_k has k Boolean inputs such that for *all* k and *all* $x \in E_k$ we have:

$$A_k(x) = f_k^{-1}(x).$$

This lemma just says that if we can easily solve the predicate B_k on all inputs, then we can easily invert the assumed one-way function f_k on all inputs. Although we do not actually need the lemma in order to show our desired result, its proof will shed some light on why the fact that f_k is hard to invert implies that B_k is difficult to compute.

Proof of Lemma 1.2.1 Let us examine the predicate B_k a bit more closely:

$$\begin{aligned}
B_k(\,(x_1, x_2, \ldots, x_{ck})\,) &= \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{c} B_k^i(x_{c(i-1)+j}) \\
&= B_k^1(x_1) \oplus B_k^1(x_2) \oplus \cdots \oplus B_k^1(x_c) && (1)\\
&\oplus B_k^2(x_{c+1}) \oplus B_k^2(x_{c+2}) \oplus \cdots \oplus B_k^2(x_{2c}) && (2)\\
&\oplus B_k^3(x_{2c+1}) \oplus B_k^3(x_{2c+2}) \oplus \cdots \oplus B_k^3(x_{3c}) && (3)\\
&\cdot && \cdot\\
&\cdot && \cdot\\
&\oplus B_k^k(x_{(k-1)c+1}) \oplus B_k^k(x_{(k-1)c+2}) \oplus \cdots \oplus B_k^k(x_{kc}). && (k)
\end{aligned}$$

Note that the predicate B_k^1 is applied to inputs x_1 through x_c (line (1)), B_k^2 is applied to inputs x_{c+1} through x_{2c} (line (2)), and so on, down to line (k). Thus, $B_k(\,(x_1, x_2, \ldots, x_{ck})\,)$ is just a big exclusive-or of the first bit of $f_k^{-1}(x_j)$ for $j = 1$ to c, the second bit of $f_k^{-1}(x_j)$ for $j = c+1$ to $2c$, and so on, down to the kth bit of $f_k^{-1}(x_j)$ for $j = (k-1)c+1$ to kc.

So, suppose we are given some $x \in D_k$ and wish to compute $f_k^{-1}(x)$. We will calculate each bit of $f_k^{-1}(x)$ separately. To get the first bit, choose some random elements $x_2 \in E_k, x_3 \in E_k, \ldots, x_{ck} \in E_k$ and calculate u where:

$$u = C_k(\,(x, f_k(x_2), f_k(x_3), \ldots, f_k(x_{ck}))\,).$$

(Remember the circuit C_k correctly computes the predicate B_k on all inputs.) This is just a big exclusive-or of $B_k^1(x)$ (i.e., the *first* bit of $f_k^{-1}(x)$) and a series of terms of the form $B_k^i(f_k(x_j))$. Recall that $B_k^i(f_k(x_j))$ is just the ith bit of x_j, thus the exclusive-or of all of these latter terms can be computed in polynomial time. More formally, if the exclusive-or of all the $B_k^i(f_k(x_j))$ terms is denoted v then:

$$\begin{aligned}
v &= \bigoplus_{j=2}^{c} B_k^1(f_k(x_j)) \bigoplus_{i=2}^{k} \bigoplus_{j=1}^{c} B_k^i(f_k(x_{c(i-1)+j})) \\
&= \bigoplus_{j=2}^{c} (\text{first bit of } x_j) \bigoplus_{i=2}^{k} \bigoplus_{j=1}^{c} (i\text{th bit of } x_{c(i-1)+j})
\end{aligned}$$

and v is computable in polynomial time. Thus, if we exclusive-or u with v then everything is "cancelled out" with the exception of $B_k^1(x)$ which is just the first bit of $f_k^{-1}(x)$ as desired. Formally:

$$\begin{aligned}
u \oplus v &= B_k^1(x) \\
&= \text{the first bit of } f_k^{-1}(x).
\end{aligned}$$

Since C_k is, by assumption, a polynomial-size circuit and v is polynomial-time computable, it is possible to get the first bit of $f_k^{-1}(x)$ with a polynomial number of circuit gates.

It should be clear that the second bit of $f_k^{-1}(x)$ can be computed similarly. Suppose we choose $x_1 \in E_k, \ldots, x_c \in E_k, x_{c+2} \in E_k, \ldots, x_{ck} \in E_k$ and compute:

$$C_k((f_k^{-1}(x_1), \ldots, f_k^{-1}(x_c), x, f_k^{-1}(x_{c+2}), \ldots, f_k^{-1}(x_{ck}))).$$

This is just a big exclusive-or of $B_k^2(x)$ (the *second* bit of $f_k^{-1}(x)$) and a series of terms of the form $B_k^i(f_k(x_j))$ each of which can be calculated in polynomial time. Thus, using the same trick as before, it is possible to "isolate out" the second bit of $f_k^{-1}(x)$ with a polynomial number of circuit gates. The remaining bits can be computed similarly. The following algorithm can easily be transformed into a polynomial-size circuit A_k with k Boolean inputs such that for all $x \in E_k$ we have:

$$A_k(x) = f_k^{-1}(x).$$

Algorithm A1

```
input: $x \in E_k$
output: $y \in E_k$ such that $y = f_k^{-1}(x)$
    (1) Set $y = null$
    (2) For $h = 1$ to $k$ do
            (2.1) Choose $x_1, x_2, \ldots, x_{c(h-1)}, x_{c(h-1)+2}, \ldots, x_{ck}$
                  all elements of $E_k$
            (2.2) Set $u = C_k((f_k(x_1), \ldots, f_k(x_{c(h-1)}), x, f_k(x_{c(h-1)+2}), \ldots, f_k(x_{ck})))$
            (2.3) Set $v = \bigoplus_{i=1}^{h-1} \bigoplus_{j=1}^{c}(i\text{th bit of } x_{c(i-1)+j})$
                      $\bigoplus_{j=2}^{c} (h\text{th bit of } x_{c(h-1)+j})$
                      $\bigoplus_{i=h+1}^{k} \bigoplus_{j=1}^{c}(i\text{th bit of } x_{c(i-1)+j})$
            (2.4) Set $w = u \oplus v$
            (2.5) Set $y = w \circ y$ ($\circ$ denotes concatenation)
    (3) Output $y$
```

Note that this algorithm need not actually choose a new set of $ck - 1$ elements in E_k each time it calculates a new bit, as specified in line (2.1). This has been done here for clarity of exposition. In fact, when this algorithm is converted into the circuit A_k, any set of $ck - 1$ elements in E_k can simply be "built-in" to this circuit. These same elements can be reused to get each of the k bits of $f_k^{-1}(x)$. This completes the proof of Lemma 1.2.1. Since f_k is assumed to be a one-way function, we have, in fact just shown:

Corollary 1.2.1 No family of polynomial-size circuits $C = \{C_k\}$ can compute the predicate $B = \{B_k\}$ on all inputs.

Before going on to the next lemma, we introduce some notation. We can think of each $(x_1, x_2, \ldots, x_{ck}) \in D_k$ as consisting of k "groups", each containing c elements. The first "group" consists of $x_1, x_2, \ldots, x_c$, the second "group" consists of $x_{c+1}, x_{c+2}, \ldots, x_{2c}$, and so on until the kth group: $x_{(k-1)c+1}, x_{(k-1)c+2}, \ldots, x_{kc}$. As we saw before, the value of $B_k(x_1, x_2, \ldots, x_{ck})$ is just a big exclusive-or of terms of the form $B_k^1(x_j)$ for each x_j in the first group, $B_k^2(x_j)$ for each x_j in the second group, and so on, until $B_k^k(x_j)$ for each x_j in the kth group. Thus, B_k is just a big exclusive-or of the ith bit of $f_k^{-1}(x_j)$ for each x_j in the ith group, where i ranges from 1 to k.

Definition Let $C = \{C_k\}$ be a family of circuits where each C_k takes inputs from the domain D_k. Think of the circuit C_k as trying to compute the predicate B_k. Now, for each $x \in E_k$ define the probability $s_{i,k}^C(x)$ as follows:

$$s_{i,k}^C(x) = \text{Prob}[\,C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))]$$

where $(x_1, x_2, \ldots, x_{ck})$ is randomly drawn from the elements of E_k
which have x in the ith group.

In other words, suppose that $(x_1, x_2, \ldots, x_{ck})$ has x in the ith group, i.e., $x =$ at least one x_j for $j = (i-1)c+1$ to ic. Then, $s_{i,k}^C(x)$ is just the probability that C_k will correctly compute the predicate B_k on the input $(x_1, x_2, \ldots, x_{ck})$. Intuitively, if $s_{i,k}^C(x)$ is large, then we have a good chance of being able to get the ith bit of $f_k^{-1}(x)$ by repeatedly using the circuit C_k. If $s_{i,k}^C(x)$ is large for *every* value of i from 1 to k, then we have a good chance of being able to get *all* the bits of $f_k^{-1}(x)$ by repeatedly using the circuit C_k. This last statement is made precise in the following lemma.

Lemma 1.2.2 Let $C = \{C_k\}$ be a family of polynomial-size circuits where each circuit C_k takes inputs from the domain D_k. Suppose that there exists a polynomial $V(k)$ and a constant d such that for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[\, s_{i,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ for } \textit{every} \text{ value of } i \text{ from 1 to } k \right] \geq 1 - \frac{1}{k^d}.$$

Then, there exists a family of polynomial-size circuits $A = \{A_k\}$ where each A_k has k Boolean inputs and for all sufficiently large k:

$$A_k(x) = f_k^{-1}(x) \text{ for at least a fraction } 1 - \frac{1}{k^d} \text{ of the } x \in E_k.$$

In other words, there exists a family of polynomial-size circuits which invert f_k with high probability if, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s_{1,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ and } s_{2,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)}, \ldots, \text{ and } s_{k,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)} \right] \geq 1 - \frac{1}{k^d}.$$

We will show in the proof of this lemma that if $s_{i,k}^C(x)$ is large then we can,with a polynomial-size circuit, correctly compute the ith bit of $f_k^{-1}(x)$. Thus, if $s_{i,k}^C(x)$ is large for *every* value of i from 1 to k, then we can, with a polynomial-size circuit, compute all of $f_k^{-1}(x)$. The lemma simply says that if, for all sufficiently large k $s^C_{i,k}(x)$ is large for *most of the* $x \in E_k$, then we can, for all sufficiently large k , invert f_k on *most of the* $x \in E_k$.

Let us first see how to compute the *first* bit of $f_k^{-1}(x)$ for some x which has the property that $s_1^C(x) \geq \frac{1}{2} + \frac{1}{V(k)}$. On all inputs $(x_1, x_2, \ldots, x_{ck}) \in E_k$ which have x in the first group we know that:

$$\text{Prob}[\,C_k(\,(x_1, x_2, \ldots, x_{ck})\,) = B_k(\,(x_1, x_2, \ldots, x_{ck})\,)\,] \geq \frac{1}{2} + \frac{1}{V(k)}.$$

Given $(x_1, x_2, \ldots, x_{ck}) \in E_k$, there are c elements in the first group namely, $x_1, x_2, \ldots, x_c$. Among these elements, let us say that x_j occupies "position" j. Then, since C_k correctly computes the predicate B_k with probability at least $\frac{1}{2} + \frac{1}{V(k)}$ whenever x is in the first group, there must be some position j from 1 to c such that C_k correctly computes B_k with probability at least $\frac{1}{2} + \frac{1}{V(k)}$ whenever x is in position j. Since this position will be "built-in" to the circuit which computes $f_k^{-1}(x)$, we may assume, without loss of generality that it is position 1.

Now, just as in Lemma 1.2.1, we begin by choosing some random elements $x_2 \in E_k, x_3 \in E_k, \ldots, x_{ck} \in E_k$ and compute u where:

$$u = C_k(\,(x, f_k(x_2), f_k(x_3), \ldots, f_k(x_{ck})\,).$$

Also just as before, we compute v, a big exclusive-or of terms of the form $B_k^i(f_k(x_j))$:

$$\begin{aligned} v &= \bigoplus_{j=2}^{c} B_k^1(f_k(x_j)) \bigoplus_{i=2}^{k} \bigoplus_{j=1}^{c} B_k^i(f_k(x_{c(i-1)+j})) \\ &= \bigoplus_{j=2}^{c} (\text{first bit of } x_j) \bigoplus_{i=2}^{k} \bigoplus_{j=1}^{c} (i\text{th bit of } x_{c(i-1)+j}) \end{aligned}$$

Finally, we calculate $u \bigoplus v$. If C_k correctly computes B_k on *all* inputs (as assumed in Lemma 1.2.1) then, as we saw earlier, $u \bigoplus v$ is the first bit of $f_k^{-1}(x)$. Here, however, we know only that C_k succeeds with *high probability* when x is in position 1. In fact, C_k may have computed the predicate B_k *incorrectly* on the chosen input, in which case $u \bigoplus v$ may well *not* be the correct first bit of $f_k^{-1}(x)$. However, if we recompute $u \bigoplus v$ a number of times using a new set of inputs $x_2, x_3, \ldots, x_{ck}$ each time, we should expect to get the correct first bit about $\frac{1}{2} + \frac{1}{V(k)}$ of the time. Saying it differently, if the average value of $u \bigoplus v$ is at least $\frac{1}{2} + \frac{1}{V(k)}$ then the correct first bit is probably 1, whereas if this average value is at most $\frac{1}{2} - \frac{1}{V(k)}$ then the correct bit is probably 0.

This method of repeated recalculation using different inputs is known as "sampling" and each recalculation is called a "trial". Clearly, the more trials we perform, the more certain we will be about which bit to choose. Let *avg* be the average value of

$u \oplus v$ computed during sampling to determine the first bit of $f_k^{-1}(x)$. Let *avgtrue* be the value of *avg* which we would obtain if we performed a trial for every possible distinct set of $ck - 1$ values $x_2, x_3, \ldots, x_{ck}$ (i.e., *avgtrue* is the *true* value which *avg* is estimating). Now, suppose that we perform enough trials to be certain that *avg* is within $1/(2V(k))$ of avg_t. Then, if the first bit is actually 1, we must have:

$$avg \geq \frac{1}{2} + \frac{1}{V(k)} - \frac{1}{2V(k)}$$

so

$$avg \geq \frac{1}{2} + \frac{1}{2V(k)}.$$

Alternatively, if the first bit actually 0, we must have:

$$1 - avg \geq \frac{1}{2} + \frac{1}{V(k)} - \frac{1}{2V(k)}$$

so

$$avg \leq \frac{1}{2} - \frac{1}{2V(k)}.$$

Thus, if we perform enough trials to be *certain* that *avg* is within $1/(2V(k))$ of *avgtrue* then we can, *with certainty*, choose the correct first bit of $f_k^{-1}(x)$. We show below that this can be achieved with only a polynomial number of trials.

It should be clear that, for any i, if $s_{i,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)}$ then we can, with certainty, compute the ith bit of $f_k^{-1}(x)$ using the same method. Thus, for each $x \in E_k$ such that $s_{i,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)}$ for *every* i from 1 to k, we can compute $f_k^{-1}(x)$ with a polynomial-size circuit. Since the hypothesis of this lemma assumes that, for infinitely many values of k, this is true for at least $1 - \frac{1}{k^d}$ of the $x \in E_k$, we can construct a family of polynomial size circuits which, for infinitely many values of k, compute $f_k^{-1}(x)$ for at least $1 - \frac{1}{k^d}$ of the $x \in E_k$.

It remains only to prove that we can calculate *avg* to within $1/(2V(k))$ of *avgtrue* with only a polynomial number of trials. If we perform t trials of the sampling procedure describe above to compute *avg* then, by the Central Limit Theorem (see [HPS]) we have:

$$\text{Prob}(\,|avg - avgtrue| \geq c\,) \approx 2(1 - \Phi(\delta))$$

where

$$\delta = 2c\sqrt{t}$$

and

$$\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du.$$

Note that we can never be certain our estimate is within the bound c but only sure with arbitrarily high probability. If we wish to know how many trials are necessary in order to be sure with probability ϵ that *avg* is within a bound c of *avgtrue*, we must solve the following equation for t:

$$2(1 - \Phi(2c\sqrt{t})) < \epsilon. \tag{1}$$

In fact, when our procedure is converted into a circuit, we will be able to "build-in" a particular set of t sets of inputs which can be used in sampling to assure that our estimate *avg* is off by at most $1/(2V(k))$. By calculations which can be found in the Appendix, equation (1) above is satisfied if t is chosen such that:

$$t > \frac{\ln 2 - \ln \epsilon}{2c^2}.$$

Here, we want $c = 1/(2V(k))$ and for reasons which will be made clear below, we set $\epsilon = 1/2^{2k}$ giving:

$$t > (\ln 2 - \ln(2^{-2k}))(2V(k)).$$

This is certainly satisfied by:

$$t \geq 6kV(k).$$

We now show that, since t is polynomial in k, we can build a *Poly*(k) size circuit which correctly computes $f_k^{-1}(x)$ for every $x \in E_k$ such that $s_{i,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)}$. The argument is probabilistic. Suppose that $t = 6kV(k)$ and let

$$W = \{\text{input}_1, \text{input}_2, \ldots, \text{input}_t\}$$

be a set of t randomly chosen inputs where each input_i is a set of $ck - 1$ elements of E_k which will be used for sampling,i.e., for each i:

$$\text{input}_i = \{x_1, x_2, ..., x_{ck-1}\} \text{ each } x_j \in E_k.$$

For a randomly chosen string $x \in E_k$ such that $s_{i,k}^C(x) \geq \frac{1}{2} + \frac{1}{V(k)}$ for every value of i from 1 to k, let us say that the set W *fails on* x if, for at least one value of i, the value of of *avg* computed while trying to get the ith bit of $f_k^{-1}(x)$ differs from *avgtrue* by more than $1/(2V(k))$. Since the chance of failing on any given bit of $f_k^{-1}(x)$ is less than $1/2^{2k}$ (because $\epsilon = 1/2^{2k}$ above) and there are k bits we get:

$$\text{Prob}[W \text{ fails on } x] < \frac{k}{2^{2k}}.$$

Since there are at most 2^k elements in E_k:

$$\text{Prob}[W \text{ fails on } \textit{at least one } x \in E_k] < \frac{k2^k}{2^{2k}}.$$

Thus:

$$\text{Prob}[W \text{ does } \textit{not} \text{ fail on } \textit{any } x \in E_k] \geq 1 - \frac{k2^k}{2^{2k}} > 0.$$

So, there must exist some set W of t inputs which can be used in sampling to assure that for every $x \in E_k$ such that $s^C_{i,k}(x) \geq \frac{1}{2} + \frac{1}{V(k)}$, the value of $f_k^{-1}(x)$ can be correctly computed with certainty. This set W can simply be "built-in" to the circuit A_k which inverts f_k. The entire algorithm for computing $f_k^{-1}(x)$ follows on the next page.

Let x be such that $s^C_{i,k}(x) \geq \frac{1}{2} + \frac{1}{V(k)}$. Then, by definition, the circuit C_k correctly computes the predicate B_k with probability at least $\frac{1}{2} + \frac{1}{V(k)}$ on all inputs $(x_1, x_2, \ldots, x_{ck}) \in D_k$ which have x in the ith group. As before, given such an input, we say that element x_j occupies "position" j in the input. Since the ith group consists of elements $x_{c(i-1)+1}$ through x_{ci}, there must exist some position j from $c(i-1)+1$ to ic such that C_k computes B_k with probability at least $\frac{1}{2} + \frac{1}{V(k)}$ whenever x is in position j. This position will, for each i, be "built-in" to the circuit A_k which inverts f_k. We have assumed in the following algorithm, without loss of generality, that it is position $c(i-1)+1$, i.e., the first position of each group.

Algorithm A2

```
input: $x \in E_k$
output: $y \in E_k$ such that $y = f_k^{-1}(x)$ with probability $1 - \frac{1}{h^d}$
        (1) Set $y = null$
        (2) For $h = 1$ to $k$ do
                (2.1) Set $count = 0$
                (2.2) Set $t = 6kV(k)$
                (2.3) For $sample = 1$ to $t$ do
                        (2.3.1) Choose next "built-in" input for sampling:
                                $x_1, x_2, \ldots, x_{c(h-1)}, x_{c(h-1)+2}, \ldots, x_{ck}$
                        (2.3.2) Set $u = C_k((f_k(x_1), \ldots, f_k(x_{c(h-1)}), x, f_k(x_{c(h-1)+2}), \ldots, f_k(x_{ck})))$
                        (2.3.3) Set $v = \oplus_{i=1}^{h-1} \oplus_{j=1}^{c}$(ith bit of $x_{c(i-1)+j}$)
                                $\oplus_{j=2}^{c}$ (hth bit of $x_{c(h-1)+j}$)
                                $\oplus_{i=h+1}^{k} \oplus_{j=1}^{c}$(ith bit of $x_{c(i-1)+j}$)
                        (2.3.4) Set $count = u \oplus v$
                (2.4) Set $avg = \frac{count}{t}$
                (2.5) If $avg \geq \frac{1}{2} + \frac{1}{2V(k)}$ then set $w = 1$
                (2.6) Elseif $avg \leq \frac{1}{2} - \frac{1}{2V(k)}$ then set $w = 0$
                (2.7) Set $y = w \circ y$ ($\circ$ denotes concatenation)
        (3) Output $y$
```

It should be clear that this algorithm can easily be converted into a polynomial size circuit A_k with k Boolean inputs. This completes the proof of Lemma 1.2.2. Note that f_k is assumed to be a one-way function which means that, for some constant d, *no* family of polynomial-size circuits can invert it with probability at least $1 - \frac{1}{k^d}$. Thus we have actually shown:

Corollary 1.2.2 Let $f = \{f_k\}$ be a polynomial time computable function which takes inputs from the domain E_k and suppose that there exists a constant d such that any family of polynomial-size circuits which tries to invert f_k is, for infinitely many k, correct for less than $1 - \frac{1}{k^d}$ of the $x \in E_k$. Let $C = \{C_k\}$ be any family of polynomial-size circuits where each circuit C_k takes inputs from the domain D_k (i.e., each circuit C_k is trying to compute the predicate B_k). Let $V(k)$ be any polynomial.

Then, for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s_{i,k}^{C}(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ for every value of } i \text{ from 1 to } k \right] < 1 - \frac{1}{k^d}.$$

The following lemma completes the proof of Lemma 1.2 and therefore also completes the proof of Theorem 1.

Lemma 1.2.3 Let I_k be the set of all k-bit strings and let $f = \{f_k\}$ be a weak one-way function over a domain $E_k \subseteq I_k$. Thus, there exists a constant d such that any family of polynomial-size circuits which tries to invert f_k is, for infinitely many k, correct for less than $1 - \frac{1}{k^d}$ of the $x \in E_k$. Let D_k be a new domain consisting of the cartesian product of k^{2d} copies of E_k, i.e.:

$$D_k = \{ (x_1, x_2, \ldots, x_{k^{2d}}) \mid x_1 \in E_k, \ldots, x_{k^{2d}} \in E_k \}$$

and let B_k be the predicate over the domain D_k which is described in Lemma 1.2. Note that we are finally giving the the value of the constant c which has been used until now to specify inputs in D_k; c is equal to k^{2d-1}. Now, let $C = \{C_k\}$ be any family of polynomial-size circuits where each circuit C_k takes inputs from the domain D_k and tries to compute the predicate B_k. Let $Q(k)$ be any polynomial. Then, for all sufficiently large k we have, for a randomly chosen $(x_1, x_2, \ldots, x_{k^{2d}}) \in D_k$:

$$\text{Prob}[\, C_k((x_1, x_2, \ldots, x_{k^{2d}})) = B_k((x_1, x_2, \ldots, x_{k^{2d}}))] < \frac{1}{2} + \frac{1}{Q(k)}.$$

Proof Let $V(k)$ be any polynomial. Then, from Corollary 1.2.2, for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s_{i,k}^{C}(x) < \frac{1}{2} + \frac{1}{V(k)} \text{ for } \textit{some} \text{ value of } i \text{ from 1 to } k \right] > \frac{1}{k^d}.$$

Then, as shown by Goldwasser[G], there exists a polynomial $Q(k) = aV(k)$ for some constant a such that, for a random $(x_1, x_2, \ldots, x_{k^{2d}}) \in D_k$:

$$\text{Prob}[\, C_k((x_1, x_2, \ldots, x_{k^{2d}})) = B_k((x_1, x_2, \ldots, x_{k^{2d}}))] < \frac{1}{2} + \frac{1}{Q(k)}.$$

This is a somewhat complicated counting argument so the proof is omitted here. Details can be found in [G].

2.2 Statistical Tests for Strings

It turns out that in order to be useful for simulating general probabilistic computation, we would like the output from a "good" pseudo-random bit generator to be unpredictable in a slightly different way from that discussed so far. Until now, we have required that, given any prefix of a sequence output by our generator, it be computationally impossible to consistently predict the next bit in the sequence. Consider a somewhat different notion of unpredictability: Given some n-bit sequence output by the generator and some truly random n-bit sequence, it is computationally infeasible to distingush the two sequences with a better than 50-50 chance. Note that this kind of unpredictability involves the impossibility of distinguishing between two *sequences* whereas the next-bit test involves the impossibility of predicting a given *bit*. The following definition formalizes this:

Definition Let P be a polynomial, S_k a multiset consisting of $P(k)$-long bit sequences and $S = \bigcup_k S_k$. A *polynomial-size statistical test for strings* is a family of circuits $C = \{C_k\}$. Each circuit C_k has $P(k)$ Boolean inputs, one Boolean output and size polynomial in $P(k)$. The multiset S *passes the test* C if for every polynomial Q, and all sufficiently large k:

$$|p_k^S - p_k^R| < \frac{1}{Q(k)}$$

where p_k^S denotes the probability that C_k outputs 1 on a randomly selected $s \in S_k$ and p_k^R denotes the probability that C_k outputs 1 on a randomly selected $P(k)$-long bit sequence. We will refer to both C and C_k as statistical tests.

It turns out, as proven in the next theorem of Yao's, that these two notions of unpredictability are equivalent. Thus, the output of any CSB generator passes all polynomial-size statistical tests.

Theorem 2 (Yao[Y]) Let P be a polynomial, S_k a multiset consisting of $P(k)$-long bit sequences and $S = \bigcup_k S_k$. The set S passes all polynomial-size next-bit tests if and only if it passes all polynomial-size statistical tests.

Proof The easy half is to show that if S passes all polynomial-size statistical tests then it passes all polynomial-size next-bit tests. Suppose $S = \bigcup_k S_k$ fails a polynomial-size next-bit test. Then, by definition, there is a family of $Poly(k)$ size circuits $C = \{C_k^i\}$ such that each C_k^i has $i < P(k)$ Boolean inputs and one Boolean output. Furthermore, there exists a polynomial Q such that for each of infinitely many values of k there exists some i such that:

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

Recall that $p_{k,i}^C$ is the probability that the circuit C_k^i outputs the correct $i+1$st bit of a sequence $s \in S_k$ when given the first i bits of s as input.

We now show that S must fail a statistical test $A = \{A_k\}$ where each circuit A_k has $P(k)$ inputs and size $Poly(P(k))$. This statistical test simply uses the next-bit test as a subroutine. Suppose k is chosen such that the next-bit test succeeds with high probability on strings in S_k. Then, there must be a value of i, such that the circuit C_k^i predicts the ith bit of strings in S_k with high probability. The statistical test circuit A_k operates as follows. It inputs the first i bits of its own input to the next-bit test circuit C_k^i. It then compares the bit predicted by C_k^i with the true $i+1$st bit of its own input. If the circuit C_k^i predicted correctly, then the statistical test A_k outputs a

1, otherwise it outputs a 0. Now, the next-bit test C_k^i succeeds with high probability given the first i bits of a string in S_k but can only succeedat most half the time on truly random i-bit strings. Thus, the statistical test A_k will effectively distinguish between sequences from S_k and truly random $P(k)$-bit sequences.

More formally, let p_k^S denote the probability that A_k outputs 1 on a randomly selected $s \in S_k$ and p_k^R denote the probability that A_k outputs 1 on a randomly selected $P(k)$-long bit sequence. Then, by the argument above, we get for infinitely many pairs k and i:

$$p_k^S = p_{k,i}^C$$

and

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

but

$$p_k^R = \frac{1}{2}$$

so

$$p_k^S - p_k^R \geq \frac{1}{Q(k)}.$$

Thus, S fails the statistical test $A = \{A_k\}$ and this half of the theorem is proven.

The more difficult task is proving that if $S = \bigcup S_k$ passes all next-bit tests then it passes all polynomial-time statistical tests. The idea is as follows. Suppose S fails some statistical test $C = \{C_k\}$. We want to construct a next-bit test for S. For infinitely many values of k, the circuit C_k can distinguish between elements of S_k and random $P(k)$-bit strings. The first obvious question is: Which of these $P(k)$ bits should our next-bit test attempt to predict? It turns out that it is possible to find a value of i which has the following interesting property: If the circuit C_k is given as input the first i bits of some string in $s \in S_k$ followed by a random sequence of $P(k) - i$ bits, it can detect with a certain probabilty whether the $i+1$st bit of its input is the correct $i+1$st bit of s. By repeating this experiment a number of times with different sequences of $P(k) - i$ bits, it is possible to ascertain the $i+1$st bit of s with a better than 50 percent chance. A circuit A_k^i which has the statistical test C_k "built in" can be constructed to repeat this experiment the appropriate number of times and thereby constitute a next-bit test.

Formally, suppose that a multiset $S = \bigcup_k S_k$ fails some polynomial-time statistical test. Then, by definition, there is a family of polynomial-size circuits $C = \{C_k\}$ such that C_k has $P(k)$ inputs and one output. Furthermore, there exists a polynomial Q such that for infinitely many values of k:

$$|p_k^S - p_k^R| \geq \frac{1}{Q(k)}$$

where p_k^S is the probability that C_k outputs 1 on a randomly selected sequence in S_k and p_k^R is the probability that C_k outputs 1 on a randomly selected $P(k)$-long bit sequence. We now show that S must necessarily fail a next-bit test.

Let $[x|i]y$ denote the concatenation of the first i bits of x with y. For each $i < P(k)$, let p_k^i denote the probability that $C_k([x|i]y) = 1$ where x is chosen randomly from S_k and y is a random string of $P(k) - i$ bits. Thus, p_k^i is the probability that the statistical test C_k outputs 1 when given as input the first i bits of some string in S_k followed by $P(k) - i$ random bits. Note that $p_k^0 = p_k^R$ and $p_k^{P(k)} = p_k^S$.

We now construct a family of circuits $A = \{A_k^i\}$ which constitute a next-bit test for S. Each A_k^i has $i < P(k)$ inputs and size polynomial in $P(k)$. Let k be chosen such that:

$$|p_k^S - p_k^R| \geq \frac{1}{Q(k)}.$$

Assume, without loss of generality, that

$$p_k^S - p_k^R \geq \frac{1}{Q(k)}.$$

Then, clearly, there must be a value of i such that

$$p_k^{i+1} - p_k^i \geq \left(\frac{1}{Q(k)}\right)\left(\frac{1}{P(k)}\right).$$

(Remember, i varies between 0 and $P(k)$.) The circuit A_k^i will correctly predict this $i + 1$st bit of any sequence in S_k with high probability.

Recall that p_k^{i+1} is the probability that C_k outputs 1 on any string consisting of an $i + 1$-bit initial segment of a string in S_k followed by $P(k) - (i + 1)$ random bits. Let q_k^{i+1} be the probability that C_k outputs 1 when given the first i bits of a string in S_k followed by the *incorrect* $i + 1$st bit, followed by $P(k) - (i + 1)$ random bits. Note that:

$$p_k^i = \frac{1}{2}p_k^{i+1} + \frac{1}{2}q_k^{i+1}$$

thus:

$$p_k^{i+1} - p_k^i = p_k^{i+1} - \left(\frac{1}{2}p_k^{i+1} + \frac{1}{2}q_k^{i+1}\right) \geq \left(\frac{1}{Q(k)}\right)\left(\frac{1}{P(k)}\right)$$

$$\frac{1}{2}p_k^{i+1} - \frac{1}{2}q_k^{i+1} \geq \left(\frac{1}{Q(k)}\right)\left(\frac{1}{P(k)}\right)$$

$$p_k^{i+1} - q_k^{i+1} \geq \frac{2}{Q(k)P(k)} \tag{1}$$

so:

$$p_k^{i+1} > q_k^{i+1}.$$

Consider in particular line (1). The fact that the difference between p_k^{i+1} and q_k^{i+1} is large means that if C_k is given as input the first i bits of some string $s \in S_k$ followed by $P(k) - i$ random bits, it can effectively distinguish between those strings containg the correct $i+1$st bit of s in their $i+1$st position, and those containing the incorrect $i+1$st bit.

Since $p_k^{i+1} + q_k^{i+1} = 1$ and $p_k^{i+1} > q_k^{i+1}$, if we can be sure of correctly predicting the $i+1$st bit of a randomly drawn string in S_k with probability p_k^{i+1}, then we will be correct more than half the time. The next-bit test A_k^i is constructed to do this.

So, suppose A_k^i is given as input $b_1, b_2, ..., b_i$, the first i bits of a randomly chosen sequence $s \in S_k$. Let b be the correct $i+1$st bit of s and let $\bar{b}$ be the incorrect $i+1$st bit. The circuit A_k^i first calculates the probabilities:

$$r_c = \text{Prob}\Big[C_k(b_1, \ldots, b_i, b, b_{i+2}, \ldots, b_{P(k)}) = 1\Big]$$

and

$$r_i = \text{Prob}\Big[C_k(b_1, \ldots, b_i, \bar{b}, b_{i+2}, \ldots, b_{P(k)}) = 1\Big]$$

where $b_{i+2}, \ldots, b_{P(k)}$ are chosen randomly. In fact, these probabilities can only be estimated with high accuracy but for the sake of clarity, let us assume for a moment that they can be calculated exactly. A discussion on estimating the probabilities appears later in the proof. Note that A_k^i does not actually know which is the correct and which the incorrect $i+1$st bit. It simply calculates the two probabilities–one of them will be r_c the other, r_i. The predicting circuit A_k^i now chooses $b_{i+1} = b$ with probability $\frac{r_c}{r_c+r_i}$ and chooses $b_{i+1} = \bar{b}$ with probability $\frac{r_i}{r_c+r_i}$.

Suppose we calculate the value of r_c for each i-bit prefix of a string in S_k and take the average of all these probabilities. This average is just the total probability with which the circuit A_k^i correctly predicts the $i+1$st bit of a randomly drawn string in S_k. This average is also equal to p_k^{i+1} so A_k^i predicts correctly with probability $p_k^{i+1} > \frac{1}{2}$. Thus there must exist some polynomial R such that A_k^i correctly chooses the $i+1$st bit for sequences in S_k with probability at least $\frac{1}{2} + \frac{1}{R(k)}$. The set S therefore fails the next-bit test $A = \{A_k^i\}$ and this part of the proof is finished. A discussion on estimating the probabilities r_c and r_i follows.

Estimating the probabilities

Recall the probabilities r_c and r_i which are used to predict the $i+1$st bit:

$$r_c = \text{Prob}\Big[C_k(b_1, \ldots, b_i, b, b_{i+2}, \ldots, b_{P(k)}) = 1\Big]$$

and

$$r_i = \text{Prob}\Big[C_k(b_1, \ldots, b_i, \bar{b}, b_{i+2}, \ldots, b_{P(k)}) = 1\Big]$$

where b is the correct $i+1$st bit and $\bar{b}$ is the incorrect $i+1$st bit. As mentioned above, these probabilities cannot be calculated exactly. Here we show that they can, however, be estimated closely enough so that the next-bit test will still succeed with more than a 50-50 chance.

The idea is as follows. Suppose the next-bit test circuit is given as input the bits $b_1, \ldots, b_i$, an i-bit prefix of some string in S_k. In order to estimate the probability r_c the next-bit circuit first chooses a sequence of random bits $b_{i+2}, ..., b_{P(k)}$ and then uses the statistical test C_k as a "subroutine" to calculate whether

$$C_k(b_1, b_2, \ldots, b_i, b, b_{i+2}, \ldots, b_{P(k)})$$

causes a 0 or a 1 to be output. The next-bit circuit then repeats this same procedure a number of times using different random sequences $b_{i+2}, \ldots, b_{P(k)}$. Finally, it takes the average of all the 0,1 values output by the circuit C_k and uses this as the estimate for r_c. The estimate for r_i is computed similarly. As we saw in lemma 1.2.2, this method is known as "sampling" and each repetition of the procedure is a "trial". Note that the next-bit circuit does not actually know whether $b = 0$ or $b = 1$. It simply does this "sampling" first with $b = 1$ and then with $b = 0$. One of the results will be an estimate for r_c, the other an estimate for r_i.

Clearly, the more trials we make the closer the resulting estimates will be to r_c and r_i. The question is, for each input of i bits, how far from r_c and r_i can these estimates be and still assure that the next-bit circuit will correctly predict the $i + 1$st bit more than fifty percent of the time?

Recall that p_k^{i+1} is the probability that the statistical test C_k outputs 1 when given as input the first $i + 1$ bits of a sequence in S_k followed by $P(k) - (i + 1)$ random bits. Also, q_k^{i+1} is the probability that C_k outputs 1 when given as input the first i bits of a sequence in S_k followed by the *incorrect* $i + 1$st bit, followed by $P(k) - (i + 1)$ random bits. Thus, p_k^{i+1} is just the average of all values of r_c taken over all i-bit prefixes and q_k^{i+1} is, similarly, the average of all values of r_i. If we could calculate each r_c and r_i *exactly*, our next-bit circuit would predict the $i + 1$st bit correctly with probability p_k^{i+1}. From above, we know that:

$$p_k^{i+1} - q_k^{i+1} \geq \frac{2}{P(k)Q(k)}$$

where P and Q are both polynomials. Thus, if each of the estimates for r_c and r_i is off by less than $1/(P(k)Q(k))$ then the resulting estimate for p_k^{i+1} will still be greater than the estimate for q_k^{i+1}. So, the next-bit circuit will still succeed more than half the time.

We must therefore show that with only *Poly*(k) trials, we can get estimates for r_c and r_i which are off by less than $1/(P(k)Q(k))$. This is necessary to insure that the next bit circuit has size polynomial in k as required.

Let p be a probability which is estimated by a value $\hat{p}$, calculated by performing t trials of the sampling procedure described above. Then, by the Central Limit Theorem (see [HPS]) we have:

$$\text{Prob}(|p - \hat{p}| \geq c) \approx 2(1 - \Phi(\delta))$$

where

$$\delta = 2c\sqrt{t}$$

and

$$\Phi(z)=\int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}}e^{-u^2/2}du.$$

Note that we can never be certain our estimate is within the bound c but only sure with arbitrarily high probability. If we wish to know how many trials are necessary in order to be sure with probability ϵ that our estimate is within the bound c, we must solve the following equation for t:

$$2(1-\Phi(2c\sqrt{t}))<\epsilon. \tag{1}$$

In fact, when our procedure is converted into a circuit, we will be able to "build-in" a particular set of t strings of bits which can be used in sampling to assure that our estimates are off by less than $1/(P(k)Q(k))$. By calculations which can be found in the Appendix, equation (1) above is satisfied by:

$$t>\frac{\ln 2-\ln\epsilon}{2c^2}.$$

As discussed above, we want $c=1/(P(k)Q(k))$ and, for reasons which will be made clear below, we set $\epsilon=1/2^{P(k)}$:

$$t>\frac{1}{2}(\ln 2-\ln(2^{-P(k)}))((P(k))^2(Q(k))^2).$$

This is certainly satisfied by:

$$t\geq(P(k))^2(Q(k))^2+(P(k))^3(Q(k))^2.$$

We will now show that if the number of trials t satisfies this inequality then the next-bit circuit can actually be built in such a way that, given *any* i-bit prefix of a string in S_k as input, the circuit can, *with certainty*, estimate r_c and r_i with an error of less than $1/(P(k)Q(k))$. The argument is probabilistic. Suppose that $t=(P(k))^2(Q(k))^2+(P(k))^3(Q(k))^2$ and let $W=\{w_1,\ldots,w_t\}$ be a set of random strings each of length $P(k)-(i+1)$. In other words each element w_i of W is a string of random bits:

$$w_i=b_{i+2},b_{i+3},\ldots,b_{P(k)}.$$

These strings will be used to do the sampling. Let x be a randomly chosen i-bit prefix of a string in S_k. We will say that the set W *fails on* x if, given x as input, at least one of the probabilities r_c and r_i cannot be estimated with an error of less than $1/(P(k)Q(k))$ when using all of the strings in W to do the sampling. Since the chance of failing on either r_c or r_i is less than $1/2^{P(k)}$ (because $\epsilon=1/2^{P(k)}$), we get:

$$\text{Prob}[W\text{ fails on a random }i\text{-bit prefix of a string in }S_k]<\frac{2}{2^{P(k)}}.$$

Since there are at most 2^i i-bit prefixes:

$$\text{Prob}[W \text{ fails on } \textit{at least one } i\text{-bit prefix of a string in } S_k] < \frac{2}{2^{P(k)}}(2^i).$$

Since the next-bit circuit is trying to predict the $i+1$st bit and strings in S_k have length $P(k)$, we know that i can be at most $P(k)-1$. Therefore,

$$\text{Prob}[W \text{ fails on at least one } i\text{-bit prefix of a string in } S_k] < 1.$$

So, there must exist at least one set W of t strings which does not fail on any inputs to the next-bit circuit. This set W will be "built into" the next-bit circuit, assuring that this circuit can always estimate the probabilities r_c and r_i with sufficient accuracy to be sure of predicting correctly more than half the time. Since t is polynomial in k, the next-bit circuit will have size polynomial in k as required. A complete algorithm for computing the next-bit is given on the next page. It should be clear that this procedure can be converted into a polynomial size next-bit test.

Algorithm A3

input: an i-bit prefix of a string $x \in S_k$ given as $b_1, \ldots, b_i$
output: $y = i + 1$st bit of x (with probability $\frac{1}{2} + \frac{1}{R(k)}$ for some polynomial $R(k)$)

```
(1) Set $count_0 = 0$
(2) Set $count_1 = 0$
(3) Set $t = (P(k))^2\,(Q(k))^2 + (P(k))^3\,(Q(k))^2$
(4) For $sample$ = 1 to $t$ do
    (4.1) Choose next "built-in" input for sampling:
          $b_{i+2}, b_{i+3}, \ldots, b_{P(k)}$
    (4.2) Set $count_0 = count_0 + C_k(b_1, \ldots, b_i, 0, b_{i+2}, \ldots, b_{P(k)})$
    (4.3) Set $count_1 = count_1 + C_k(b_1, \ldots, b_i, 1, b_{i+2}, \ldots, b_{P(k)})$
(5) Set $q_0 = \frac{count_0}{t}$
(6) Set $q_1 = \frac{count_1}{t}$
(7) Output $y = 0$ with probability $\frac{q_0}{q_0+q_1}$
    Output $y = 1$ with probability $\frac{q_1}{q_0+q_1}$
```

This concludes the section on estimating the probabilities as well as the proof of Theorem 2. Since the output of any CSB generator passes all next bit tests, we have shown:

Corollary 2 Let G be a CSB generator whose output consists of the set $S = \bigcup_k S_k$ as defined above. Then S passes all polynomial-size statistical tests.

2.3 Simulating Probabilistic Computation

It remains to explore the usefulness of CSB generators in simulating general probabilistic computation. It is here that the notion of statistical tests and the result of Theorem 2 will be helpful. First, we give two standard definitions from complexity theory.

Definition The class $DTIME(T(n))$ denotes the family of languages accepted by deterministic Turing machines which halt after at most $T(n)$ steps, where n is the input length.

Definition A language L is in the class $\mathcal{R}$ ("Random polynomial time") if and only if there exists a polynomial P and a Turing machine M_R such that:

(i) M_R takes two inputs x and y;input x is to be checked for membership in L and the meaning of input y will become clear below. The machine M_R runs in time $P(|x|)$.

(ii) if $x \notin L$ then for all y such that $|y| = P(|x|)$ we have Prob$\{M_R(x,y)$accepts$\} = 0$.

(iii) if $x \in L$ then for all y such that $|y| = P(|x|)$ we have Prob$\{M_R(x,y)$accepts$\} = \frac{1}{2}$.

The input y in the above definition can be thought of as a random sequence of bits or coin flips which may witness the membership of x in L. For the sake of clarity in our definition, the string y is given to M_R as input and the machine then computes a result deterministically which depends only on x and y. Intuitively, one may also choose to think of M_R as being given only x and actually generating the string y through some random selection process such as coin flips. In this setting, M_R is considered a "probabilistic" machine. Given some input x, if M_R outputs 1, we know for certain that $x \in L$ but if M_R outputs 0 we know only that $x \notin L$ with probability $\frac{1}{2}$. The computation of M_R on x can be repeated a number of times so that we can be sure of our answer with arbitrarily high probability.

In terms of our original definition, if we run $M_R(x,y)$ for each of the $2^{P(|x|)}$ possible values of y and the machine outputs 0 every time, then we know with total certainty that $x \notin L$. Thus, we have essentially shown the following:

Fact $\mathcal{R} \subseteq \bigcup_{\epsilon>0} DTIME(2^{n^\epsilon})$.

Proof Let $L \in \mathcal{R}$ be accepted by a probabilistic machine M_R in time $P(n)$ where P is a polynomial. The language L can be accepted by a deterministic machine M as follows. On input x machine M runs $M_R(x,y)$ on each of the $2^{P(|x|)}$ possible strings y of length equal to $P(|x|)$. The machine M accepts x if and only if there exists some value of y such that $M_R(x,y)$ accepts. Otherwise M rejects. It should be clear that M acepts exactly the language L in time $2^{Q(|x|)}$ for some polynomial Q. Thus $L \in \bigcup_{\epsilon>0} DTIME(2^{n^\epsilon})$.

The class $\bigcup_{\epsilon>0} DTIME(2^{n^\epsilon})$ is also known as "exponential time". Problems requiring exponential time are considered intractable since the cost of solving them by computer quickly becomes prohibitive. If we could somehow avoid having to test *every* string y of length $P(|x|)$ in our simulation of machine M_R, this simulation could obviously be sped up. A natural question at this point is whether the bit sequences output by a CSB generator are sufficiently "random" to be useful in this context. Yao has answered this question affirmatively by proving that such a generator can in fact be used to deterministically simulate a probabilistic machine and that the simulation requires only "sub-exponential" time.

Theorem 3 (Yao[Y]) Suppose that for any polynomial P, it is possible to construct a CSB generator which takes k-bit inputs and produces $P(k)$-bit outputs. Then $\mathcal{R} \subseteq \bigcap_{\epsilon>0} DTIME(2^{n^\epsilon})$.

Proof Let $L \in \mathcal{R}$ be accepted by a probabilistic machine M_R. Suppose that M_R runs in time n^j where n is the input length. We construct a deterministic machine M which accepts L in sub-exponential time. In the previous, exponential-time simulation, the machine M whould have tried *every possible* witness y of length n^j in simulating the probabilistic machine. There are 2^{n^j} such strings which is what forces the exponential time bound. Here, M will only try those witnesses of length n^j *which are output by the CSB generator*. Suppose, for example that the generator takes k-bit inputs and produces k^c-bit outputs. Thus, an input to the generator (a "seed") of length $n^{j/c}$ will produce an output of length n^j. The deterministic machine M inputs each string of length $n^{j/c}$ to the generator producing a possible witness y of length n^j. M simulates $M_R(x,y)$ for each such y and accepts if and only if M_R ever accepts. Since there are at most $2^{n^{j/c}}$ seeds of length $n^{j/c}$, the entire simulation can be carried out in time $2^{n^{d/c}}$ for some fixed d. By assumption, the generator can be constructed to "stretch" its inputs by *any* polynomial amount. Here, this means that the value of c can be chosen to be any fixed value. Thus, for *any* ϵ, the simulation can be carried out in time 2^{n^ϵ}.

Note that in our original definition of CSB generators (Lemma 1.2), we assumed only that each seed could be found in *probabilistic* polynomial time. Here, where we want to use the CSB generator for a *deterministic* algorithm, we simply try every string of length $n^{j/c}$ as an input to the generator. Some of these might not be actual seeds for the generator, hence the resulting output might not be a pseudo-random sequence. However, this method will cover every possible seed which is all that is necessary here.

It remains to show that the machine M accepts the same language L which is accepted by M_R. As usual, let I_k be the set of inputs to the generator of length k and let $S_k = \{s_x | x \in I_k\}$. We will show that if M does not accept L, then the set $S = \bigcup_k S_k$ output by the generator fails a polynomial-size statistical test. The relative ease of this proof should make clear the reason for introducing statistical tests as criteria for our generators.

Recall that if $x \in L$ then for a randomly chosen string y, $M_R(x,y)$ accepts with probability $\frac{1}{2}$. If $x \notin L$, then for *no* string y does $M_R(x,y)$ to accept. So if M does not accept the language L, there must be some $z \in L$ which M incorrectly rejects (i.e., none of the possible witnesses w output by the CSB generator cause $M_R(z,w)$ to accept even though $z \in L$.) Furthermore, there must exist infinitely many such strings $z \in L$ or else M could be repaired without increasing its running time.

Let z be any element of L which M fails to accept. Recall that the probabilistic machine M_R runs in time n^j where n is input length. So, on input z, the deterministic machine M will simulate $M_R(z,y)$ on all strings y of length $|z|^j$ output by the CSB generator. Thus M will have to try all seeds of length $|z|^{j/c}$ since the generator "stretches" k-bit inputs into k^c-bit outputs. For the remainder of the proof, we will write k in place of $|z|^{j/c}$.

For each of the infinitely many such strings z, there will be a circuit C_k with $k^c = (|z|^{j/c})^c = |z|^j$ inputs which has z "hard-wired" in. On any input y, the circuit C_k simulates $M_R(z,y)$. This circuit outputs 1 if and only if $M_R(z,y)$ acepts.

Since $y \in L$ we have, by the definition of $\mathcal{R}$, for a randomly chosen string y of length $|z|^j$:

$$p_k^R = \text{Prob}\{C_k(y) = 1\} = \frac{1}{2}.$$

However, since $M_R(z,y)$ does not accept for *any* $y \in S_k$, for a randomly chosen $y \in S_k$ we have:

$$p_k^S = \text{Prob}\{C_k(y) = 1\} = 0.$$

Thus $p_k^R - p_k^S = \frac{1}{2}$ so there is certainly a polynomial R such that $p_k^R - p_k^S > \frac{1}{R(k)}$. Therefore, the set S fails the statistical test $\{C_k\}$ contradicting our assumption that S is the output of a CSB generator. This concludes the proof.

Recall now Theorem 1 which said that given any weak one-way function, it is possible to construct a CSB generator which "stretches" its seeds by any polynomial amount. Combined with Theorem 3, we get:

Corollary 3 If there exists a weak one-way function, then $\mathcal{R} \subseteq \bigcap_{\epsilon>0} DTIME(2^{n^{\epsilon}})$.

Chapter 3

Refinements of Yao's Theorems

In this chapter we refine somewhat Yao's results presented in Chapter 2. More specifically, we assume the existence of two new kinds of one-way function, both in turn different from the weak one-way function described in Chapter 2. We then explore the types of pseudo-random bit generators which can be constructed from these functions.

3.1 A Low-Level Refinement

Recall that in Chapter 2 we defined a weak one-way function as being polynomial-time computable but difficult to invert by any family of polynomial-size circuits. Here, we consider an even more restricted definition. Informally, an *R-weak one-way function* is polynomial-time computable but difficult to invert by any family of $R(k)$-size circuits where $R(k)$ is some polynomial. A rigorous definition of these functions follows the definition of "O notation":

Definition Let $f(n)$ and $g(n)$ be functions. We say that:

$$f(n) = O(g(n))$$

if there exists some constant c such that for all sufficiently large values of n:

$$f(n) \leq cg(n).$$

Definition Let I_k be the set of all k-bit strings, let $D_k \subseteq I_k$ and let $f_k: D_k \mapsto D_k$ be a sequence of permutations. We will write $D = \bigcup D_k$ and $f = \{f_k\}$. Let $R(k)$ be some polynomial. Then, f is an R-weak one-way function (or simply an R-one-way function) if the following properties are satisfied:

(1) The domain is accessible: there exists a probabilistic polynomial-time algorithm which, on input k, selects an $x \in D_k$ with uniform probability.

(2) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $f_k(x)$.

(3) There exists a polynomial Q such that the following holds. Let $C = \{C_k\}$ be any family of circuits where each C_k has k inputs and size $O(R(k))$. Then for all

sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{Q(k)} \text{ of the } x \in D_k.$$

Just as we did in Chapter 2, it is possible to construct a pseudo-random bit generator from an R-one-way function. However, as might be expected, the output from such a generator cannot be expected to pass all polynomial-size next bit tests. Instead, we need a weaker notion of such a test, called an *RQ-next-bit test.* Intuitively, a set of strings passes an RQ-next-bit test if no family of size $R(k)$ circuits can, infinitely often, predict the next bit with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$.

Definition Let P, Q and R be fixed polynomials, S_k a multiset consisting of $P(k)$-long bit sequences and $S = \bigcup_k S_k$. An *RQ-next-bit tsst* for S is a family of circuits $C = \{C_k^i\}$. Each circuit C_k^i has i Boolean inputs where $i < P(k)$, one Boolean output and size $O(R(k))$. On input the first i bits of a sequence s randomly selected from 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 S *passes the test* C if for all sufficiently large k and all $i < P(k)$:

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

We will refer to both C and C_k^i as RQ-next-bit tests.

Note the differences between this definition and the definition of general next-bit tests given in Chapter 2. Here, we restrict both the size of the circuit doing the predicting, as well as the degree of accuracy to which it can predict. In general, we will expect that the polynomial R, the size of the next-bit circuit, will depend on the poynomial Q, the accuracy threshold. The reason for this will be made clear below. Finally, we define the analogous notion of the CSB generator, an *RQ-weak pseudo-random bit generator:*

Definition Let P, Q and R be fixed polynomials, I_k the set of all strings of length k, and $D_k \subseteq I_k$ a set of inputs (or "seeds") of length k. Let G be a deterministic algorithm which, on input a seed $x \in D_k$, outputs a $P(k)$-long bit sequence s_x in $Poly(k)$ time. Let $S_k = \{s_x | x \in D_k\}$. The algorithm G is a *Cryptographically RQ-weak pseudo-random bit generator* (or simply an *RQ-weak* generator) if the multiset $S = \bigcup_k S_k$ passes every RQ-next-bit test.

Recall that in Chapter 2 we showed, given a function which was difficult to invert with any polynomial-size circuit, how to construct a bit generator whose output passed all polynomial-size next-bit tests. Here, given a function which is difficult to invert with some fixed polynomial-size circuit, it would be nice to construct a generator whose output passed all next-bit tests of some, possibly different, fixed polynomial size. Unfortunately, using the techniques of Chapter 2, this does not seem possible. The logic behind the proof in Chapter 2 was that if there existed a polynomial-size next-bit test which succeeded with probability $\frac{1}{2} + \frac{1}{Q(k)}$ for some polynomial Q, then it would be possible to construct a polynomial-size circuit to invert the assumed one-way function on most inputs. Unfortunately, the size of this circuit depended on the polynomial

$Q(k)$. Thus if we assume the existence of a U-one-way function, it is only possible, using these methods, to construct a bit generator whose output passes all next-bit tests of some fixed size which succeed with probability less than $\frac{1}{2} + \frac{1}{Q(k)}$ for some *fixed* polynomial Q. We now state the main theorem of this section:

Theorem 4 Let I_k be the set of all strings of length k and let $f = \{f_k\}$ be a U-one-way function over some accessible domain $E_k \subseteq I_k$. Thus, there exists a constant d such that, given any family of circuits $C = \{C_k\}$ where each C_k has k Boolean inputs and size $O(U(k))$ we have, for all sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{k^d} \text{ of the } x \in D_k.$$

By definition, $f_k(x)$ can be computed in polynomial time for all $x \in E_k$ thus by a standard result from complexity theory (see [FP]) we may assume that there exists some polynomial $F(k)$ such that for each value of k, $f_k(x)$ can be computed by a circuit of size $F(k)$ for all $x \in E_k$. Then, for any polynomial P, it is possible to construct an RQ-weak generator which "stretches" seeds of length k into outputs of length $P(k)$, where:

$$R(k) = O\left(\frac{U(k)}{k^2 Q(k)} - F(k)k^{2d} - P(k)k^{2d}\right).$$

Proof We first show a set of conditions, very similar to those given in Lemma 1.1 which are sufficient for constructing weak generators. We then show how to satisfy these conditions, given any U-one-way function.

Lemma 4.1 Let I_k be the set of k-bit strings and let $D_k \subseteq I_k$. Let $g_k:D_k \mapsto D_k$ be a sequence of permutations and let $B_k:D_k \mapsto \{0,1\}$ be a sequence of predicates. We will write $D = \bigcup_k D_k$, $g = \{g_k\}$ and $B = \{B_k\}$. Let $P(k)$ be any polynomial. If the following set of conditions hold, then it is possible to construct an RQ-weak generator which "stretches" k-bit seeds into $P(k)$-bit outputs. The polynomial R is equal to:

$$R(k) = T(k) - P(k) size(B_k(g_k))$$

where $T(k)$, and $size(B_k(g_k))$ are defined below.

(1) The domain is accessible: there exists a probabilistic polynomial-time algorithm which, on input k, chooses $x \in D_k$ with uniform probability.

(2) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $g_k(x)$.

(3) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $B_k(g_k(x))$. Thus, $B_k(g_k(x))$ can be computed by a polynomial-size circuit. Let this circuit have size $size(B_k(g_k))$.

(4) Let $C = \{C_k\}$ be any family of circuits such that each C_k has k inputs and size $O(T(k))$ for some fixed polynomial T,and let Q be some fixed polynomial. Then, for all sufficiently large k:

$$C_k(x) \neq B_k(x) \text{ for at least a fraction } \frac{1}{2} - \frac{1}{Q(k)} \text{ of the } x \in D_k.$$

Note that the polynomial T may well be a function of the polynomial Q thus, as expected, R will probably depend on Q.

Proof of Lemma 4.1 First we construct the weak generator and then prove that its outputs must pass every RQ-next-bit test. The proof is very similar to that of Lemma 1.1.

Choose an appropriate value of k to be the seed length and choose in probabilistic polynomial time a random $x \in D_k$ to be used as the seed. Set $c = P(k)$, the desired length of the output sequence, and generate the bits:

$$B_k(g_k(x)), B_k(g_k^2(x)), \ldots, B_k(g_k^c(x)).$$

The notation g_k^j indicates the j-fold composition of g_k. Now, output these bits in reverse order, i.e.:

$$B_k(g_k^c(x)), B_k(g_k^{c-1}(x)), \ldots, B_k(g_k(x)).$$

It should be clear that all of this can be accomplished in polynomial time by conditions (2) and (3).

It remains to show that the sequences output by this generator pass every RQ-next-bit tests where:

$$R(k) = T(k) - P(k)size(B_k(g_k))$$

where $P(k)$ is the amount that the generator "stretches" k-bit seeds, $size(B_k(g_k))$ is the minimum size circuit computing $B_k(g_k(x))$, and the predicate B_k cannot be computed with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$ by any circuit of size $O(T(k))$.

Suppose that this is not true. Then there exists a family of circuits $C = \{C_k^i\}$ where each circuit C_k^i has $i < P(k)$ inputs, size $O(R(k))$ and the following holds. For each of infinitely many values of k there exists some i such that:

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

where $p_{k,i}^C$ is the probability that the circuit C_k^i outputs the correct $i + 1$st bit of a sequence when given the first i bits as input.

We now construct a family of circuits $A = \{A_k\}$ where each A_k has k inputs, size $O(T(k))$ and such that for infinitely many values of k, A_k correctly computes the predicate $B_k(x)$ with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$. This contradicts condition (4).

Choose one of the infinitely many values of k such that for some $i < P(k)$:

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

The circuit A_k uses C_k^i as a "subroutine". On input $x \in D_k$, the circuit A_k first generates the i-bit sequence:

$$B_k(g_k^i(x)), B_k(g_k^{i-1}(x)), \ldots, B_k(g_k(x))$$

and inputs this sequence to the next-bit test circuit C_k^i. This can be accomplished with $O(\, i(size(B_k(g_k)))\,)$ circuit gates by condition (3). The circuit A_k then outputs whatever value C_k^i outputs on these bits making the total size of A_k equal to:

$$\begin{aligned} size(A_k) &= O(\, i(size(B_k(g_k))) + R(k)\,) \\ &= O(\, i(size(B_k(g_k))) + T(k) - P(k)(size(B_k(g_k)))\,) \\ &= O(T(k))(\text{since } i < P(k)\,) \end{aligned}$$

By showing that the circuit A_k solves the predicate B_k with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$ we get a contradiction to condition (4).

Note that the bits:

$$B_k(g_k^i(x)), B_k(g_k^{i-1}(x)), \ldots, B_k(g_k(x))$$

are the first i bits of the CSB sequence:

$$B_k(g_k^i(x)), \ldots, B_k(g_k(x)), B_k(x), \ldots, B_k(g_k^{i-c}(x)).$$

Since the $i+1$st bit of this sequence is $B_k(x)$, the circuit A_k will correctly compute $B_k(x)$ whenever the circuit C_k^i correctly outputs the $i+1$st bit. Furthermore, the seed of this sequence is $g_k^{i-c-1}(x)$ and since g_k is, by assumption, a permutation, so is g_k^{i-c-1}. Thus, g_k^{i-c-1} generates all possible seeds $x \in D_k$ which means that the next-bit circuit C_k^i will correctly output the $i+1$st bit of this particular sequence for a fraction at least $\frac{1}{2} + \frac{1}{Q(k)}$. Thus, the circuit A_k computes $B_k(x)$ for a fraction at least $\frac{1}{2} + \frac{1}{Q(k)}$ of the $x \in D_k$ and has size $O(T(k))$ contradicting condition (4). This completes the proof of the lemma.

We must now show that given any U-weak one-way function over an accessible domain $E = \bigcup_k E_k$, it is possible to construct a new domain $D = \bigcup_k D_k$, a function $g = \{g_k\}$ and a predicate $B = \{B_k\}$ which satisfy the four conditions of Lemma 4.1.

Lemma 4.2 Let $f = \{f_k\}$ be a U-weak one-way function over an accessible domain $E = \bigcup_k E_k$. Then, by definition, there exists some constant d such that given any family of circuits $C = \{C_k\}$ where C_k has k inputs and size $O(U(k))$, we have, for all sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{k^d} \text{ of the } x \in D_k.$$

Furthermore, assume that $f_k(x)$ can be computed by a circuit of size $F(k)$. The following construction satisfies conditions (1)-(4) of lemma 4.1:

(a) Set D_k to be the cartesian product of k^{2d} copies of E_k. It will be less cumbersome to write ck instead of k^{2d} so we will generally choose to do this, as in Chapter 2. Thus, D_k is the cartesian product of ck copies of E_k where $c = k^{2d-1}$. Formally:

$$D_k = \{ (x_1, x_2, \ldots, x_{ck}) \mid x_1 \in E_k, \ldots, x_{ck} \in E_k \}.$$

(b) Let $g_k((x_1, x_2, \ldots, x_{ck})) = (f_k(x_1), f_k(x_2), \ldots, f_k(x_{ck}))$ where each $x_j \in E_k$.

(c1) Let $B_k^i(x) =$ the ith bit of $f_k^{-1}(x)$ where $x \in E_k$.

(c2) Let

$$B_k((x_1, x_2, \ldots, x_{ck})) = \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{c} B_k^i(x_{c(i-1)+j})$$

where each $x_j \in D_k$ and "$\oplus$" denotes the "exclusive-or" operation.

Proof of Lemma 4.2 It should be fairly obvious that conditions (1), (2) and (3) of Lemma 4.1 are satisfied by this construction. The domain D_k is certainly accessible since, by assumption, E_k is accessible. Given any $(x_1, x_2, \ldots, x_{ck}) \in E_k$, the function $g_k((x_1, x_2, \ldots, x_{ck}))$ can be computed in polynomial time since, by assumption, each $f_k(x_j)$ is computable in polynomial time. Finally, the predicate:

$$\begin{aligned} B_k(g_k(x_1, x_2, \ldots, x_{ck})) &= B_k((f_k(x_1), f_k(x_2), \ldots, f_k(x_{ck}))) \\ &= \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{c} B_k^i(f_k(x_{c(i-1)+j})) \end{aligned}$$

can easily be computed in polynomial time since for any $x \in D_k$:

$$\begin{aligned} B_k^i(f_k(x)) &= \text{the } i\text{th bit of } f_k^{-1}(f_k(x)) \\ &= \text{the } i\text{th bit of } x \end{aligned}$$

In fact, for any $(x_1, x_2, \ldots, x_{ck}) \in D_k$, the predicate $B_k(((x_1, x_2, \ldots, x_{ck})))$ can be computed by a circuit of size $O(ck) = O(k^{2d})$ since it simply consists of k^{2d} exclusive-ors. Thus, the value here of $size(B_k(g_k))$ in condition (3) above is:

$$size(B_k(g_k)) = O(k^{2d}).$$

The more difficult task is to prove that, as constructed, the predicate B_k satisfies condition (4) of Lemma 4.1. In particular, we must show that for an appropriately chosen polynomial $T(k)$ the following holds. Let $C = \{C_k\}$ be any family of circuits such that each circuit C_k takes inputs from the domain D_k constructed in (a) above. and has size $O(T(k))$. Let $T(k)$ be set to:

$$T(k) = \frac{U(k)}{k^2 Q(k)} - F(k)k^{2d}.$$

were $Q(k)$ is a polynomial. We will show that for every such family of circuits and all sufficiently large k:

$$C_k((x_1, x_2, \ldots, x_{ck})) \neq B_k((x_1, x_2, \ldots, x_{ck}))$$
$$\text{for at least a fraction } \frac{1}{2} - \frac{1}{Q(k)} \text{ of the } (x_1, x_2, \ldots, x_{ck}) \in D_k$$

Actually, we will choose to formulate this problem slightly differently. For every such family of circuits $\{C_k\}$ and every we will show, for all sufficiently large k:

$$\text{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}$$

where $(x_1, x_2, \ldots, x_{ck})$ is a randomly chosen element of D_k. This will be proven through a sequence of two lemmas. First, we repeat the definition of the probability $s^C_{i,k}$ defined in Chapter 2. For details on what a "group" is, see page 15.

Definition Let $C = \{C_k\}$ be a family of circuits where each C_k takes inputs from the domain D_k. Think of the circuit C_k as trying to compute the predicate B_k. Now, for each $x \in E_k$ define the probability $s^C_{i,k}(x)$ as follows:

$$s^C_{i,k}(x) = \text{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))]$$
where $(x_1, x_2, \ldots, x_{ck})$ is randomly drawn from the elements of E_k which have x in the ith group.

Lemma 4.2.1 Let $C = \{C_k\}$ be a family of circuits where each circuit C_k takes inputs from the domain D_k. Suppose that there exists a polynomial $V(k)$ and a constant d such that for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s^C_{i,k}(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ for } \textit{every} \text{ value of } i \text{ from 1 to } k \right] \geq 1 - \frac{1}{k^d}.$$

If each circuit C_k has size:

$$O\left(\frac{U(k)}{k^2 V(k)} - k^{2d} F(k) \right).$$

Then, there exists a family of circuits $A = \{A_k\}$ where each A_k has k Boolean inputs, size $O(U(k)$ and for all sufficiently large k:

$$A_k(x) = f_k^{-1}(x) \text{ for at least a fraction } 1 - \frac{1}{k^d} \text{ of the } x \in E_k.$$

Proof The main details of the proof are identical to the proof of Lemma 1.2.2 in Chapter 2. The only remaining step necessary here is to show that, under the assumptions of this lemma, Algorithm A2 on page 20 can be converted into a circuit A_k of size $O(U(k))$ We reproduce this algorithm on the next page.

Algorithm A2

```
input: $x \in E_k$
output: $y \in E_k$ such that $y = f_k^{-1}(x)$ with probability $1 - \frac{1}{h^d}$
    (1) Set $y = null$
    (2) For $h = 1$ to $k$ do
        (2.1) Set $count = 0$
        (2.2) Set $t = 6kV(k)$
        (2.3) For $sample = 1$ to $t$ do
            (2.3.1) Choose next "built-in" input for sampling:
                    $x_1, x_2, \ldots, x_{c(h-1)}, x_{c(h-1)+2}, \ldots, x_{ck}$
            (2.3.2) Set $u = C_k(\,(f_k(x_1), \ldots, f_k(x_{c(h-1)}), x, f_k(x_{c(h-1)+2}), \ldots, f_k(x_{ck}))\,)$
            (2.3.3) Set $v = \bigoplus_{i=1}^{h-1} \bigoplus_{j=1}^{c} (i\text{th bit of } x_{c(i-1)+j})$
                    $\bigoplus_{j=2}^{c}\ (h\text{th bit of } x_{c(h-1)+j})$
                    $\bigoplus_{i=h+1}^{k} \bigoplus_{j=1}^{c} (i\text{th bit of } x_{c(i-1)+j})$
            (2.3.4) Set $count\ =\ u \bigoplus v$
        (2.4) Set $avg = \frac{count}{t}$
        (2.5) If $avg \geq\ \frac{1}{2} + \frac{1}{2V(k)}$ then set $w = 1$
        (2.6) Elseif $avg \leq\ \frac{1}{2} - \frac{1}{2V(k)}$ then set $w = 0$
        (2.7) Set $y\ =\ w \circ y$ ($\circ$ denotes concatenation)
    (3) Output $y$
```

Let us analyze the "circuit size" of this algorithm. There are two loops, one of k iterations (step (2)) and one of t iterations (step (2.3)). Inside the inner loop, steps (2.3.1) and (2.3.3) each require $O(ck)$ circuit gates and step (2.3.2) requires size:

$$O(\,ckF(k) + size(C_k)\,)$$

where $F(k)$ is the assumed circuit size for computing f_k and $size(C_k)$ is the size of the circuit C_k which is computing the predicate B_k. Thus the entire algorithm can be converted to a circuit A_k which computes f_k^{-1} with probability at least $1 - \frac{1}{k^d}$ and has size:

$$O(\,kt[ckF(k) + size(C_k)]\,).$$

The constant c has been defined to be k^{2d-1} and from step (2.2) we see that $t = O(kV(k))$. Also, by assumption, we know that:

$$size(C_k) = O\left(\frac{U(k)}{k^2V(k)} - k^{2d}F(k)\right).$$

Thus, the circuit A_k has size:

$$O\left(k^2V(k)\left[k^{2d}F(k) + \frac{U(k)}{k^2V(k)} - k^{2d}F(k)\right]\right) = O(U(k)).$$

This completes the proof of the lemma. Since f_k is assumed to be a U-one-way function, we have actually shown:

Corollary 4.2.1 Let $C = \{C_k\}$ be any family of circuits where each C_k takes inputs from the domain D_k and has size:

$$O\left(\frac{U(k)}{k^2V(k)} - k^{2d}F(k)\right).$$

Then for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s^C_{i,k}(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ for every value of } i \text{ from 1 to } k \right] < 1 - \frac{1}{k^d}.$$

We need one more lemma to finish the proof of Theorem 4.

Lemma 4.2.2 Let $C = \{C_k\}$ be any family of circuits where each C_k takes inputs from the domain D_k and has size:

$$O\left(\frac{U(k)}{k^2Q(k)} - k^{2d}F(k)\right).$$

Then, for a randomly chosen $(x_1, x_2, \ldots, x_{ck}) \in D_k$ we have, for all sufficiently large k:

$$\text{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}.$$

Proof Suppose first that some family of circuits $C = \{C_k\}$ has the property that each C_k takes inputs from the domain D_k and has size:

$$O\left(\frac{U(k)}{k^2V(k)} - k^{2d}F(k)\right). \tag{1}$$

for some fixed polynomial $V(k)$. Then, from Corollary 4.2.1, for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s^C_{i,k}(x) < \frac{1}{2} + \frac{1}{V(k)} \text{ for } \textit{some} \text{ value of } i \text{ from 1 to } k \right] > \frac{1}{k^d}.$$

From Lemma 1.2.3 of Chapter 2, it then follows that there exists a constant a and polynomial Q such that $V(k) = aQ(k)$ and:

$$\mathrm{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}.$$

Replacing $V(k)$ by $aQ(k)$ in line (1) above gives:

$$O\left(\frac{U(k)}{k^2 aQ(k)} - k^{2d}F(k)\right).$$

which is just:

$$O\left(\frac{U(k)}{k^2 Q(k)} - k^{2d}F(k)\right).$$

proving the lemma.

Thus, the construction of Lemma 4.2 satisfies all four conditions of Lemma 4.1. For this construction we have that the quantity $size(B_k(g_k))$ of condition (3) is, as noted above, equal to k^{2d}. As just proven in Lemma 4.2.2, the polynomial $T(k)$ in condition (4) is equal to:

$$O\left(\frac{U(k)}{k^2 Q(k)} - k^{2d}F(k)\right).$$

Thus, the construction of Lemma 4.2 produces an RQ-weak generator where:

$$R(k) = O\left(\frac{U(k)}{k^2 Q(k)} - k^{2d}F(k) - P(k)k^{2d}\right).$$

This completes the proof of Theorem 4.

3.2 Fast Simulation of Probabilistic Computation

We now explore a very much stronger notion of a one-way function. Namely, we consider functions which can be computed in polynomial time but whose inverse is difficult to compute with high probability by any circuits of some fixed *sub-exponential* size. We show that if any such function exists, it is possible to build a bit generator which can be used to simulate polynomial-time probabilistic computation in deterministic polynomial time.

Definition Let I_k be the set of all k-bit strings, let $D_k \subseteq I_k$ and let $f_k : D_k \mapsto D_k$ be a sequence of permutations. We will write $D = \bigcup D_k$ and $f = \{f_k\}$. Then, f is a 2^{k^α}-weak one-way function (or simply a 2^{k^α}-one-way function) if the following properties are satisfied:

(1) The domain is accessible: there exists a probabilistic polynomial-time algorithm which, on input k, selects an $x \in D_k$ with uniform probability.

(2) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $f_k(x)$.

(3) There exists a polynomial Q and a fixed constant $\alpha < 1$ such that the following holds. If $C = \{C_k\}$ is a family of circuits where each C_k has k inputs and size $Poly(2^{k^\alpha})$ then for all sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{Q(k)} \text{ of the } x \in D_k.$$

We now define an analagous next-bit test which corresponds to this new function. As might be expected, these next-bit tests will have sub-exponential size. Similarly, the analagous bit generator will pass all such next-bit tests.

Definition Let $\alpha < 1$ be a fixed constant, P be a polynomial, S_k a multiset consisting of 2^{k^α}-long bit sequences and $S = \bigcup_k S_k$. A 2^{k^α}*-next-bit test* for S is a family of circuits $C = \{C_k^i\}$. Each circuit C_k^i has i Boolean inputs where $i < 2^{k^\alpha}$, one Boolean output and size $Poly(2^{k^\alpha})$. On input the first i bits of a sequence s randomly selected from 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 S *passes the test* C if for all sufficiently large k and all $i < 2^{k^\alpha}$:

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

We will refer to both C and C_k^i as 2^{k^α}-next-bit tests.

Definition Let $\alpha < 1$ be a fixed constant, I_k the set of all strings of length k, and $D_k \subseteq I_k$ a set of inputs (or "seeds") of length k. Let G be a deterministic algorithm which, on input a seed $x \in D_k$, outputs a 2^{k^α}-long bit sequence s_x in $Poly(2^{k^\alpha})$ time. Let $S_k = \{s_x | x \in D_k\}$. The algorithm G is a *Cryptographically* 2^{k^α}*-strong pseudo-random bit generator* (or simply a 2^{k^α} -generator) if the multiset $S = \bigcup_k S_k$ passes all 2^{k^α}-next-bit tests.

Definition The class $\mathcal{P}$ denotes those languages which can be accepted deterministically in time polynomial in the length of the input.

We now state the main theorem of this section:

Theorem 5 If any 2^{k^α}-one way function exists, then $\mathcal{R} \subseteq \mathcal{P}$.

Proof The proof will proceed as follows. First we show that if any 2^{k^α}-one way function exists, then it is possible to build a 2^{k^α}-generator. Then we show that the sequences output by this generator can be used to simulate any probabilistic machine in deterministic polynomial time. First we give a set of conditions which is sufficient to build 2^{k^α}-generators.

Lemma 5.1 Let $\alpha < 1$ be a fixed constant, let I_k be the set of k-bit strings and let $D_k \subseteq I_k$. Let $g_k{:}D_k \mapsto D_k$ be a sequence of permutations and let $B_k{:}D_k \mapsto \{0,1\}$ be a sequence of predicates. We will write $D = \bigcup_k D_k$, $g = \{g_k\}$ and $B = \{B_k\}$. If the following set of conditions hold, then it is possible to construct a 2^{k^α}-generator.

(1) The domain is accessible: there exists a probabilistic polynomial-time algorithm which, on input k, chooses $x \in D_k$ with uniform probability.

(2) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $g_k(x)$.

(3) There exists a polynomial-time algorithm which, on inputs k and $x \in D_k$, computes $B_k(g_k(x))$.

(4) Let $C = \{C_k\}$ be any family of circuits such that each C_k has k inputs and size $Poly(2^{k^a})$. Let Q be any polynomial. Then, for all sufficiently large k:

$$C_k(x) \neq B_k(x) \text{ for at least a fraction } \frac{1}{2} - \frac{1}{Q(k)} \text{ of the } x \in D_k.$$

Proof of Lemma 5.1 First we construct the 2^{k^a}-CSB generator and then prove that its outputs must pass all 2^{k^a}-next-bit tests. The proof is very similar to that of Lemma 4.1.

Choose an appropriate value of k to be the seed length and choose in probabilistic polynomial time a random $x \in D_k$ to be used as the seed. Set $c = 2^{k^a}$, the desired length of the output sequence, and generate the bits:

$$B_k(g_k(x)), B_k(g_k^2(x)), \ldots, B_k(g_k^c(x)).$$

The notation g_k^j indicates the j-fold composition of g_k. Now, output these bits in reverse order, i.e.:

$$B_k(g_k^c(x)), B_k(g_k^{c-1}(x)), \ldots, B_k(g_k(x)).$$

It should be clear that all of this can be accomplished in time $Poly(2^{k^a})$ by conditions (2) and (3).

It remains to show that the sequences output by this generator pass all 2^{k^a}-next-bit tests. Suppose that this is not true. Then there exists a polynomial Q and a family of $Poly(2^{k^a})$ size circuits $C = \{C_k^i\}$ where each C_k^i has $i < 2^{k^a}$ inputs and the following holds. For each of infinitely many values of k there exists some i such that:

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

where $p_{k,i}^C$ is the probability that the circuit C_k^i outputs the correct $i+1$st bit of a sequence when given the first i bits as input.

We now construct a family of circuits $A = \{A_k\}$ where each A_k has k inputs, size $Poly(2^{k^a})$ and such that for infinitely many values of k, A_k correctly computes the predicate $B_k(x)$ with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$. This contradicts condition (4).

Choose one of the infinitely many values of k such that for some $i < P(k)$:

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

The circuit A_k uses C_k^i as a "subroutine". On input $x \in D_k$, the circuit A_k first generates the i-bit sequence:

$$B_k(g_k^i(x)), B_k(g_k^{i-1}(x)), \ldots, B_k(g_k(x))$$

and inputs this sequence to the next-bit test circuit C_k^i. This can be accomplished with $O(\,i(Poly(k))\,)$ circuit gates since conditions (2) and (3) imply that $B_k(g_k(x))$ can be computed in polynomial size for all $x \in D_k$. Since $i < 2^{k^a}$, this whole process takes size:

$$O(\,2^{k^a}(Poly(k))\,) = Poly(2^{k^a})$$

The circuit A_k then outputs whatever value C_k^i outputs on these bits making the total size of A_k equal to:

$$Poly(2^{k^a} + 2^{k^a}) = Poly(2^{k^a}).$$

The proof that the circuit A_k solves the predicate B_k with probability at least $\frac{1}{2} + \frac{1}{Q(k)}$ is given in Lemma 4.1 and so we omit it here. This completes the proof of the lemma.

We now show that, given any 2^{k^a}-one-way function over an accessible domain $E = \bigcup_k E_k$, it is possible to construct a new domain $D = \bigcup_k D_k$, a function $g = \{g_k\}$ and a predicate $B = \{B_k\}$ which satisfy the four conditions of Lemma 5.1. Since the proof is virtually identical to that of Lemma 4.2, we give very few details.

Lemma 5.2 Let $f = \{f_k\}$ be a 2^{k^a}-weak one-way function over an accessible domain $E = \bigcup_k E_k$. Then, by definition, there exists some constant d such that given any family of circuits $C = \{C_k\}$ where C_k has k inputs and size $Poly(2^{k^a})$, we have, for all sufficiently large k:

$$C_k(x) \neq f_k^{-1}(x) \text{ for at least a fraction } \frac{1}{k^d} \text{ of the } x \in D_k.$$

The following construction satisfies conditions (1)-(4) of lemma 5.1:

(a) Set D_k to be the cartesian product of k^{2d} copies of E_k. It will be less cumbersome to write ck instead of k^{2d} so we will generally choose to do this, as in Chapter 2. Thus, D_k is the cartesian product of ck copies of E_k where $c = k^{2d-1}$. Formally:

$$D_k = \{\,(x_1, x_2, \ldots, x_{ck}) \mid x_1 \in E_k, \ldots, x_{ck} \in E_k\,\}.$$

(b) Let $g_k(\,(x_1, x_2, \ldots, x_{ck})\,) = (\,f_k(x_1), f_k(x_2), \ldots, f_k(x_{ck})\,)$ where each $x_j \in E_k$.

(c1) Let $B_k^i(x) =$ the ith bit of $f_k^{-1}(x)$ where $x \in E_k$.

(c2) Let

$$B_k(\,(x_1, x_2, \ldots, x_{ck})\,) = \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{c} B_k^i(x_{c(i-1)+j})$$

where each $x_j \in D_k$ and "$\bigoplus$" denotes the "exclusive-or" operation.

Proof of Lemma 5.2 It should be fairly obvious that conditions (1), (2) and (3) are satisfied by this construction (for more details, see the proof of Lemma 4.2). It remains to show that for any family of circuits $C = \{C_k\}$ where each C_k takes inputs from the domain D_k and has size $Poly(2^{k^\alpha})$, we get for all sufficiently large k:

$$\mathrm{Prob}[\,C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}$$

where $(x_1, x_2, \ldots, x_{ck})$ is a randomly chosen element of D_k. We do this through a sequence of two further lemmas.

Lemma 5.2.1 Let $C = \{C_k\}$ be a family of circuits where each circuit C_k takes inputs from the domain D_k and has size $Poly(2^{k^\alpha})$. Suppose that there exists a polynomial $V(k)$ and a constant d such that for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\mathrm{Prob}\left[\, s^C_{i,k}(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ for } \textit{every} \text{ value of } i \text{ from 1 to } k \right] \geq 1 - \frac{1}{k^d}.$$

Then, there exists a family of circuits $A = \{A_k\}$ where each A_k has k Boolean inputs, size $Poly(2^{k^\alpha})$, and for all sufficiently large k:

$$A_k(x) = f_k^{-1}(x) \text{ for at least a fraction } 1 - \frac{1}{k^d} \text{ of the } x \in E_k.$$

Proof The main details of the proof are identical to the proof of Lemma 1.2.2 in Chapter 2. The only remaining step necessary here is to show that, under the assumptions of this lemma, Algorithm A2 on page ? can be converted into a circuit A_k of size $Poly(2^{k^\alpha})$. As we saw in Lemma 4.2, where Algorithm A2 is reproduced, this algorithm can be converted into a circuit A_k of size:

$$O(\,Poly(k)(size(C_k))\,)$$

where $size(C_k)$ is the size of the circuit C_k. Since, by assumption, we have:

$$size(C_k) = Poly(2^{k^\alpha})$$

we know that A_k has total size:

$$O(\,Poly(k)Poly(2^{k^\alpha})\,) = Poly(2^{k^\alpha})$$

This finishes the proof of the lemma. Since f_k is assumed to be a 2^{k^α}-one-way function we have actually shown:

Corollary 5.2.1 Let $C = \{C_k\}$ be any family of circuits where each C_k takes inputs from the domain D_k and has size $Poly(2^{k^\alpha})$. Let $V(k)$ be any polynomial. Then for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\mathrm{Prob}\left[\, s^C_{i,k}(x) \geq \frac{1}{2} + \frac{1}{V(k)} \text{ for every value of } i \text{ from 1 to } k \right] < 1 - \frac{1}{k^d}.$$

We need one more lemma to complete the proof of Lemma 5.2.

Lemma 5.2.2 Let $C = \{C_k\}$ be any family of circuits where each C_k takes inputs from the domain D_k and has size $Poly(2^{k^\alpha})$. Let $Q(k)$ be any polynomial. Then, for a randomly chosen $(x_1, x_2, \ldots, x_{ck}) \in D_k$ we have, for all sufficiently large k:

$$\text{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}.$$

Proof Let $V(k)$ be any polynomial. Then, from Corollary 5.2.1, for all sufficiently large k we have, for a randomly chosen $x \in E_k$:

$$\text{Prob}\left[s_{i,k}^{C}(x) < \frac{1}{2} + \frac{1}{Q(k)} \text{ for } \textit{some} \text{ value of } i \text{ from 1 to } k \right] > \frac{1}{k^d}.$$

Then, from Lemma 1.2.3, there exists a polynomial $Q(k) = aR(k)$ for some constant a such that, for a random $(x_1, x_2, \ldots, x_{ck}) \in D_k$:

$$\text{Prob}[C_k((x_1, x_2, \ldots, x_{ck})) = B_k((x_1, x_2, \ldots, x_{ck}))] < \frac{1}{2} + \frac{1}{Q(k)}.$$

This completes the proof of both Lemma 5.2.2 and Lemma 5.2.

We now continue the proof of Theorem 5 by showing that the 2^{k^α}-generator which we have just constructed also passes all statistical tests of a type similar to those defined in Chapter 2.

Definition Let $\alpha < 1$ be a fixed constant, S_k be a multiset consisting of 2^{k^α}-long bit sequences and $S = \bigcup_k S_k$. A 2^{k^α}-*statistical test for strings* is a family of circuits $C = \{C_k\}$. Each circuit C_k has 2^{k^α} Boolean inputs, one Boolean output and size $Poly(2^{k^\alpha})$. The multiset S *passes the test* C if for every polynomial Q, and all sufficiently large k:

$$|p_k^S - p_k^R| < \frac{1}{Q(k)}$$

where p_k^S denotes the probability that C_k outputs 1 on a randomly selected $s \in S_k$ and p_k^R denotes the probability that C_k outputs 1 on a randomly selected $P(k)$-long bit sequence. We will refer to both C and C_k as statistical tests.

We can now show that 2^{k^α}-generators pass all 2^{k^α}-statistical tests.

Lemma 5.3 Let $\alpha < 1$ be a fixed constant, S_k a multiset consisting of 2^{k^α}-long bit sequences and $S = \bigcup_k S_k$. If the set S passes all 2^{k^α}-next-bit tests then S passes all 2^{k^α}-statistical tests.

Proof The proof is very similar to the proof of Theorem 2 so many details will be omitted. So suppose that a multiset $S = \bigcup_k S_k$ fails some 2^{k^α}-statistical test. Then, by definition, there is a family circuits $C = \{C_k\}$ such that C_k has 2^{k^α} inputs and size $Poly(2^{k^\alpha})$. Furthermore, there exists a polynomial Q such that for infinitely many values of k:

$$|p_k^S - p_k^R| \geq \frac{1}{Q(k)}$$

where p_k^S is the probability that C_k outputs 1 on a randomly selected sequence in S_k and p_k^R is the probability that C_k outputs 1 on a randomly selected 2^{k^a}-long bit sequence. We now show that S must necessarily fail a 2^{k^a}-next-bit test.

Let $[x|i]y$ denote the concatenation of the first i bits of x with y. For each $i < 2^{k^a}$, let p_k^i denote the probability that $C_k([x|i]y) = 1$ where x is chosen randomly from S_k and y is a random string of $2^{k^a} - i$ bits. Thus, p_k^i is the probability that the statistical test C_k outputs 1 when given as input the first i bits of some string in S_k followed by $2^{k^a} - i$ random bits. Note that $p_k^0 = p_k^R$ and $p_k^{2^{k^a}} = p_k^S$.

We now construct a family of circuits $A = \{A_k^i\}$ which constitute a next-bit test for S. Each A_k^i has $i < 2^{k^a}$ inputs and size $Poly(2^{k^a})$. Let k be chosen such that:

$$|p_k^S - p_k^R| \geq \frac{1}{Q(k)}.$$

Assume, without loss of generality, that

$$p_k^S - p_k^R \geq \frac{1}{Q(k)}.$$

Then, clearly, there must be a value of i such that

$$p_k^{i+1} - p_k^i \geq \left(\frac{1}{Q(k)}\right)\left(\frac{1}{2^{k^a}}\right).$$

(Remember, i varies between 0 and 2^{k^a}.) The circuit A_k^i will correctly predict this $i+1$st bit of any sequence in S_k with high probability.

Recall that p_k^{i+1} is the probability that C_k outputs 1 on any string consisting of an $i+1$-bit initial segment of a string in S_k followed by $2^{k^a} - (i+1)$ random bits. Let q_k^{i+1} be the probability that C_k outputs 1 when given the first i bits of a string in S_k followed by the *incorrect* $i+1$st bit, followed by $2^{k^a} - (i+1)$ random bits. Note that:

$$p_k^i = \frac{1}{2}p_k^{i+1} + \frac{1}{2}q_k^{i+1}$$

thus:

$$p_k^{i+1} - p_k^i = p_k^{i+1} - \left(\frac{1}{2}p_k^{i+1} + \frac{1}{2}q_k^{i+1}\right) \geq \left(\frac{1}{Q(k)}\right)\left(\frac{1}{2^{k^a}}\right)$$

$$\frac{1}{2}p_k^{i+1} - \frac{1}{2}q_k^{i+1} \geq \left(\frac{1}{Q(k)}\right)\left(\frac{1}{2^{k^a}}\right)$$

$$p_k^{i+1} - q_k^{i+1} \geq \frac{2}{Q(k)2^{k^a}} \qquad (1)$$

so:

$$p_k^{i+1} > q_k^{i+1}.$$

Consider in particular line (1). The fact that the difference between p_k^{i+1} and q_k^{i+1} is large means that if C_k is given as input the first i bits of some string $s \in S_k$ followed by $2^{k^\alpha} - i$ random bits, it can effectively distinguish between those strings containg the correct $i+1$st bit of s in their $i+1$st position, and those containing the incorrect $i+1$st bit.

Since $p_k^{i+1} + q_k^{i+1} = 1$ and $p_k^{i+1} > q_k^{i+1}$, if we can be sure of correctly predicting the $i+1$st bit of a randomly drawn string in S_k with probability p_k^{i+1}, then we will be correct more than half the time. The next-bit test A_k^i is constructed to do this.

So, suppose A_k^i is given as input $b_1, b_2, ..., b_i$, the first i bits of a randomly chosen sequence $s \in S_k$. Let b be the correct $i+1$st bit of s and let $\bar{b}$ be the incorrect $i+1$st bit. The circuit A_k^i first calculates the probabilities:

$$r_c = \text{Prob}[\,C_k(b_1, \ldots, b_i, b, b_{i+2}, \ldots, b_{2^{k^\alpha}}) = 1\,]$$

and

$$r_i = \text{Prob}\Big[\,C_k(b_1, \ldots, b_i, \bar{b}, b_{i+2}, \ldots, b_{2^{k^\alpha}}) = 1\,\Big]$$

where $b_{i+2}, \ldots, b_{2^{k^\alpha}}$ are chosen randomly. In fact, these probabilities can only be estimated with high accuracy but for the sake of clarity, let us assume for a moment that they can be calculated exactly. A discussion on estimating the probabilities appears later in the proof. Note that A_k^i does not actually know which is the correct and which the incorrect $i+1$st bit. It simply calculates the two probabilities–one of them will be r_c the other, r_i. The predicting circuit A_k^i now chooses $b_{i+1} = b$ with probability $\frac{r_c}{r_c + r_i}$ and chooses $b_{i+1} = \bar{b}$ with probability $\frac{r_i}{r_c + r_i}$.

Suppose we calculate the value of r_c for each i-bit prefix of a string in S_k and take the average of all these probabilities. This average is just the total probability with which the circuit A_k^i correctly predicts the $i+1$st bit of a randomly drawn string in S_k. This average is also equal to p_k^{i+1} so A_k^i predicts correctly with probability $p_k^{i+1} > \frac{1}{2}$. Thus there must exist some polynomial R such that A_k^i correctly chooses the $i+1$st bit for sequences in S_k with probability at least $\frac{1}{2} + \frac{1}{R(k)}$. The set S therefore fails the next-bit test $A = \{A_k^i\}$ and this part of the proof is finished.

Estimating the probabilities r_c and r_i is done using the same sampling procedure described in the proof of Theorem 2. However, here we have from line (1) above that:

$$p_k^{i+1} - q_k^{i+1} \geq \frac{2}{Q(k)2^{k^\alpha}}.$$

Thus, we would like our estimates to be off by less than $1/(2^{k^\alpha}Q(k))$ and, for reasons which will become clear below, we want to be confident that these estimates are off by this amount with probability:

$$1/2^{2^{k^\alpha}}.$$

To get the number of trials t we therefore solve the following equation where $c = 1/(2^{k^\alpha}Q(k))$ and $\epsilon = 1/2^{2^{k^\alpha}}$:

$$\begin{aligned} t &> \frac{\ln 2 - \ln \epsilon}{2c^2} \\ &> \frac{1}{2}\Big[\ln 2 - \ln(2^{-2^{k^\alpha}})(2^{2k^\alpha}(Q(k))^2)\Big]. \end{aligned}$$

This is certainly satisfied by:

$$t > 2^{2k^\alpha}(Q(k))^2 + 2^{3k^\alpha}(Q(k))^2.$$

We will now show that if the number of trials t satisfies this inequality then the next-bit circuit can actually be built in such a way that, given *any* i-bit prefix of a string in S_k as input, the circuit can, *with certainty*, estimate r_c and r_i with an error of less than $1/(2^{k^\alpha}Q(k))$. The argument is probabilistic. Suppose that $t = 2^{2k^\alpha}(Q(k))^2 + 2^{3k^\alpha}(Q(k))^2$ and let $W = \{w_1, \ldots, w_t\}$ be a set of random strings each of length $2^{k^\alpha} - (i+1)$. In other words each element w_i of W is a string of random bits:

$$w_i = b_{i+2}, b_{i+3}, \ldots, b_{2^{k^\alpha}}.$$

These strings will be used to do the sampling. Let x be a randomly chosen i-bit prefix of a string in S_k. We will say that the set W *fails on* x if, given x as input, at least one of the probabilities r_c and r_i cannot be estimated with an error of less than $1/(2^{k^\alpha}Q(k))$ when using all of the strings in W to do the sampling. Since the chance of failing on either r_c or r_i is less than $1/2^{2^{k^\alpha}}$ (because $\epsilon = 1/2^{2^{k^\alpha}}$), we get:

$$\text{Prob}[W \text{ fails on a random } i\text{-bit prefix of a string in } S_k] < \frac{2}{2^{2^{k^\alpha}}}.$$

Since there are at most 2^i i-bit prefixes:

$$\text{Prob}[W \text{ fails on } \textit{at least one } i\text{-bit prefix of a string in } S_k] < \frac{2}{2^{2^{k^\alpha}}}(2^i).$$

Since the next-bit circuit is trying to predict the $i+1$st bit and strings in S_k have length 2^{k^α}, we know that i can be at most $2^{k^\alpha} - 1$. Therefore,

$$\text{Prob}[W \text{ fails on at least one } i\text{-bit prefix of a string in } S_k] < 1.$$

So, there must exist at least one set W of t strings which does not fail on any inputs to the next-bit circuit. This set W will be "built into" the next-bit circuit, assuring that this circuit can always estimate the probabilities r_c and r_i with sufficient accuracy to be sure of predicting correctly more than half the time. Since t is polynomial in k, the next-bit circuit will have size polynomial in k as required. A complete algorithm for computing the next-bit is given on the next page.

Algorithm A3

input: an i-bit prefix of a string $x \in S_k$ given as $b_1, \ldots, b_i$
output: $y = i + 1$st bit of x (with probability $\frac{1}{2} + \frac{1}{R(k)}$ for some polynomial $R(k)$)

```
(1) Set $count_0 = 0$

(2) Set $count_1 = 0$

(3) Set $t = 2^{2k^a}(Q(k))^2 + 2^{3k^a}(Q(k))^2 n$
(4) For $sample$ = 1 to $t$ do

        (4.1) Choose next "built-in" input for sampling:
              $b_{i+2}, b_{i+3}, \ldots, b_{2^{k^a}}$

        (4.2) Set $count_0 = count_0 + C_k(b_1, \ldots, b_i, 0, b_{i+2}, \ldots, b_{2^{k^a}})$

        (4.3) Set $count_1 = count_1 + C_k(b_1, \ldots, b_i, 1, b_{i+2}, \ldots, b_{2^{k^a}})$

(5) Set $q_0 = \frac{count_0}{t}$

(6) Set $q_1 = \frac{count_1}{t}$

(7) Output $y = 0$ with probability $\frac{q_0}{q_0 + q_1}$
    Output $y = 1$ with probability $\frac{q_1}{q_0 + q_1}$
```

It should be clear that this algorithm can be built into a next-bit circuit of size:

$$O(t(size(C_k))$$

where $size(C_k)$ is the size of the circuit C_k. By assumption we have:

$$size(C_k) = Poly(2^{k^a})$$

and from step (3) we see that:

$$t = Poly(2^{k^a}).$$

Thus, we can build a next-bit circuit of size $Poly(2^{k^a})$ which succeeds with probability at least $\frac{1}{2} + \frac{1}{R(k)}$ for some polynomial $R(k)$. This completes the proof of Lemma 5.3. Since, by definition, any 2^{k^a}-generator passes all 2^{k^a}-next-bit tests, we have actually shown:

Corollary 5.3 Let G be a 2^{k^a}-generator whose output consists of the set $S = \bigcup_k S_k$ as defined above. Then S passes all 2^{k^a}-statistical tests.

We are now finally ready to show how 2^{k^a}-generators can be used to simulate probabilistic computation deterministically in polynomial time.

Lemma 5.4 Suppose that a 2^{k^a}-generator exists. Then, $\mathcal{R} \subseteq \mathcal{P}$.

Proof Let $L \in \mathcal{R}$ be accepted by a probabilistic machine M_R. Suppose that M_R runs in time n^j where n is the input length. We construct a deterministic machine M which accepts L in polynomial time. On input x, machine M simulates $M_R(x, y)$ using every possible ouput from the generator of length n^j as a possible witness y. The machine M accepts x if and only if some output w from the generator causes $M_R(x, w)$ to accept. By assumption, the generator "stretches" k-bit seeds into 2^{k^α}-bit pseudo-random sequences. Thus, in order to get all pseudo-random sequences of length n^j output by the generator, the deterministic machine M tries all inputs to the generator of length

$$(j \log n)^{1/\alpha}.$$

Also, the generator can output each sequence of length n^j in time $Poly(n^j)$ and the time required for each deterministic simulation of the probabilistic machine M_R is $Poly(n^j)$. The total time required for the entire simulation is therefore:

$$\begin{aligned} Poly(n^j)(2^{(j \log n)^{1/\alpha}}) &= Poly(n^j)(n^{(1/\alpha)(j)^{1/\alpha}}) \\ &= Poly(n). \end{aligned}$$

Thus, the machine M_R can be simulated deterministically by M in polynomial time.

Note that in our original definition of 2^{k^α}-generators (Lemma 5.2), we assumed only that each seed could be found in *probabilistic* polynomial time. Here, where we want to use the generator for a deterministic algorithm, we simply try every string of length $(j \log n)^{1/\alpha}$ as an input to the generator. Some of these might not be actual seeds for the generator, hence the resulting output might not be a pseudo-random sequence. However, this method will cover every possible seed which is all that is necessary here.

It remains to show that the machine M accepts the same language L which is accepted by M_R. As usual, let I_k be the set of inputs to the generator of length k and let $S_k = \{s_x | x \in I_k\}$. We will show that if M does not accept L, then the set $S = \bigcup_k S_k$ output by the generator fails a 2^{k^α}-statistical test.

Recall that if $x \in L$ then for a randomly chosen string y, $M_R(x, y)$ accepts with probability $\frac{1}{2}$. If $x \notin L$, then for no string y does $M_R(x, y)$ to accept. So if M does not accept the language L, there must be some $z \in L$ which M incorrectly rejects (i.e., none of the possible witnesses w output by the generator cause $M_R(z, w)$ to accept even though $z \in L$.) Furthermore, there must exist infinitely many such strings $z \in L$ or else M could be repaired without increasing its running time.

Let z be any element of L which M fails to accept. Recall that the probabilistic machine M_R runs in time n^j where n is input length. So, on input z, the deterministic machine M will simulate $M_R(z, y)$ on all strings y of length $|z|^j$ output by the CSB generator. Thus M will have to try all seeds of length $(j \log(|z|))^{1/\alpha}$ since the generator "stretches" k-bit inputs into 2^{k^α}-bit outputs. For the remainder of the proof, we will write k in place of $(j \log(|z|))^{1/\alpha}$.

For each of the infinitely many such strings z, there will be a circuit C_k with $|z|^j$ inputs which has z "hard-wired" in. On any input y, the circuit C_k simulates $M_R(z, y)$. This circuit outputs 1 if and only if $M_R(z, y)$ accepts. Since $M_R(z, y)$ runs in $Poly(|z|)$

time, the circuit C_k has size $Poly(|z|)$. Since:

$$k = (j \log(|z|))^{1/\alpha}$$

the circuit has size $Poly(2^{k^\alpha})$.

Since $y \in L$ we have, by the definition of $\mathcal{R}$, for a randomly chosen string y of length $|z|^j$:

$$p_k^R = \text{Prob}\{C_k(y) = 1\} = \frac{1}{2}.$$

However, since $M_R(z, y)$ does not accept for *any* $y \in S_k$, for a randomly chosen $y \in S_k$ we have:

$$p_k^S = \text{Prob}\{C_k(y) = 1\} = 0.$$

Thus $p_k^R - p_k^S = \frac{1}{2}$ so there is certainly a polynomial R such that $p_k^R - p_k^S > \frac{1}{R(k)}$. Therefore, the set S fails the 2^{k^α}-statistical test $\{C_k\}$ contradicting our assumption that S is the output of a 2^{k^α}-generator. This concludes the proof of both Lemma 5.4 and Theorem 5.

Appendix

In this appendix, we find a bound for t which satisfies the following inequality:

$$2(1-\Phi(z)) < \epsilon$$

where

$$z = 2c\sqrt{t}$$

and

$$\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du.$$

First, we solve the following for z:

$$2\left(1 - \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du\right) < \epsilon.$$

Rearranging and using the fact that $\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du = \sqrt{2\pi}$ gives:

$$\int_{z}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du < \frac{\epsilon}{2}.$$

There is, unfortunately, no closed form solution to the integral on the left side. We can, however, bound the left side from above by something which is easy to integrate. Assuming that z is greater than or equal to 1, the value of z obtained from solving the following inequality will certainly satisfy the original inequality. We will address the fact that z must be greater than or equal to 1 below.

$$\int_{z}^{\infty} u\, e^{-u^2/2}.du < \frac{\epsilon}{2}.$$

Integrating gives:

$$z^2 > 2\,(\ln 2 - \ln \epsilon).$$

Now, recall that $z = 2c\sqrt{t}$ which gives:

$$t > \frac{\ln 2 - \ln \epsilon}{2c^2}. \qquad (1)$$

In order to bound our original integral, we had to assume that z be greater than or equal to 1. Since $z = 2c\sqrt{t}$ we get:

$$2c\sqrt{t} \geq 1$$

which means:

$$t \geq \frac{1}{4c^2}. \qquad (2)$$

Now, since $0 \leq \epsilon \leq 1$ we can satisfy both inequalities (1) and (2) by choosing t so that:

$$t > \frac{\ln 2 - \ln \epsilon}{2c^2}.$$

References

[AL] D. Angluin and D. Lichtenstein, *Provable Security of Cryptosystems: a Survey,* Yale University, Dept. of Computer Science, TR-288,1983

[BBS] L. Blum, M. Blum and M. Shub, "A Simple Secure Pseudo-Random Number Generator,"*Advances in Cryptology,* ed. D. Chaum, R.L. Rivest, and A. T. Sherman, Plenum Press, 1983

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

[G] S. Goldwasser, *Probabilistic Encryption: Theory and Applications,* Ph.D. Dissertation, Univ. of California at Berkeley, 1984

[GMT] 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, pp. 134-144

[HPS] P. Hoel, S. Port and C. Stone, *Introduction to Probability Theory,* Houghton Mifflin, 1971, pp. 190-192

[PF] N. Pippenger and M. Fischer, "Relations Among Complexity Measures," *Journal of the Association for Computing Machinery,* Vol. 26, No. 2, April 1979, pp. 361-381

[Y] A. Yao, "Theory and Applications of Trapdoor Functions," *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science,* IEEE,1982, pp. 80-91