Abstract

Randomness plays a vital role in almost all areas of computer science, both in theory and in practice. However, the ability to provide high quality random bits to many users is expensive, either in terms of time or in terms of additional hardware. Therefore, programmers attempt to minimize the amount of random bits actually used through the utilization of pseudo-random generators (PRG's). A deterministic process is performed on a short random "seed" to produce a much longer output. It is hoped that this longer output will serve the same purpose as a truly random string of the same size. Theoretical work constructing good PRG's has fallen into two camps: cryptographically secure PRG's, which are based on the assumed intractiblity of some problem, and are usable for any polynomial-time purpose([BM],[Y]); and special purpose PRG's, which are provably good for some specified purposes, and are typically closer to those used in practice ([B],[KR],[BNS],[N],[CWi],[IZ]).

In this thesis, we develop some techniques which are useful in both the study of cryptographically secure PRG's and in analysizing the performance of simple, efficient PRG's when used in certain types of task. Our results are of three sorts. First, we construct several very simple and efficient PRG's (including one which is a slight variant of the linear congruential generator). We prove that the outputs of these generators are, for many purposes, nearly as good as truly random strings. Most of the results in this section appear in [IZ]. Second , we construct a cryptographically secure pseudo-random generator from any function which is one-way with respect to polynomial-size circuit families. This result was presented in [ILL]. Finally, we show that most tasks from modern cryptography are impossible unless one-way functions (and, hence, cryptographically secure pseudo-random generators) exist. These results appear in [IL2].

0.1 Acknowledgements

I have so many people to thank for so much that I will have to acknowledge people in categories. Some people belong to more than one, and may be mentioned in either or both.

I first want to thank the people who made direct technical contributions to the work described here.

Manuel Blum and Umesh Vazirani were infallible oracles pointing out the most promising and interesting lines of research. They taught me mathematics like a psychologist trains a rat through positive reinforcement: in running the maze of possible proofs, my motivation was Manuel's beatific smile of enlightenment and Umesh's excited, "Too good!" that awaited if I were successful. (Actually, to describe Manuel's smile as "beatific" is somewhat misleading. It actually looks more as if God had just told him the punchline to a slightly off-color joke.)

If I tried to write down all I owe Steven Rudich, both mathematically and personally, no one but he and I would believe it. Also, much of it would be embarrasing. Thus, I will limit myself to a few of my most plausible debts to Steven, and omit most of the good stuff. Steven and I spent countless hours together, as undergraduates and as graduates, exchanging puzzles, research ideas, and philosophies. Between us, we must have explored every plausible fallacy in complexity theory, many two or more times. Steven gave me my first taste of what it was like to do research as part of a community, and, upon our arrival at Berkeley, was responsible for making me part of the theory community there, introducing me and selling me to Manuel, Umesh, and many of the others that I haven't mentioned yet. His research blossomed immediately, whereas mine was more sluggish; I have to admit that my reason for research the first year or two at Berkeley was largely jealousy of Steven, a less saintly motivation than Manuel's smile. He helped me at many stages of the research described here; a specific example is that it was Steven's idea that the Leftover Hash Lemma might be used to construct a quasi-perfect PRG. Steven is an avid collector and inventor of interesting and counter-intuitive mathematical puzzles and problems, and he generously shares his inventory with anyone who will listen. I have often been among the beneficiaries of this generosity. Steven is both an inspired researcher, and himself an inspiration. If Steven were a Greek god, he would be Demonstrato, the transvestite muse of mathematics.

Another Berkeley student from and with whom I learned an enormous amount is Moni Naor. Moni is known for his encyclopedic knowledge and perspective, insight, and contagious enthusiasm. Working with Moni is incredibly educational, incredibly exhausting, and, if he is driving, incredibly dangerous. I benefitted from Moni's insight at many stages of this thesis; one small example is that originally, the Leftover Hash Lemma was proved for a specific hash function, and Moni pointed out that any universal hash function would suffice.

Ronitt Rubinfeld was also a major influence both on this thesis and my mathematical thinking in general. I would like to give her credit for a conversation in which she pointed out a direction which eventually led to the Leftover Hash Lemma, but she claims I misunderstood her. I am still not convinced that this is true, but I can still thank her for numerous prior and subsequent conversations where many of the questions about PRG's arose. I can also thank her for the many other problems that I've had the pleasure of working with her on, or of observing her clever solutions for.

Noam Nisan directly influenced a great deal of the first part of this thesis. Besides many helpful conversations, his own work provided many of the frameworks for measuring the quality of a PRG, and a standard of comparison, frequently unmatchable. Noam is the person of whose work I am most jealous.

Amos Fiat taught me much of what I know about cryptography. Many of the issues discussed in this thesis arose from conversations with Amos.

Charlie Rackoff is owed credit both for his wonderful proof of the Leftover Hash Lemma, and for a conversation leading to the third section.

I thank my co-authors Mike Luby, Leonid Levin, and David Zuckerman. Besides all of his technical contributions, Mike told me how to write mathematics, most of which advice I foolishly ignore. So don't blame Mike for the writing in this thesis, which is largely incomprehensible. I also want to thank him for his advice on a host of other matters; he is one of the few people in the field with more than their share of common sense. I thank Leonid for sharing his unique perspective on PRG's and complexity theory in general, and in particular, for an hour-long lunch at FOCS which required nine months of pre-processing to begin to understand, and another month to digest. I thank David for teaching me about expanders, and many other things.

In addition to people who made direct technical contributions, I would also like to thank all the other students at Berkeley who I learned with and from. A partial list includes Sampath Kannan, Valerie King, Mark Gross, Richard Cleve, Roman Smolensky, Lance Fortnow, David Wolfe, and Eddy Grove. I thank Mike Sipser and Richard Karp for some of the best taught courses it has been my pleasure to attend. I would also like to acknowledge some of the people who enriched my intellectual life as an undergraduate: David Feldman, Kevin Compton, Carol Wood, Tony Hager, Nathan Tawil, Josh Macy, and Julia Kay. I also want to thank Kevin for a letter he wrote me during my first year of graduate school which was a great encouragement.

I thank Robert Solovay, Ronitt Rubinfeld, Jeff Edmonds, Richard Cleve, Toni Pittasi, Steve Bellantoni, Dan Simon, and Valerie King for helpful comments on various versions of this thesis.

I thank Margaret for her emotional support and practical assistance.

I thank my parents for their unwavering confidence in my ability, for encouraging my intellectual interests without pressuring me towards any particular goal.

Contents

Chapter 1

Introduction

Randomness plays a vital role in almost all areas of computer science, both in theory and in practice. Randomized algorithms, algorithms with the ability to "flip coins" during their computations, have been used to speed up computation (e.g., [Rab],[SS]), parallelize sequential algorithms(e.g., [KW]), and to compute using less memory (e.g., [AKLLR]). Randomized algorithms are also frequently much simpler than their deterministic counterparts, and thus easier to implement. As well as random bits being used as tools for improving computation, there are also many situations when the desired output is not determined by the input, but is instead a sample from some probability distribution. For example, a cryptographic protocol might require a random prime or list of primes. A scientific simulation modelling a probabilistic or chaotic process (e.g. the weather) will also require randomness. For these tasks, there is no alternative to the use of random bits. Thus, randomness is an important computational resource.

To obtain random bits, a program might make a system call to determine the exact time to many digits of accuracy and use the least significant digits. To avoid this time-consuming process, and to obtain a higher quality of randomness, computers of the future might allow access to a physical source of randomness, say a Zener diode or Geiger counter. In either case, being able to provide high quality random bits to many users is expensive, either in terms of time or in terms of additional hardware.

Since random bits are expensive, programmers attempt to minimize the amount of random bits actually used through the utilization of pseudo-random generators (PRG's). A deterministic process is performed on a short random "seed" to produce a much longer output. It is hoped that this longer output will serve the same purpose as a truly random string of the same size. The most commonly used PRG is the linear-congruential generator.

Recent theoretical work has provided some ways of conserving random bits in algorithms. Blum and Micali ([BM]) introduced the notion of a cryptographically secure PRG; Yao ([Y1]) showed that such a generator produces output strings which are indistinguishable (in polynomial time) from truly random strings. Such pseudo-random generators could be used to reduce the amount of random bits used in any polynomial-time computation without any noticeable difference in the output. (Cryptographically secure pseudo-random generators also play a vital role in modern cryptography, particularly in private-key encryption.)

Nevertheless, there are problems with using cryptographically secure PRG's in randomized algorithms. First, such generators must be based on the intractibility of inverting some one-way function, an assumption at least as strong as the famous $P \neq NP$ conjecture. Second, even assuming that the function in question is one-way, the performance of such a PRG is only guaranteed asymptotically. In a particular example, one cannot compute exact bounds on the size of the seed needed. (This last objection could be eliminated if our knowledge concerning circuit lower bounds were better. However, at present, no non-linear circuit bounds for problems in NP are known; we would need a dramatic improvement before we could perform such computations.) Third, most such generators are too inefficient to be used in practice. Linear congruential generators, although known not to be cryptographically secure (see e.g. [FKL],[P]), continue to be used in practice.

Because of these disadvantages, work has been done to construct good generators for more specific tasks. For example, Santha [Sa] and Sipser [Si] introduced the notion of a "quasi-perfect pseudo-random generator." A quasi-perfect PRG can be used to decrease the probability of error of a probabalistic algorithm from a constant $< 1/2$ to an exponentially small amount, using only a constant factor more random bits. Until recently ([AKS], [CWi], [IZ]), no PRG was proven to be quasi-perfect, although Vazirani [V1] conjectured that a specific PRG, related to the shift register PRG to be presented here, is quasi-perfect.

Another recent direction of research has been to theoretically explain the practical success of simple PRG's like the linear congruential generator. Bach [B] and Karloff and Raghavan [KR] have shown that linear congruential generators or variants thereof have provably good performances when used in specific algorithms, such as taking square roots modulo a prime and Quicksort; the latter paper also gives a task for which the usual linear congruential generators are inadequate.

In this thesis, we develop some techniques (from probability theory and the theory of universal hash functions) which are useful in both the study of cryptographically secure PRG's and in analysizing the performance of simple, efficient PRG's when used in certain types of task. Our results are of three sorts, corresponding to the three chapters of this thesis:

1) In Chapter 2, (the results of which have been published in [IZ]), we construct several very simple and efficient PRG's (including one which is a slight variant of the linear congruential generator). We prove that the outputs of these generators are, for many purposes, nearly as good as truly random strings. The results in this section are information-theoretic in nature, and use no assump-

tions from complexity theory. We feel they provide a theoretical justification for the widely held belief that very simple, computationally predictable generators (such as linear congruential generators) work well in practice.

For example, our generators can be used to decrease the error probability of a probabalistic algorithm to an exponentially small amount, using only a small constant factor more random bits (i.e., they are quasi-perfect). They can also be used to sample from any distribution many times, using, in the limit, an optimally small number of random bits. More generally, we show how to sample from an arbitrary sequence of distributions, where the next distribution in the sequence can depend on the previous samples in an arbitrary way. (It is assumed that we are given an algortihm which samples from the desired distribution, and an upper bound on the entropy of the distribution. Our output may differ from a random sequence of samplings by an exponentially small amount.) We also show our PRG's can be used in any algorithm which uses less space (memory) than the number of random bits it uses, and has one-way access to its source of random bits (see [BNS],[N]).

The method we give for sampling from distributions can be used for any of the following applications: A) A cryptographer wants to generate a long list of keys of a certain type, say a long list of prime numbers of length n. Using our PRG's, she could do so using $n - logn + c$ bits per prime (amortized). This is the information-theoretic lower bound, since there are $O(2^n/n)$ primes of length n. B) A scientist wants to run a simulation many times; the outcomes of the simulations might cause her to alter the parameters for the next run. The simulation requires many random bits, but has a succinctly describeable outcome (say, the simulation is of global weather conditions, and the outcome is the temperature in several major cities on August 1.) Using the method described here, she would only need to use a number of random bits per simulation equal to the number of bits to describe the outcome per simulation. C) A probabalistic algorithm uses different sub-routines, called many times throughout the course of the algorithm. The outcome of these sub-routines are significantly shorter than the number of random bits they use. Then the method described here can reduce the amount of randomness the algorithm uses, without significantly affecting the running time taken by the algorithm. D) Last, imagine an operating system which is asked to provide random strings to different users' probabalistic algorithms, either to solve a problem or to sample from some distribution. If the users are required to provide an upper bound on the entropy of distributions they want to sample from, the system can satisfy all requests on-line while using the optimal number of random bits, amortized over requests. (A good upper bound is the length of the output of the algorithm, if the entropy is hard to calculate. The closer the estimate is to the actual entropy, the fewer random bits will be required.)

2) Chapter 3 applies the results and techniques of Chapter 2 to the study of cryptographically secure pseudo-random generators. More specifically, we construct a cryptographically secure pseudo-random generator from any function

which is one-way with respect to polynomial-size circuit families. (The generator thus obtained is also secure with respect to small circuits.) This result was presented in [ILL]. Hastad ([H]) has since extended this result, obtaining an analagous theorem for the model of cryptographic security in which adversaries are polynomial-time probabalistic machines, rather than circuits. (The final version of both [ILL] and [H] will appear as [HILL]. Although as stated here, both Hastad's hypothesis and conclusion are weaker than ours, his result is in fact a strict generalization of ours.)

Since cryptographically secure pseudo-random generators are themselves one-way functions, these results provide a natural necessary and sufficient condition for the existence of cryptographically secure pseudo-random generators. They culminate a long line of research. Cryptographically secure PRG's were first constructed based on specific number-theoretic problems ([BM],[BBS]), then on any one-way permutation ([Y1]), and then on any one-way function with a certain regularity condition on the number of pre-images ([GKL]) or which is also one-way on its iterates ([L]).

As well as using the first section, we develop a notion of "computational entropy", related to and based on Shannon's entropy function ([Sh]) which measures the randomness or uncertainty of a distribution.

3) The third section shows that most tasks from modern cryptography are impossible unless one-way functions (and, hence, cryptographically secure pseudo-random generators) exist. We show that any secure protocol for any of the following tasks can be used to construct a one-way function: identification/authentication, bit commitment, private key encryption, and coin flipping by telephone. This remains true even if we start with a weak, highly interactive protocol. We informally conjecture that, in fact, any task in cryptography which does not have an information-theoretic solution requires a one-way function. In other words, the existence of a one-way function is the weakest assumption with any consequences for cryptography. Although we only prove this for the above four tasks, we claim that our results can be easily extended to most other tasks in cryptography. The four chosen tasks represent a broad spectrum of modern cryptography; the proofs for these tasks give general methods for showing that cryptographic tasks require one-way functions. Also, frequently one can directly reduce one of these four tasks to the task in question, showing that a solution to the task could be used to accomplish one of the four described here, and, hence, can be used to cnstruct a one-way function. For example, any protocol for secret exchange can be used in a direct manner for identification; thus, from the theorem proved here, if secret exchange is possible, one-way functions exist. (This last result was also proved directly by Bellare, Cowan, and Goldwasser ([BCG]) using theorems from [IL1].)

Finally, the three Appendices present results related to the above, but which are not due to the author (although Appendices 1 and 3 do present new proofs of these results). The first gives new, direct proofs of versions of several classical theorems from probability theory, specifically Chernoff's bounds [Sp] and the

Martengale Tail Inequality [Sp]. The second appendix presents a simplified proof of Hastad's result in [H], which generalizes the main result of Chapter 3 , that a cryptographically secure PRG can be constructed from any one-way function, to uniform notions of security.

Chapter 2

Recycling Random Bits: Provably Good Pseudo-Random Generators for Amplifying Probabilities of Correctness and Sampling from Distributions

2.1 Overview

In this chapter, we show how to modify some simple pseudo-random generators which are used in practice, linear congruential and shift register generators, in order to make the outputs of these generators provably as good as truly random strings for many purposes. Both generators are of the following general form. The input to the generator consists of a random seed and a randomly chosen, simply describeable function. For example, in a linear congruential generator, the seed is a number x between 1 and a fixed prime p, and the function f is described by two numbers in the same range, a and b, where $f(x) = ax + b \ (mod\ p)$. The function is applied to the seed, obtaining a new seed $x' = f(x)$. The process is iterated, yielding the pseudo-random sequence $x, f(x), f(f(x)), \ldots$. Our modification consists of, at each iteration, replacing a relatively small number of bits of the seed with new, truly random bits. Thus, while our modification makes the generator slightly less efficient in terms of random bits used, it is just as efficient in terms of running time and memory.

Since the introduction of pseudo-random generators, researchers have been trying to find ways of comparing different generators as to the "randomness" of their outputs. For example, Knuth devotes Chapter 3 of *The Art of Programming* to determining how well generators perform on certain "statistical tests", such as counting the number of occurences of certain substrings in the outputs ([Kn],pp. 1-177). However, while failure of such specific statistical tests indicates that a pseudo-random generator is not producing random strings, passing such a test is only flimsy evidence that the strings produced have any "random" properties. For this reason, Blum and Micali ([BM]) introduced the notion of a *cryptographically secure* pseudo-random generator; such a generator was shown by Yao ([Y1]) to pass *any* statistical test which can be administered in polynomial-time. Thus, for all practical purposes, the output of such a generator is as good as a truly random string. They showed how to construct such a generator, assuming that a certain number-theoretic problem (discrete log) was computationally intractible.

However, there were certain problems with using cryptographically secure pseudo-random generators. First, there was the (mostly theoretical) problem that such a generator must be based on the existence of a *one-way function*, an assumption which seems much stronger than the famous $P \neq NP$ conjecture ([Co],[L2]). (In Sections 2 and Appendix 3, we will show the results from [ILL] and [H], proving that the existence of one-way functions is sufficient, as well as necessary, for the existence of cryptographically secure pseudo-random generators.) Second, such generators were only guaranteed to work asymptotically; for a specific purpose, it is difficult to calculate the required length of input to the PRG. Finally, and most importantly, the suggested cryptographically secure PRG's were too inefficient and complicated to be used in practice; actual programmers continued to use simple, efficient PRG's such as linear congruential generators. Although not cryptographically secure ([FKL],[P]), these generators seemed to work quite well in practice.

The goal of this chapter is to at least partially explain this seeming gap between theory and practice. Our results both help to theoretically justify the use of simple pseudo-random generators in practice, and suggest a minor alteration in the commonly used pseudo-random generators which would make them provably good for many purposes. For example, we show that our modified generators can be used to decrease the error of a BPP-type (Las Vegas) probabalistic algorithm from any constant less than 1/2 to an exponentially small amount. (The notion of such a *quasi-perfect* pseudo-random generator was introduced by Santha [Sa] and Sipser [Si]. Other constructions of such generators are based on random walks on expanders ([AKS],[CWi],[IZ], and see Appendix 2) and iterated hashing ([Ni]). There is also some earlier, related work which achieve increased probabilities of success using few random bits, but not an exponentially small error ([KPS],[CG]).) We also show that these generators can be used to sample repeatedly from any distribution using, in the limit, the information-theoretic smallest number of random bits. We show how to use these ideas to construct an operating system which feeds its users pseudo-random strings for use in various algorithms. If the users tell the system a bound on the entropy of the output of their algorithms (for example, the length of the output is always such a bound), the operating system will be able to satisfy all requests on-line, with only an exponentially small statistical error, using an optimal number of random bits, amortized. We also show that our methods can be used to save random bits in any algorithm which has subroutines whose outputs are significantly shorter than the number of random bits the subroutines use. In particular, our generators are almost as good as truly random strings for the purposes of use in any algorithm which uses less space than time and has one-way access to its source of random bits (c.f., [BNS],[Ni]). We can also iterate our method to get a more efficient PRG for the purposes of space-bounded computation.

2.2 Notation and Basic Definitions

Strings: For $x \in \{0,1\}^n, y \in \{0,1\}^m$, we will denote by x_i, $1 \leq i \leq n$, the i'th bit of x, by $x \uparrow i$, the i bit string $x_1 x_2 ... x_i$ and by $x \circ y$, the concatenation of x and y, i.e., the $n+m$ bit string $x_1 ... x_n y_1 ... y_m$. We let 0^n denote the n bit string every bit of which is 0, and 1^n denote the string of all 1's. We can also think of $\{0,1\}^n$ as an n dimensional vector space over the field with 2 elements, GF_2. In this context, for $x, y \in \{0,1\}^n$, we use $x \oplus y$ to denote the sum of the vectors x and y, or, equivalently, their bitwise exclusive or. We also use $x \cdot y$ to denote the inner product of x and y, i.e., $x \cdot y = 0$ if the number of places $1 \leq i \leq n$ where $x_i = y_i = 1$ is even and $x \cdot y = 1$ otherwise. Finally, if M is an $n \times k$ dimensioned matrix of 0's and $1's$, and $x \in \{0,1\}^n$, we denote by Mx the k bit string given by taking the product of M and x as matrices over GF_2. Note that if the rows of M are $m_1, ... m_k$, we have $(Mx)_j = m_j \cdot x$ for $1 \leq j \leq k$.

Distributions: In this thesis, we only need to consider distributions over finite sets. This allows us to avoid most of the complexities of measure theory and thus to take a naive approach to probability theory. If S is a finite set, a distribution on S is any map D, $D : S \to [0,1]$, so that $\sum_{s \in S} D(s) = 1$. For $X \subset S$, we denote by $D(X)$ the probability that an element chosen according to D is in X, i.e., $D(X) = \sum_{x \in X} D(x)$. We use $D|X$ to denote the conditional distribution on X determined by D, i.e., $D|X(x) = D(x)/D(X)$ for every $x \in X$. Of course, this is only defined when $D(X) \neq 0$. If S and T are finite sets, and f is a map from S to T, and D is a distribution on S, there is a corresponding distribution on T, $f(D)$ given by $f(D)(t) = \sum_{s | f(s) = t} D(s)$. Such a distribution is referred to as a *random variable* determined by D. Note that, although it is sometimes useful to think of T as being infinite, e.g., an interval of real numbers, the random variable really has only a finite number of possible values, so we can treat $f(D)$ as a distribution on the finite set $f(S)$. An *event* is a Boolean valued random variable, i.e., $T = \{0, 1\}$. If f is an event determined by D, a distribution on S, we use $Prob_D[f] = Prob[f(D)]$ to denote the probability that the event occurs, $f(D)(1)$. If f, D, S are as before, then *the conditional distribution determined by D given f, $D|f$,* is the conditional distribution determined by D on the set $X = \{s \in S | f(s) = 1\}$. If f is a random variable determined by D whose range is a subset of the real numbers, we use $Exp[f(D)]$ or $Exp_D(f)$ to denote the expected value of f on distribution D, $\sum_{t \in T} t f(D)(t) = \sum_{s \in S} f(s) D(s)$.

2.3 Universal Hash Functions

Here, we define the concept of a universal family of hash functions, and give various examples of such families. Later, we will show how to use any universal family of hash functions to construct a PRG. All PRG's constructed in this manner will be good for the purposes mentioned earlier. However, the specific family of hash functions used will determine the efficiency of computing the PRG, and the number of random bits the PRG will require. For example, we will discuss one family of hash functions based on linear congruences modulo a prime, and another based on convolutions with a random string. The first family of hash functions will yield a PRG which is a modified version of linear congruential generator; the second, a modified version of shift register generator. The hash family based on convolutions will have the advantages of using fewer random bits than the linear congruential hash, and being implementable quickly in parallel. However, this generator is only more efficient in terms of random bits if certain reasonable conjectures from number theory hold, and the linear congruential generator is more widely used in practice.

Definition 2.3.1 Universal Hash Functions (Carter and Wegman [CW]) *Let H be a family of functions mapping $\{0,1\}^n$ to $\{0,1\}^l$. We say H is* universal *or a* universal family of hash functions *if, for every pair of distinct n bit strings,x and y, and for h a hash function chosen uniformly at random from H, $Prob[h(x) = h(y)] = 1/2^l$. We say H is* almost universal *if, for x,y,h as before, $Prob[h(x) = h(y)] \leq 1/2^l + 1/2^n$. We say H is* hereditarily *(almost) universal if, for any $k \leq l$, the family of functions $\{h \uparrow k | h \in H\}$, where $h \uparrow k(x) = h(x) \uparrow k$ is (almost) universal.*

Example 2.3.2 Matrix Hash

Let $M_{(n,l)}$ be the family of functions indexed by all $n \times l$ matrices over the field with 2 elements, GF_2, where the value of the function indexed by a matrix M on an n bit input x is simply Mx. Here, nl bits are required to specify an element of this family, which will be far too many for our purposes in this chapter; however, it is the simplest construction. Matrix hash functions have been used in numerous applications , see e.g., [Si2].

Proposition 2.3.3 $M_{(n,l)}$ *is universal.*

Proof: Let x and y be any two distinct n bit strings. Then, for $M \in M_{(n,l)}$, $Mx = My$ if and only if $M(x \oplus y) = 0^l$. Since $x \oplus y \neq 0^n$, this event occurs with probability exactly 2^{-l} when M is selected at random, since the bits of $M(x \oplus y)$ are independent variables, each having probability 1/2 of being 0. (Let m_i be the i'th row of M. Each m_i is chosen independently . The i'th bit of $M(x \oplus y)$ is $m_i \cdot (x \oplus y)$.)

Example 2.3.4 Toeplitz Matrix Hash

The Toeplitz Matrix Hash is a variation of Matrix Hash which can be implemented using fewer random bits. A Toeplitz Matrix is one which is constant along diagonals. Thus, an $n \times l$ Toeplitz Matrix over GF_2 can be specified by $n + l - 1$ bits, n to specify the first row, and $l - 1$ to specify the first column (except the corner, which has already been specified). We can use an argument similar to the previous proposition to show that Toeplitz matrices are universal. Using Toeplitz Matrices as a hash function will yield a pseudo-random generator which is quite similar to shift register generators used in practice. The PRG obtained using Toeplitz matrices will also be implementable in parallel. Toeplitz matrices have been suggested for use as hash functions by, among others, Goldreich and Levin ([GL]).

Let $T_{(n,l)}$ denote the set of all $n \times l$ Toeplitz hash functions over GF_2. Again, the matrix T indexes the function mapping vector x to Tx.

Proposition 2.3.5 $T_{(n,l)}$ *is universal.*

Proof: Just as in the last proposition, the proof reduces to showing that, for any fixed n bit string $z \neq 0^n$, the probability that, for $T \in T_{(n,l)}$ chosen at random, $Tz = 0^l$, is exactly $1/2^l$. Let i be the least bit location so that $z_i = 1$. Let $t_1, ...t_n$ be the first row of T, and $t_1, t_0, t_{-1}, ..t_{-(l-1)}$ be the first column. The t_i are uniformly and independently chosen unbiased bits. Fix any values for $t_{i+1}, ..t_n$. The first bit of Tz will then be determined by t_i, since flipping t_i will flip the value of the inner product and $t_1, ...t_{i-1}$ do not affect this inner product. Similarly, once we have fixed $t_i, ..t_n$, the second bit of Tz is determined by t_{i-1}, and so on. Thus, the probability that the j'th bit of Tz is 0 given that the previous bits are 0 is exactly $1/2$, so the probability that all l bits of Tz are 0 is $1/2^l$. $\square$.

Example 2.3.6 Convolution Hash

The Convolution Hash is a version of matrix hash which, for lengths n with certain number theoretical properties, uses even fewer bits than the Toeplitz Matrix Hash. In fact, if certain reasonable conjectures from number theory hold, the convolution hash uses only $1 + o(1)$ times the minimum number n bits needed to specify a universal hash function on strings of length n. Like the Toeplitz Matrix hash, the Convolution Hash will yield a pseudo-random generator which is very similar to the shift-register generators used in practice, and which can be implemented in parallel.

Let $r = r_1 r_2 \ldots r_n, x = x_1 x_2 \ldots x_n \in \{0,1\}^n$. For $1 \leq l \leq n$, let $r^{(l)}$ denote the l'th shift of r, i.e. $r^{(l)} = r_{l+1} r_{l+2} \ldots r_n r_1 \ldots r_l$. The *convolution* of r on x, $Con(r,x)$, is the string $r \cdot x, r^{(1)} \cdot x, ..., r^{(n-1)} \cdot x$. Equivalently, let M_r be the $n \times n$ matrix whose $i'th$ row is $r^{(i-1)}$. (Notice that M_r is Toeplitz.) Then $Con(r,x) = M_r x$. Let $l \leq n$. Then let ${M_r}^l$ be the $n \times l$ matrix whose rows are the first l rows of M_r. Let n be such that $p = n+1$ is a prime and 2 is a generator modulo p, and let $l \leq n-1$. Then let $C_{n,l}$ be the following set of functions, indexed by the set of strings of length p. Let the value of the function described by a string r on an n bit input x be the first l bits of $Con(r, 1 \circ x)$, i.e., $f_r(x) = {M_r}^l (1 \circ x)$. We wish to prove that $C_{n,l}$ is universal.

We need the following lemma from [V1]:

Lemma 2.3.7 *Let p be as above; let z be any string of length p except 0^p or 1^p. Then the $p \times p-2$ matrix ${M_z}^{p-2}$ has full rank.*

We now use the lemma to prove that $C_{n,l}$ is universal.

Proposition 2.3.8 *$C_{n,l}$ is universal for $l \leq n-1$, if $n+1 = p$ is a prime and 2 is a generator mod p.*

Proof: Let x and y be distinct n bit vectors. Let $z = (1 \circ x) \oplus (1 \circ y)$. Then z is not 0^p , since $x \neq y$, and z is not 1^p , since $z_1 = 0$. Now, $r(1 \circ x) = r(1 \circ y)$ iff $r(z) = 0$. Let z^R be the reverse of z, i.e., $z^R = z_p z_{p-1} ... z_1$. A direct computation shows that $Con(r,z) = Con(z^R, r) = M_{z^R} r$. Thus, the same is true of the first l bits of $Con(r,z)$, so ${M_r}^l z = {M_{z^R}}^l r$. By the Lemma, the matrix ${M_{z^R}}^l$ has full rank, so the solution set to ${M_{z^R}}^l r = 0$ is a subspace of dimension $p - l$; thus, for a random r, the probability that ${M_{z^R}}^l r = 0$ is exactly $1/2^l$. $\Box$.

The length required to hash an n bit string using this method is the first prime larger than n with 2 a generator. If the extended Riemann Hypothesis holds, this will be $O(n)$ (see [V1]) and, for infinitely many n, will be $n+1$. Heuristic arguments, modelling the distribution of primes by a probabalistic process, suggest that there should always be such a prime between n and $n + O(\log^2 n)$; this is consistent with what is known to number theorists. Thus, it is plausible to conjecture that the least such prime larger than n grows as $n + o(n)$. If this is true, the Convolution hash is the most efficient possible in terms of the number of bits needed to specify a hash function. Since the prime involved is of magnitude $O(n)$, rather than length $O(n)$, every number between n and cn can be deterministically tested until such a prime is found. For practical purposes, the time taken by this process is irrelevant, since p will be a fixed parameter . (This assumes that the generator is implemented as a system operation, which always outputs a string of a fixed length, rather than having the length of output as part of its input.) The naive algorithm for computing $Con(r, x)$ takes $O(nm)$ time; however, using a Fast Fourier Transform, this can be reduced to $O(n \log n)$.

Example 2.3.9 Linear Congruential Hash mod p.

Let p be a prime of length $n+1$. Let $L_{n,l,p}$ be the set of pairs (a, b) of residues modulo p. We let (a, b) represent the function mapping input x to the l least significant bits of $ax+b$, where operations are performed modulo p. Now, $ax+b$ and $ay+b$ will be pairwise independent, i.e., for a fixed x and y, $ax+b$ and $ay+b$ are equally likely to be any pair of residues. However, their l least significant digits will not be pairwise independent, since the distribution on least significant bits of a random number modulo p is not uniform. However, no sequence of length l has probability exceeding $1/2^l + 1/p$; a simple calculation shows that this suffices for this family of hash functions to be almost universal.

Linear congruential hash functions as above require only $2n+2$ bits to represent. The disadvantage of this method is that it requires a prime of length $n+1$. Such a prime can be found quickly probabilistically, but it is not known how to do this determistically. In practice, this is not really a problem, since p is a fixed parameter. We could eliminate this technical obstacle by working in the field with 2^n elements instead, but calculations in an arbitrary field are less efficient than those modulo a prime. When converted into a PRG, this method of hashing will be the closest to the methods in general use.

2.4 The Leftover Hash Lemma

Probabalistic algorithms are proven correct assuming that the random bits used in the algorithm are in fact chosen independently and uniformly. However, in practice, it is unclear that it is possible to obtain such truly random strings. Devices such as time checks or even Zener diodes can actually yield outputs that are highly correlated, although perhaps in a complex manner. A natural model for these devices is as a *slightly random source*. A slightly random source is a black box which outputs strings from an unknown distribution, which may change over time. The exact nature of the distribution is not known to the user, but the user is guaranteed that the distribution must always have some minimal amount of unpredictability or "randomness" (entropy). In the last several years, much research has dealt with various notions of slightly random source, with different randomness conditions. A partial list of papers on this topic include: [B1], [SV], [VV], [V2], [CGHFRS], [CG], [LLS], [CWi], [McI] and

[Zu]. Although different results hold for different randomness conditions, it is in general impossible to obtain even 1 truly random (or even almost random) bit from such a source, by itself. (However, it is possible to obtain random bits from two independent such sources, and it is possible, if the source has enough randomness, to simualte BPP algorithms using one such source. ([SV], [VV],[Vaz2], [CG], [CWi], [Zu].))

In this section, we give a method for using truly random bits to obtain almost all the randomness from a slightly random source. The bits thus obtained will be (almost) uniformly distributed and (almost) uncorrelated with the truly random bits used. Thus, the truly random bits can be reused to obtain random bits from future outputs of the source. In this manner, a small number of truly random bits can be used to convert large numbers of slightly random strings to truly random strings with very little loss of "entropy". Although interesting from the point of view of slightly random sources, we will mainly be concerned with other applications of this lemma. This lemma is the key step in proving that the above families of hash functions yield quasi-perfect PRG's. It will also be used extensively in Chapter 3. (Zuckerman [Zu] applies the Leftover Hash Lemma, among other techniques, to the study of slightly random sources.)

First, we need some definitions from probability theory. We define two measures of the randomness of a distribution, the collision probability and Shannon's entropy function ([Shan]). We also give a measure, statical indistinguishability, of how close two distributions on the same set are.

Definition 2.4.1 *Let D and D' be distributions on a finite set S. Let the* collision probability *of D be the probability that two elements chosen independently according to D are the same, i.e., $\sum_{s\in S}(D(s))^2$. The* (Shannon) entropy *of D is given by:*

$$Ent[D] = \sum_{s\in S} -D(s)\log(D(s)) = Exp_D[-\log(D(s))]$$

. We say distributions D and D' are statistically indistinguishable within ϵ if, for every $X \subset S$, $|D(X) - D'(X)| < \epsilon$. (Equivalently, $\sum_{x\in S}|D(x) - D'(x)| < 2\epsilon$.) We say D is quasi-random within ϵ on S if D is statistically indistinguishable from the uniform distribution on S within ϵ.

Both the entropy and the collision probability of a distribution gives a measure of its amount of "randomness". The collision probability of the uniform distribution on any set of size S is $1/S$; the entropy for such a distribution is $\log S$. Thus, for uniform distributions on some set, the entropy of the distribution is the negative logarithm of the collision probability. The negative logarithm of the collision probability is the same as Renyi's 2-entropy ([Ren]). We will talk more about entropy in later sections; this section uses only the collision probability. In this section, we will occasionally use the term "entropy" in an informal manner, denoting any measure of randomness.

The following is a somewhat stronger and cleaner version of a lemma in [ILL]; the strengthened version and proof found here are due to Rackoff [Rac1]. A very similar lemma appears in [BBR], stated in terms of information theory. [McI] also proves some related results.

To see the lemma's application to the theory of slightly random sources, think of D as the distribution provided by such a source, and hash function h as being determined using $O(n)$ truly random coin flips. Say the source has l "bits worth of inherrent randomness", and we are willing to sacrifice e of these bits to make the error small. Then we can apply the hash function h to obtain $l-e$ bits which are uniformly distributed and uncorrelated with the bits used to pick h, within an error exponentially small in e.

Leftover Hash Lemma. Let D be a distribution on $0,1^n$ whose collision probability is at most 2^{-l}. Let $e>0$, and let H be an almost universal family of hash functions mapping n bits to $l-2e$ bits. Then the distribution $(h,h(x))$ is quasi-random within $1/2^e$ (on the set $H \times \{0,1\}^{l-2e}$), where h is chosen uniformly at random from H, and x is chosen according to D.

Proof.

We first show that the collision probability of the distribution $(h,h(x))$ is close to that of the uniform distribution on $H \times \{0,1\}^{l-2e}$. We then show that if a distribution on a set S has collision probability close to $1/|S|$, then that distribution is quasi-random on S .

Claim 1. The collision probability of the distribution $(h,h(x))$ is at most $(1+2/2^{2e})/(|H|2^{l-2e})$.

Proof. The collision probability is the probability that both $h_1 = h_2$ and $h_1(x_1) = h_2(x_2)$, when $h_1, h_2 \in H$ are hosen uniformly at random and $x_1, x_2 \in S$ are chosen according to D. This is $1/|H|$ times the probability that, for $x_1, x_2, h \in H$ randomly chosen, $h(x_1) = h(x_2)$. For any $x_1 \neq x_2$, the conditional probability of this event is at most $1/2^{l-2e} + 1/2^n$ by the definition of almost universal. By our assumption concerning the collision probability of D, the probability that $x_1 = x_2$ is at most $1/2^l$. Therefore, the collision probability is bounded by

$$1/|H| \cdot (1/2^{l-2e} + 1/2^n + 1/2^l) \leq (1+2/2^{2e})/(|H|2^{l-2e}).$$

Claim 2. Let D be a distribution on a finite set S. If the collision probability of D is at most $(1+2\delta^2)/|S|$, then D is quasi-random on S within δ.

Proof. By contradiction. Assume D were not. Then there exists an $X \subset S$ with $D(X) > |X|/|S| + \delta$. Without loss of generality, assume the above inequality is an equality; if not, we can use the same argment with $\delta' = D(X) - |X|/|S| > \delta$. Let d_1, d_2 be independently chosen according to D. The conditional probability that $d_1 = d_2$ given that both are in X is at least $1/|X|$; similarly, the conditional probability given that both are in $S - X$ is at least $1/(|S| - |X|)$. Thus, the collision probability is at least

$$\frac{D(X)^2}{|X|}+\frac{(1-D(X))^2}{|S|-|X|}=\frac{(|X|/|S|+\delta)^2}{|X|}+\frac{((|S|-|X|)/|X|-\delta)^2}{|S|-|X|}=\frac{1}{|S|}+\frac{\delta^2}{|X|}+\frac{\delta^2}{|S|-|X|}$$

. This last expression is minimized when $|X|=|S|/2$, so the collision probability is at least $(1+4\delta^2)/|S|$, contrary to assumption.

The Leftover Hash Lemma follows from combining Claims 1 and 2. □

2.5 PRG's Based on Hash Functions

In this section, we apply the results of the previous section to show that any family of almost universal hash functions can be used to construct a quasi-perfect pseudo-random number generator. In particular, this yields modified versions of linear congruential and shift register generators which are provably quasi-perfect.

Construction of Hash-Based PRG. Let r, k, k' be integers. Let H be a family of almost universal hash functions mapping r bits to $r-k'$ bits. The pseudo-random generator takes as a seed an r bit string x, $h \in H$, and a sequence of $k-1$ strings of length k', $s_1, s_2, \ldots, s_{k-1}$. It outputs k r bit strings $x_1, \ldots, x_k$ defined as follows:

- $x_1 = x$.
- $x_{i+1} = h(x_i) \circ s_i$.

In other words, to get the next string in the sequence, hash the previous string, and add new random bits to replace the bits lost in hashing. In the case of the linear congruential hash, it is the same as first applying a linear congruential generator to x, then replacing the k' most significant bits of the result with new random bits, repeating this operation k times.

Before stating the main theorem for this section, we prove some easy propositions about statistical indistinguishability which will be used in the proof.

Proposition 2.5.1 *Let D, D' be distributions on S, and E be a distribution on T. If D is indistinguishable from D' within ϵ, then so are the distributions (d,e), d chosen according to D, e according to E, and (d',e), d' chosen according to D',e according to E.*

Proof: Let $X \subset S \times T$. Assume without loss of generality that $Prob[(d,e) \in X] - Prob[(d',e) \in X] > \epsilon$. Then there must be some value e_0 so that the conditional probability, $Prob[(d,e_0) \in X]$ is more than ϵ greater than $Prob[(d',e_0) \in X$. Then, if we let $Y \subset S$ be given by $s \in Y \iff (s,e_0) \in X$, we have $Prob[d \in Y] - Prob[d' \in Y] > \epsilon$, contrary to the definition of statistically indistinguishable.

Proposition 2.5.2 *Let D, D', D'' be distributions on S. If D and D' are indistinguishable within ϵ and D' and D'' are indistinguishable within δ, then D and D'' are indistinguishable within $\epsilon + \delta$.*

Proof: Let $X \subset S$; then $|D(X) - D''(X)|/le|D(X) - D'(X)| + |D'(X) - D''(X)|/le\epsilon + /delta$.

Proposition 2.5.3 *Let D, D' be distributions on S and let f be a function from S to T. If D and D' are indistinguishable within ϵ, then so are the distributions $f(D)$ and $f(D')$.*

Proof: Let $Y \subset T$, and let $X = f^{-1}(Y)$. Then $|[f(D)(Y)] - [f(D')(Y])| = |D(X) - D'(X)| \leq \epsilon$.

Proposition 2.5.4 *Let D, D' be distributions on S and let $X \subset S$. If $D|X$ and $D'|X$ are indistinguishable within ϵ and $D|(S - X)$ and $D'|(S - X)$ are indistinguishable within ϵ, and if $D(X) = D'(X)$, then D and D' are also indistinguishable within ϵ.*

Proof: Let $Y \subset S$. Then $D|X(Y \cap X) - D'|X(Y \cap X) = D(Y \cap X)/D(X) - D'(Y \cap X)/D'(X) < \epsilon$. Since $D(X) = D'(X)$, we have $D(Y \cap X) - D'(Y \cap X) < \epsilon D(X)$. Similarly, $D(Y \cap (S - X)) - D'(Y \cap (S - X)), \epsilon(1 - D(X))$. Therfore, $D(Y) - D'(Y) < \epsilon$.

We can now prove that any hash-based pseudo-random generator can be used to amplify the correctness of probalistic algorithms, being nearly as good as truly random bits for this purpose.

Theorem 2.5.5 *Let A be a BPP algorithm with error $< 1/3$, using $r(n)$ random bits on an input of length n. Fix n. Let k be any positive odd integer, $k' = 2k + 2, r = r(n)$. Let A' be the algorithm which, given input $x, |x| = n$ and a rk bit "random tape" simulates A k times (by dividing the tape into blocks of size r and using each block for one simulation), and outputs the majority answer. Then A', when run using a pseudo-random sequence generated by the Hash-Based PRG with parameters r, k, and k' for its "random tape", has an error probability at most $2^{-\Omega(k)}$.*

Proof. The idea is that the only information about x_1 that we use is the single bit, "Does A accept using random tape x_1?". Given this information, the conditional distribution of x_1 still has almost r bits of entropy (or, more precisely, a collision probability of at most 2^{-r}.) The Leftover Hash Lemma tells us that the hash function "recycles" almost all of this entropy for future use.

Without loss of generality, assume the input in question is in the language. Thus, the set of random tapes which cause A to accept has size at least $(2/3)2^r$. Let X be an arbitrary subset of the accepting strings of the aforementioned size, and let $b(z) = 1$ if $z \in X$, and 0 otherwise. We claim that for $1 \leq i \leq k$, the distribution $b(x_1), ..., b(x_i), h, x_{i+1}$ is statistically indistinguishable, within $2i/2^k$, of the distribution consisting of i independent coin tosses of bias $2/3$, followed by a random $h \in H$ and a random r bit string x. We will prove this claim by induction on i.

To prove the claim for $i = 1$, note that both X and its complement have size at least $2^r/3$; thus, the uniform distributions on these sets have collision probabilities no more than $3/2^r$. Applying the Leftover Hash Lemma to these distributions, with $l = r - 2$ and $e = k$, we see that, for $j \in \{0, 1\}$, the distribution on $(h, h(x_1))$ given that $b(x_1) = j$ is quasi-random within $2/2^k$. Consider the distributions D given by $b(x_1), h, h(x_1)$ and D' given by b, h, R where b is a coin flipped with bias $2/3$, and R is a random $r - k'$ bit string. Let Z be the set of triples b, h, R with $b = 1$. Then, as we have seen, $D|Z$ and $D'|Z$ are indistinguishable within $2/2^k$, as are $D|(S - Z)$ and $D'|(S - Z)$. Also, $D(Z) = D'(Z) = 2/3$. Therefore, from Proposition 1.5.4, D and D' are also indistinguishable within $2/2^k$. The claim for i=1, that the distribution $b(x_1), h, x_2$ is indistinguishable within the aforementioned amount from the distribution b, h, R' where b is a coin flipped with bias $2/3$, and R' is a random r bit string, intuitively follows from this since x_2 is merely $h(x_1)$ padded with k' independently chosen random bits. Formally, we can prove this by applying Proposition 1 to D and D' (with E being the uniform distribution on strings of length k'), and then applying Proposition 3 to the resulting distributions, (with f being concatenation).

Assume the claim is true for $i-1$. Then the distribution $b(x_1), ..., b(x_{i-1}), h, x_i$ cannot be distinguished from the distribution $b_1, ...b_{i-1}, h, x$, where the b_j's are $i-1$ independent $2/3$ biased coin flips , and h and x are chosen uniformly at random, with probability exceeding $2(i-1)/2^k$. By Proposition 3, the same is true for the distribuitons $b(x_1)...b(x_{i-1}), b(x_i), h, h(x_i)$ and $b_1, ..b_{i-1}, b(x), h, h(x)$. From the claim for $i = 1$, together with Proposition 1, we have that this last distribution is indistinguishable within $2/2^k$ from the distribution $b_1, ...b_i, h, R$ where the b_j's are independent random coins of bias $2/3$, h is a random element of H and R is a random $n - k'$ bit string. Thus, by Proposition 2, the distributions $b(x_1), ...b(x_i), h, h(x_i)$ and $b_1, ..b_i, h, R$ are indistinguishable within $2i/2^k$. The claim then follows as in the case i=1, since x_{i+1} is merely $h(x_i)$ padded with k' random bits.

Using this claim when $i = k$, we see that the sequence of $b(x_i)$'s is statistically indistinguishable within $2k/2^k$ from k random independent coin flips of bias $2/3$. Therefore, the probability that the majority of the $b(x_i)$ are 1 is $1-2^{-\Omega(k)}$, since this last is true for independent coin flips of bias $2/3$. Since all elements of X cause A to accept, we are done.□

Remark. Using any of the Toeplitz Matrix, Convolution or Linear Congruential Hash, the above algorithm will use $O(r + k^2)$ random bits to obtain an error probability of 2^{-k}. In particular, for $k = r^{1/2}$, we obtain an error of $2^{-r^{1/2}}$ using $O(r)$ random bits. For the Linear Congruential Hash or Toeplitz Matrix

Hash, the exact amount of bits to simulate k independent runs of the algorithm is $3r + 2k^2 + 2k$; for the Convolution Generator, the amount would be $2r + o(r) + 2k^2 + 2k$, if reasonable number theoretic assumptions hold.

The Toeplitz Matrix and Convolution Generators can also be implemented in parallel. To compute x_{i+1} from x_i, the shift generator computes $r - k'$ bits of the convolution of r on x, and supplements it with k' new random bits. Both of these operations can be viewed as matrix multiplications on the vector $x_1 s_1 ... s_{k-1}$ by matrices which only depend on r and i. Thus, x_i can be computed by parallel machines in the time and using the number of processors to compute iterated matrix multiplication on i matrices of size $O(r) \times O(r)$. Thus, if probabalistic algoriithm A is in NC, A' can also be implemented in NC.

By "bootstrapping," we can reduce the probability of error to $2^{-r^{1-\gamma}}$ using only $O(r)$ random bits, or to 2^{-r} using $O(rlog^{O(1)}(r))$ bits. This is because our algorithm can be made more efficient if the probability of error is already small.

Theorem 2.5.6 *Let $0 < \epsilon < \delta < 1/4$. Let A be a probabalistic algorithm which, on inputs of length n, has an error probability of at most δ and uses r random bits. Let A' be the algorithm which simulates A k times, using the outcome of the Hash Based Generator with parameters r, k, k', where $k = 4\log\epsilon/\log\delta$ and $k' = 4\log\epsilon$. Then the probability of error on strings of length n for A' is at most 2ϵ.*

Proof: Fix a set of strings of size $1 - \delta$ where the algorithm is correct on the input. Since the size of this set and of its complement are both at least $\delta > \epsilon$, we can mimic the proof of the previous theorem to show that the sequence of bits, "Is x_1 in the set?... Is x_k in the set?", is statistically indistinguishable within ϵ of k random coin flips of bias $1 - \delta$. A simple computation shows that for our choice of k, this latter distribution has a majority of 1's with probability at least $1 - \epsilon$.

Corollary 2.5.7 *Let A be a probabalistic algorithm with error probability at most $1/3$ using $r(n)$ random bits on strings of length n. Let $\gamma > 0$. Then there is an algorithm A' which accepts the same language as A, uses $O(r(n))$ random bits, and has an error probability at most $2^{-r^{1-\gamma}}$.*

Proof: We prove this inductively for $\gamma = 1/2, 1/4.....$ The case $\gamma = 1/2$ is just Theorem 2.5.5. Assuming we have an algorithm A^* which has an error probability of $2^{-r^{1-2\gamma}}$, we apply the above theorem with $\delta = 2^{-r^{1-2\gamma}}$ and $\epsilon = 2^{-r^{1-\gamma}}$. Then the algorithm described in that theorem uses $O(r + kk') = O(r + \log^2(\epsilon)/\log(\delta)) = O(r + r^{2(1-\gamma)}/r^{1-\gamma}) = O(r)$ random bits.□

Corollary 2.5.8 *Let A be a probabalistic algorithm with error probability at most $1/3$ using $r(n)$ random bits. Then there is a constant c and an algorithm A' for the same language which uses $O(r(n)\log^c r(n))$ random bits and has an error probability at most $2^{O(-r(n))}$.*

Proof: The construction in the previous Corollary required i iterations of using Theorem 2.5.5 , to achieve an error probability of $2^{-r^{1-1/2^i}}$. Each iteration increases the number of bits used by a constant factor d (d depends on the method of hashing used.) Letting $i = \log\log r$, we have an error probability of $2^{-r^{1-1/\log r}} = 2^{-r/2}$, using $rd^i = r\log r^{\log d}$ random bits. □

2.6 Recycling Random Bits for Distributions

In this section, we generalize the technique of recycling random bits from "Yes" or "No" type algorithms to arbitrary distributions. This might be particularly useful for scientific simulations via Monte Carlo methods, where many bits are used, but only a few measurements of an experiment are recorded. The methods presented here could be used to recycle these bits for use in future experiments, or even for later stages of the same experiment. Other situations where these techniques might be used are: cryptography, where it might be necessary to generate many keys or problem instances from a certain distribution, such as a long list of primes congruent to 3 modulo 4; approximation algorithms such as in [JVV], where many samples from a distribution are used to determine a succinctly describable quantity; or any probabilistic algorithm which has a subroutine or subroutines which output fewer bits than the amount of randomness they use. We will also use these results in Section 2 (although the specific results proven here can be replaced with simpler constructions using the same techniques.) The techniques presented here also have several applications to specific problems not of any of the forms explicitly mentioned here. The methods presented here can also be used to generate a random prime, for example, using few random bits, as we will see in the next section.

We repeat the definition of the entropy of a distribution.

Definition 2.6.1 *Let D be a distribution on a finite set S. Then the entropy of D is given by*

$$Ent[D] = \sum_{s\in S} -D(s)\log(D(s)) = Exp_D[-\log(D(s))]$$

Intuitively, the entropy of a distribution measures the amount of randomness in the distribution. We talk of entropy as being measured in "bits". The uniform distribution on a set S has entropy $\log(|S|)$. Shannon [Shan] has shown the connection between the entropy of a distribution and the number of random bits required to generate an element according to that distribution.

Definition 2.6.2 *A* sampling method *for a set S is a function f from $\{0,1\}^\omega$ to S so that the value of f on any input depends on only finitely many bits of the input, i.e., for every $w \in \{0,1\}^\omega$, w has a finite prefix $x \in \{0,1\}^n$ for* *some integer n, so that for any string $w' \in \{0,1\}^\omega$ so that x is a prefix of w', $f(w') = f(w)$. We call the least such n the* number of bits used *by the sampling method on input w. The distribution produced by the sampling method is that determined by $f(w)$ for w selected according to the uniform distribution on $\{0,1\}^\omega$. Let D^k be the distribution on S^k determined by taking k independent samples according to D, i.e., $D^k(s_1, ...s_k) = D(s_1)D(s_2)...D(s_k)$.*

Theorem 2.6.3 (Shannon [Shan]) *Let D be any distribution on S , and let E be the entropy of D. Then, for any sampling method f for S^k producing D^k, the expected number of bits used by f is at least Ek. Furthermore, there is a sampling method f' which produces D^k using an expected number of bits at most $Ek + 1$.*

Shannon's theorem gives a lower bound for the number of random bits needed to sample from a distribution under any model of computation; however, the upper bound does not give an efficient algorithm. The following result can be thought of as giving a effective, efficient version of the upper bound in Shannon's Theorem, (Theorem 2.6.3) for a sufficiently large number of samples.

Theorem 2.6.4 *Let $D(z)$ be an indexed family of distributions so that $D(z)$ can be sampled from in polynomial-time using $r(|z|)$ random bits. Then there is a probabalistic polynomial-time algorithm which, given z, a number of samples T, and a number E with $E \geq Ent(D)$, outputs t samples from $D(z)$ using $ET + O(T^{5/6}r(|z|))$ random bits, so that the sequence output is statistically indistinguishable from a random such sequence within an error of $2^{-O(T^{1/3})}$.*

Note. If E is a close bound on the entropy of $D(z)$, and the entropy of $D(z)$ is always non-negligible (say, at least 1/poly(n)), then, for a sufficiently large polynomial $T = T(n)$, the amortized random bits per sample is the entropy $+o(1)$. Thus, the above theorem can be thought of as an effective version of the upper bound in Shannon's Theorem (Theorem 2.6.3).

We require some lemmas from probability theory. The following lemma follows from Chernoff Bounds ([]). (For an alternate proof, see Appendix 1.)

Lemma 2.6.5 *Let D be a distribution on the interval $[a, b]$, with expectation E. Let t be a positive integer, and let $0 < \alpha < t^{1/2}$. Pick $d_1, \ldots, d_t$ independently according to D. Then*

$$Prob[| \sum_{1 \leq i \leq t} d_i - tE| > \alpha t^{1/2}(b-a)] < 2e^{-\alpha^2/2}$$

.

Corollary 2.6.6 *Let f be a function on $\{0,1\}^r$. Let E be the entropy of the distribution $f(x)$, for $x \in \{0,1\}$ chosen randomly. Let f^t be the function mapping the tr bit string $x_1 \circ \ldots \circ x_t$ to $f(x_1) \circ \ldots \circ f(x_t)$. Then, for $x = x_1 \circ \ldots \circ x_t$ chosen* *at random, with probability at least* $1-2e^{-\alpha^2/2}$, $f^t(x)$ *has at least* $2^{(r-E)k-r\alpha t^{1/2}}$ *pre-images under* f^t.

Proof. Apply the previous lemma to the distribution on the random variable $D = log|f^{-1}(f(x))|$. D is a distibution on $[0,r]$ with expectation $r-E$, since the entropy of D, E, is the expectation, over a random x, of $-log(Prob[f(x') = f(x)]) = -log(|f^{-1}(f(x)|/2^r) = r - D$. The log of the number of preimages of x under f^t has the same distribution as the sum of t random elements of D. □

Proof of Theorem 2.6.4 For input parameter z, let $r = r(|z|), D = D(z), E \geq Ent(D)$, and let T be the number of samples desired. Let $R = rT^{2/3}, k = T^{1/3}$, and $k' = ET^{2/3} + 2rT^{1/2}$. The new random sampling algorithm is as follows. Use the Hash Based PRG, with parameters R, k, and k', to generate k pseudo-random strings of length R (using $O(R + kk') = O(ET + rT^{5/6})$ bits.) Break each string into $T^{2/3}$ blocks of size r; use each block to simulate the original sampling algorithm. This gives us a sequence of T outputs. We claim this sequence is statistically indistinguishable from T independent samples from D, within $2^{-O(T^{1/3})}$.

Let $x, h, s_1, ...s_{k-1}$ be the seed for the PRG, and let $x_1, ...x_k$ be the outputs. Let A be the original sampling algorithm, and let A' be the algorithm that breaks an input of size R into $T^{2/3}$ blocks of size r and simulates A on each block. Analagously to Theorem 2.5.5, we prove, by induction on i, that the distribution $A'(x_1)...A'(x_i), h, x_{i+1}$ is indistinguishable from the distribution $d_1, ..d_{iT^{2/3}}, h, x'$, ($iT^{2/3}$ random samples from D followed by a random $h \in H$ and a random R bit string x') within $4ie^{-T^{1/3}/2}$. We prove the case $i = 1$; the induction step is exactly as in Theorem 2.5.5.

Let $E' \leq E$ be the entropy of D. Note that, by Corollary 5.3, for x a random R bit string, $A'(x)$ has at least $P = 2^{(r-E')T^{2/3}-rT^{1/2}}$ pre-images with probability at least $1 - 2e^{-T^{1/3}/2}$. (Here, we use $t = T^{2/3}, \alpha = T^{1/6}$). Assume $y = A'(x)$ has at least P pre-images under A'; let X be the set of pre-images of y. $|X| \geq P \geq 2^{R-k'+rT^{1/2}}$ (from our choice of k' and since $E' \leq E$), so applying the Leftover Hash Lemma to X, we have that the distribution h, x_2 given that x_1 is in X is quasi-random within $2^{-rT^{1/2}/2}$. Since this holds for all such y, we conclude that the distribution $A'(x_1), h, x_2$ given that $A'(x_1)$ has at least P pre-images is statistically indistinguishable from $A'(x_1), h, x'$ given that $A'(x_1)$ has at least P preimages. Since this event occurs with all but an exponentially small probability, we are done. □

An even more general theorem along these lines can be proven. It can be interpreted as giving a way in which an operating system can provide pseudo-random bits to its users for use in any sort of probabalistic algorithm (decision, search, approximation, or random generation), using an amortized information-theoretic minimal number of random bits. The users are required to provide the system with an upper bound on the entropy of the distribution they are sampling from; if a better one can't be calculated, the number of bits of output desired can always be used as such a bound. Within an exponentially small error, the entire sequence of outputs will be statistically indistinguishable from the case when truly random bits are used, even if the algorithms depend on the outputs of previous ones. The pseudo-random generator used by the system is a small modification of the hash-based generators; the general procedure (hash the current seed, then replace the most significant bits with truly random ones) remains the same, but the number of bits to be replaced is now variable, depending on the bounds provided by the users.

We model the above situation as follows. Users request random bits for use in certain computational task; when doing so, they tell the operating system an entropy bound on the task. We assume that there is a fixed maximum number of bits r which suffices for any task the users might suggest. Thus, a task can, without loss of generality, be a function from $\{0,1\}^r$ to an arbitrary finite set, say, $\{0,1\}^l$ for concreteness. The users should not use the random (or pseudo-random) strings given by the operating system for one task for any other purpose. Thus, the state of all users at a particular time should be determined by the sequence of outputs of all the tasks requested up to that time, and, perhaps, random factors which are independent of the operating system. Thus, the task requested at time t is determined by the sequence of outputs $z_1, ...z_{t-1}$ of the previous tasks, and some random string R_{users} which codes all the random factors. For our purposes, we can usually think of R_{users} as having been determined at random according to some distribution at the beginning of the process. We make no assumptions about the complexity of determining the next task from the previous tasks and random factors; we allow the users to be arbitrarily powerful, as long as they comply with the aforementioned restrictions on information. The operating system must provide random bits for the users on-line, using only a random tape R_{Op} and the previously given bounds on entropies of tasks, $E_1, ..E_{t-1}$. Efficiency is important for the operating system. To be efficient in terms of time and memory, the operating system should use only a fixed polynomial in r memory, and answer requests in a fixed polynomial in r time. Since the operating system will be used indefinitely, we want both time and memory to be independent of the number of requests which have been made so far, t. We also want the system to be efficient in terms of the number of random bits it uses. By standard information theoretic arguments, it can be seen that any algorithm for such an operating system must use at least $\sum_{1 \le i \le t} E_i$ random bits. We want our operating system to asymptotically use this information-theoretic lower bound in random bits.

More formally,

Definition 2.6.7 *Let a* probabalistic task *on r bits be a function g from $\{0,1\}^r$ to $\{0,1\}^l$ for some l. A* probabalistic request *for r bits is a probabalistic task on r bits, together with a rational number called the* entropy bound*, $E \ge ENT[g(U_r)]$, where U_r is the uniform distribution on r bit strings. A* request function for T outputs *is a method of determining sequences of probabalistic* *requests, so that the request made at a given time depends only on the outputs of the previous tasks; formally, it is a function F which maps a sequence of at most T strings to a probabalistic request for r bits. A* request distribution *is a distribution on request functions.*

A bit provider *for tasks on r bits is a probabalistic algorithm A, which, given a sequence of entropy bounds $E_1, ...E_T$, outputs a sequence $x_1, ...x_t$ of pseudo-random strings on-line; i.e., $x_t = A(R_t, E_1, ...E_t)$ for $1 \leq t \leq T$, where R_t is the sequence of random bits flipped by A in the first t requests. A bit provider is* efficient *if each x_i can be produced in a fixed polynomial time in r after E_t is input, and A needs only remember a string of a fixed polynomial length in r, which can be updated in time a fixed polynomial in r. The* trivial bit provider *simply outputs r independently chosen random bits after each request. The* output sequence generated *by a request distribution D using bit provider A is the distribution on T tuples of outputs generated as follows. Pick a request function F according to D. For $t = 1, ..T$, define probabalistic request $G_t = (g_t, E_t)$, pseudo-random string x_t and output z_t recursively as follows: $G_t = F(< z_1, ..z_{t-1} >)$, $x_t = A(R_t, E_1, ...E_t)$ and $z_t = g_t(x_t)$.*

Theorem 2.6.8 *There is an efficient bit provider A, so that:*

- *A uses at most $\sum_{1 \leq i \leq T} E_i + o(T)$ random bits if $T > O(r^6)$.*
- *For any request distribution D on sequences of T tasks on r bits, the output sequence generated by D using bit provider A is statistically indistinguishable from that using the trivial bit provider within $T2^{-r^c}$.*

Proof: The proof is similar to that of the last theorem. First, we describe the bit provider. It works along the same lines as the hash-based pseudo-random generator, but with the number of bits replaced after each hash of the seed depending on the E_i's. Let $t = r^5, R = tr$. Let H be a family of hereditarily (almost) universal hash functions from R bits to $R - 1$ bits. (Since H is hereditarily universal, we can use an element of H to hash to a variable number of bits.) The algorithm is as follows. Pick a random $h \in H$, and a random R bit string z_0. Divide z_0 into t blocks of size r, and use these as the first t outputs. Let $k_1 = \sum_{1 \leq i \leq t} E_i + 2r^4$. Use h to hash z_0 down to $R - k_1$ bits; add k_1 new random bits to obtain an R bit string z_1. Repeat, using z_1 in place of z_0, and using the next batch of E_i's, i.e., $t + 1 \leq i \leq 2t$. The algorithm uses $O(R) + 2Tr^4/t + \sum_{1 \leq i \leq T} E_i$ bits, which is as prescribed, as long as $T > O(r^6)$, and is implementable on-line, since all that need be maintained in memory is h and the current z_j, a total of $O(R) = O(r^6)$ bits.

It remains to show that, on any request distribution, using the outputs generated by this algorithm in place of truly random inputs only causes a negligible statistical difference in the sequence of outputs. It suffices to show this for any fixed request function, since then, whatever request function is chosen, the conditional statistical distinguishablity is small. The proof follows almost word for word that of Theorem 2.6.4 , once we know that, with high probability, $R-\sum_{1\leq i\leq t} E_i$ is a good lower bound for the size of the pre-image of the sequence $g_1(y_1),..g_t(y_t)$, i.e., the number of tuples $y_1',...y_t'$ with $g_i(y_i')=g_i(y_i), 1\leq i\leq t$. For this, we need the Martingale Tail Inequality [Sp](see Appendix 1 for a proof.)

Martingale Tail Inequality: Let D be a distribution on t tuples $x_1,...x_t$ of numbers in the interval $[a,b]$, so that the expectation of x_i is independent of $x_1,..x_{i-1}$. In other words, letting $E_i = Exp(x_i)$, for any $1\leq i\leq t$ and any sequence $x_1,..x_{i-1}$ with a non-zero probability of being output as the initial sequence of an element of D, we have $Exp_D[x_i|x_1,..x_{i-1}] = E_i$. Then $Prob_D[|\sum_{1\leq i\leq t} x_i - \sum_{1\leq i\leq t} E_i| > (b-a)kt^{1/2}] < e^{-k^2/2}$.

Corollary 2.6.9 *Let D be any distribution on t tuples of numbers in $[a,b]$. Let the random variable E_i be defined by $E_i = Exp_D[x_i|x_1,...x_{i-1}]$. Then $Prob_D[|\sum_{1\leq i\leq t} x_i - \sum_{1\leq i\leq t} E_i| > 2(b-a)kt^{1/2}] < e^{-k^2/2}$.*

Proof: Let the random variable y_i be defined as $y_i = x_i - E_i$. Then, whatever values $x_1,...x_{i-1}$ are chosen, $Exp_D[y_i|x_1,..x_{i-1}] = 0$, since E_i is the expectation of x_i given $x_1,..x_{i-1}$. Since $y_1,..y_{i-1}$ are completely determined by $x_1,..x_{i-1}$, the same is true for $Exp_D[y_i|y_1,...y_{i-1}]$. Thus, the conditions for the Martingale Tail Inequality hold for the distribution $y_1,..y_t$, with the values for the expectations in the theorem all 0. Since both x_i and E_i are in the interval $[a,b]$, $-(b-a)\leq y_i\leq (b-a)$. Applying the Martingale Tail Inequality to y_i gives the stated result. $\Box$

Corollary 2.6.10 *Let F be a request function on t tuples of requests on r bits. Pick t r bit strings $s_1,..s_t$ independently at random, thus generating t probabilistic requests $G_i=(g_i,E_i)$. Then with probability at least $1-e^{-k^2/4}$ there are at least $2^{rt-\sum_{1\leq i\leq t} E_i - krt^{1/2}}$ sequences $s_1',..s_t'$ with $g_i(s_i')=g_i(s_i)$ for all $1\leq i\leq t$.*

Proof: Let x_i be the random variable defined as $\log(|{g_i}^{-1}(s_i)|)$. Given $s_1,..s_{i-1}$, the expectation of x_i is $r-Ent[g_i(U_r)]\geq r-E_i$. Also, $0\leq x_i\leq r$. Thus, applying the previous corollary to the distribution $x_1,..x_t$, we have that $\sum_{1\leq i\leq t} x_i \geq rt-\sum_{1\leq i\leq t} E_i - 2rkt^{1/2}$ with probability at least $1-e^{-k^2/2}$. Since the number of pre-images is $2^{\sum_{1\leq i\leq t} x_i}$, the result follows immediately. $\Box$.

Proof of Theorem 2.6.8: As in the proof of Theorem 2.6.4, we show by induction that the distribution on pairs (h,z_k) given $g_1(y_1),...g(y_{kt}$ is, with probability exponentially close to 1, quasi-random within an exponentially small amount. The induction is just as before; we prove the base case, k=1. From the Leftover Hash Lemma, we know that the distribution in question will be quasi-random provided that the number of pre-images of the sequence $g_1(y_1),...g_t(y_t)$ is much larger than 2^{R-k_1}, since $R-k_1$ is the number of bits we hash the string down to, and from the point of view of the algorithm, the string z_0 being hashed is a random such pre-image. By definition, $k_1 = \sum_{1\leq i\leq t} E_i + 2r^4 =$

$\sum_{1\leq i\leq t} E_i + 2r^{1/2}t^{1/2}$. Applying the above Corollary with $k = r^{1/2}$, we have that, with probability at least $1 - e^{-r/4}$, the number of pre-images is at least $2^{R-k_1+r^4}$, in which case the distribution on h, z_1 will be quasi-random within $2^{-r^4/2}$. $\square$.

2.7 Generating Random Primes of Length n

The results of the previous section achieve, in the limit, an optimal method in terms of random bits of sampling from any poly-time generable distribution. However, the number of samples required before the method achieves the results is a rather large polynomial. In this section, we indicate how, for many interesting problems, the method of "recycling random bits" can be helpful when only a small number of samples are required. Instead of doing this in complete generality, we will concentrate on the natural and "practically" important problems of generating a random n bit prime, or several independent random primes of this length. It should be clear that the methods described here are using very little information about the nature of the problem, and so can be generalized greatly. Since no new techniques are introduced in this section, we will present the results as an informal discussion, rather than proving them in detail.

The naive method for generating a random prime is simply to pick a random n bit number and test for primality, repeating until the desired number of primes are discovered. Since there are $O(2^n/n)$ primes of length n, each trial has a $O(1/n)$ chance of succeeding. Thus, we expect to use $O(n^2)$ random bits per prime sampled. We will keep the general form of the algorithm, but change the random tape to a pseudo-random one which is almost as good for this purpose.

One complication with the above algorithm is that, unless the Extended Reimann Hypothesis holds ([Mil]), no deterministic polynomial-time primality tester is known. There are, however, probabalistic polynomial-time algorithms which use n bits to acheive an error bounded away from $1/2$ ([Rab],[SS]). We can use Theorem 2.5.5 to obtain a probabalistic algorithm using $O(n)$ bits and with an error probability of $2^{-n^{1/2}}$. Once we have this exponentially small chance of error, we can use the same random tape for every primality test; if we perform T tests, the probability of ever making an error will be $T/2^{n^{1/2}}$. Thus, at the expense of only an exponentially small error term, and of $O(n)$ additional random bits, we can assume we have an "oracle" which decides primality.

When we make one run of the naive algorithm, we are interested in the following information: whether the number we chose is prime, and, if so, its value. We can thus think of the algorithm as having output a random prime with probability $1/cn$, and "Not a prime" with probabiltiy $1 - 1/cn$, for c the constant given by the Prime Number Theorem for the density of primes [HW]. For any fixed prime, there is a unique random tape which causes that prime to be output. Thus, the random tape has no conditional entropy given the output; we have "gotten our money's worth" from this tape. On the other hand, "Not a prime" has $2^n(1-1/cn)$ random tapes which cause it to be output; in this case, we are entitled to recycle almost n bits . Our method of sampling will thus go as follows. Let ϵ be the allowable statistical error between our output and a random prime, and let $k = -2\log\epsilon + 3$. Pick a random hash function h mapping n bits to $n-k$ bits. Pick a random n bit string x_0. Test x_0 for primality; if prime, output x_0 as the first element on your list of primes and pick a new random n bit string x_1. Otherwise, let x_1 be $h(x_0)$ padded with k new random bits. Repeat, using the same hash function, and with x_1 in place of x_0. Using the Leftover Hash Lemma, and an induction argument as in the other results in this section, it is easy to show that (h, x_t) is quasi-random within $t\epsilon$. Thus, as long as $t\epsilon$ is $o(1/n)$, each x_t has a $O(1/n)$ chance of being prime, and the conditional distribution given x_t is prime is within $cnt\epsilon$ of the uniform distribution on n bit primes. Clearly, given h, the primes generated later are completely independent from previous primes, since after we find a prime, we pick a new seed at random. Thus, the sequence will be indistinguishable from a random sequence of primes of the same length, within $cnt^2\epsilon$. Since each x_i has a $O(1/n)$ chance of being prime, we expect to have $O(n)$ runs per prime, and since the $O(1/n)$ probability holds given that we have not yet found a prime, the probability of having more than kn tests without finding a prime is exponentially small in k. Thus, we expect to use $O(n+nk) = O(n\log 1/\epsilon)$ random bits per prime generated. To be slightly more efficient, we could let the x_i's be ln bit strings , which we divide into l n bit numbers , testing each for primality. If we found p primes among this set, we could hash x_i to $(l-p)n - k - cl/n$ bits, and pad with new random bits to obtain x_{i+1}. This method uses n bits per prime, plus $O(nl)$ bits to pick a hash function and initial seed x_0, plus an additional $k + cl/n$ bits per iteration. We would expect cn/l iterations per prime, so the expected number of bits used per prime is $n + 1 + cnk/l + O(nl/T)$, where T is the number of primes desired. The optimal value for l is $O((kT)^{1/2}) = O((T\log 1/\epsilon)^{1/2})$; when $T > -\log\epsilon$, this yields $n + O(1)$ bits per prime. In other words, it is possible to generate T primes using $n + O(1)$ (expected) bits per prime to within a $2^{-O(T)}$ statistical error.

2.8 Saving Random Bits in Space-Bounded Computation

In this section, we show that the hash-based pseudo-random generators work well in any algorithm which makes calls to probabalistic sub-routines whose outputs are significantly shorter than the number of random bits they use. In particular, they provide a provably good pseudo-random generator when used in any probabalistic space-bounded algorithm which has only one-time access to the random source (see [BNS]). If the hash-based generator is modified slightly by picking a fresh seed every so many steps, and the process is iterated, we obtain a generator which passes any space S, time T statistical test within ϵ , and which uses $O((S + \log T + \log \epsilon)(\log T)^2)$ random bits. The iterated construction resembles strongly Nisan's pseudo-random generator for space-bounded computation ([N]), which is also based on hash functions. Indeed, the two were mutually inspired; Nisan was attempting to apply the method of recycling randomness to space-bounded computation when he made his discovery, and the iterated construction given here is largely an immitation of Nisan's generator. Nisan's construction is both simpler and more efficient, using only $O((S + \log T + \log \epsilon) \log T))$ random bits. However, we feel that it is interesting to note that the recycling bits method can yield similar results. We should note that Nisan's generator can also be used for many of the same purposes as the hash-based generators given here, such as amplification of correctness probabilities. Here, the two methods yield incomparable results. See [N] for an exact statement of what is achievable using Nisan's technique. We do not think that a statement equivalent to Theorem 2.6.4 can be proven using Nisan's method.

Informally, an algorithm with space-bounded probabalistic sub-routines is one which uses its random bits in sub-procedures which have relatively small outputs, compared to the number of random bits they use. After the output of the sub-procedure is obtained, the random bits used are forgotten, i.e., not used later in the algorithm. The sub-procedures may depend arbitrarily on the previous computation, including all outputs of previously called sub-procedures. Some examples might be programs where at various stages, complex probabalistic simulations are used to estimate a simply describeable quantity, such as in scientific simulations, or algorithms which use less space than time. (This last will be discussed in more detail later.)

For simplicity, we assume that each sub-procedure uses R random bits, and outputs a string of length s. If the algorithm makes many calls to sub-procedures, compared to the number of bits used in each, then Theorem 2.6.8 shows that the program can be simulated using $s + o(1)$ random bits per call to the sub-procedures. The method used here can simulate algorithms which make relatively few calls to the sub-procedure, at the expense of less efficiency in terms of random bits used, and a greater statistical error.

Definition 2.8.1 *Let r and s be integers, $r > s$. A* sub-procedure on r bits with output bound *s is simply a function g from $\{0,1\}^r$ to $\{0,1\}^s$. An* algorithm which makes T calls to sub-procedures on r bits with space bound *s is a function mapping a "random tape" R, and a sequence of less than T s bit strings ("previous outputs"), $y_1, ...y_t$, $t < T$, to a sub-procedure on r bits with output bound s. The* distribution determined *by such an algorithm F is the distribution on sequences of length T of s bit strings defined by picking a R at random, picking T r bit strings, $x_1, ...x_T$, at random, and defining the sequence $y_1, ...y_T$ inductively by $g_i = F(R, y_1, ...y_{i-1})$ and $y_i = g_i(x_i)$. The* distribution determined by running the algorithm using a pseudo-random generator *P is obtained similarly, except that the x_i's are chosen using P rather than uniformly* *at random.*

Theorem 2.8.2 *Let A be any algorithm making T calls to sub-procedures on r bits with output bound s. Let $k' = 3\log T + s - 3\log\epsilon$. Then the distribution determined by A using truly random bits is statistically indistinguishable from that using the hash-based PRG with parameters r, T, and k' to within 4ϵ.*

Proof: Fix the random tape R. We prove by induction that the distributions $y_1, ..y_i, x_{i+1}, h$ and $y'_1, ...y'_i, x'_{i+1}, h'$ are indistinguishable within $4i\epsilon/T$, where the first distribution is obtained using the hash-based PRG (using hash function h) and the second is obtained using truly random x'_j's and picking h' at random. The theorem is then the special case $i = T$.

For $i = 1$, the first distribution is given by $g_1(x_1), h(x_1) \cdot s_1, h$. For a fixed value y_1 for $g_1(x_1)$, if $|{g_1}^{-1}(y_1)| \geq 2^{r-k'+2\log T - 2\log\epsilon} = 2^{r-s-\log T + \log\epsilon} = M$, the Leftover Hash Lemma tells us that, given y_1, the distribution $h, h(x_1)$ is quasi-random within ϵ/T . On the other hand, a simple counting argument shows that $|{g_1}^{-1}(y_1)| \geq M$ with probability at least $1 - \epsilon/T$, since there are at most 2^s $y's$ with $g_1(y) < M$, and each one is chosen with probability at most $M/2^r = \epsilon/T2^s$. Thus, the distributions y_1, x_2, h and y_1, x'_2, h', where x'_2 and h' are chosen independently at random, are indistisinguishable within $2\epsilon/T$. The induction step follows just as in the proof of Theorem 2.5.5. □

As an application, we can use the hash-based PRG's to save random bits in any algorithm which has one-way access to a random input, and uses significantly less memory than random bits. Examples of such algorithms are the probabilistic log-space algorithm for graph connectivity ([AKLLR]) and the algorithm for approximating the volume of a convex region in [DFK], which uses linear space and n^5 random bits.

Corollary 2.8.3 *Let A be any probabilistic algorithm with one-way access to a T bit random tape which uses space s. Then the distributions on outputs of A using a truly random tape, and that using as a random tape the output of a hash-based PRG with parameters $r, T/r$, and $k' = s + 3\log T - 3\log\epsilon$, are indistinguishable within 2ϵ. (In particular, for $s \geq \log T$ and $\epsilon = 1/T^{O(1)}$, (and, hence, k'=O(s)), letting $r = T^{1/2}s^{1/2}$, the PRG simulates A using $O(r+Ts/r) = O(T^{1/2}s^{1/2})$ random bits.)*

Proof: For a fixed input, view A as an algorithm making T/r calls to subroutines using r bits with output bound s in the following way. The i'th subroutine called simulates A starting at the state A was at after having read $T(i-1)/r$ random bits, and ending after A has read Ti/r random bits. The subroutine then outputs an s bit string describing the current state of A. Once the state of A at the beginning of the sub-procedure is specified, the sub-procedure determines a map from the T/r random bits on the tape read to s bit strings describing the end state. This initial state is completely determined by the outputs of the previous sub-procedures. (In fact, the initial state is the output of the $i-1$'st sub-procedure.) Therefore, by Theorem 2.8.2, the distribution on outputs of the sub-procedures does not change by more than 2ϵ if we replace A's random source with the output of a hash-based PRG with the specified parameters. Thus, the same is true of the output of the last sub-procedure, which is the final state of A. Since this final state determines A's output, the same is true of the output of A. □

We can iterate the above construction to get a PRG for one-way, space-bounded probabilistic machines using much fewer bits.

Definition 2.8.4 *Let r, k, k' and i be positive integers. The* i'th iterated recycling PRG with parameters r, k, k' *is a PRG taking as input $h_1, h_2, ...h_i$, where h_j is a universal hash function mapping $r+(j-1)kk'$ bits to $r+(j-1)kk'-k'$ bits, and an $r+jkk'$ bit seed, x. The output of the PRG, which will be rk^i bits long, is defined recursively as follows. Break x into a $r+(j-1)kk'$ bit string x' and k pads of length k', $s_1, ...s_k$. Let $x_1, ...x_k$ be the outputs of the hash-based PRG with parameters $r+(i-1)kk', k, k'$ on input $h_i, x, s_1, ...s_k$. If $i=1$, output $x_1, ..., x_k$. Otherwise, for each x_j, recursively compute the $i-1$'st iterated recycling PRG with parameters r, k, and k' on input $h_1, ..h_{i-1}$ and x_j. Output their concatenation.*

Note that the iterated recycling PRG uses a total of $O(ri+i^2kk')$ bits as input.

Theorem 2.8.5 *Let $T=rk^i$. Let $k'/geqs+3\log T-3\log\epsilon$. Then for any algorithm A using space s and having one-way access to T random bits, the output of A using a string produced by the iterated recycling PRG with parameters r, k, k' and i, is indistinguishable from that produced by using a truly random string within $4i\epsilon$.*

Proof: Let O_j be the distribution on outputs when A is simulated using a random tape generated as follows. Pick $h_1, ...h_j$ as above. Pick $T/rk^j=T_j$ strings $x_1, ...x_{T_j}$ of length $r+jkk'$ and for each compute the j'th iterated recycling PRG on input $h_1, ...h_j$ and x_l. By convention, O_0 is the distribution on outputs using a truly random string. O_i is that using the output of the i'th iterated recycling PRG. We claim O_j is indistinguishable from O_{j+1} within $4/epsilon$. In fact, this is true for any fixed values of $h_1, ...h_j$. For, when $h_1, ...h_j$ are fixed, we can view O_j as the output of an algorithm which makes T_j calls to a sub-procedure on $r+jkk'$ bits with space bound s. The l'th sub-procedure simulates A between the time A examines the $T(l-1)/T_j$'th bit of its input until it examines the Tl/T_j'th bit, and outputs A's final state in this interval. Since $h_1, ...h_j$ are fixed, these bits are determined by x_l. ¿From the construction of the iterated recycling PRG, the distribution O_{j+1} given $h_1, ...h_j$, is precisely that obtained by replacing the "random tape" of this algorithm $(x_1, ...x_{T_j})$ with the following variation on the hash-based PRG. Every k steps, the modified hash-based PRG throws away its seed entirely, and picks a new seed at random. It is easy to see that the proof of refsub-procedures can be used to show that this modification only makes the PRG better for simulating algorithms with space-bounded sub-procedures. By Theorem 2.8.2, the hash-based PRG with the value for k' chosen would produce a distribution indistinguishable from that using truly random bits within 4ϵ; thus, the same is true for our modification. Hence, O_j and O_{j+1} are indistinguishable within 4ϵ; thus, O_0 and O_i are indistinguishable within $4i\epsilon$, which is what we wanted. □

Corollary 2.8.6 *Let A be an algorithm with one-way access to T random bits using space s. Then A can be simulated using $O((\log T)^2(s + \log T + \log \epsilon))$ bits to within ϵ; in particular, if A is a log-space, polynomial-time algorithm, A can be simulated using $O(\log^3 n)$ bits to within any fixed polynomial fraction, $1/n^c$.*

Proof: Apply the above theorem with $k = 2$, $r = O(\log T)$, and $i = \log T$. □

Chapter 3

Cryptographically Secure Pseudo-Random Generation Based on Any One-Way Function

3.1 Introduction

In this chapter, we apply the techniques and results of the previous chapter to the study of *cryptographically secure* pseudo-random generators. Intuitively, a PRG is cryptographically secure if no polynomial-time algorithm can distinguish its outputs from truly random strings of the same length with a non-negligible probability of success. Unlike the special purpose PRG's we studied in the previous chapter, a cryptographically secure PRG can be used to conserve random bits in *any* polynomial-time algorithm. Also, in general, the size of the seeds needed by cryptographically secure PRG's tend to be, at least for sufficiently large inputs, much shorter than for those needed by the special purpose PRG's we studied in the last chapter. On the other hand, whereas the special purpose PRG's were provably good for the prescribed uses, cryptographically secure PRG's must be based on the intractibility of some problem in NP. Since the P vs. NP question is still very much open, creating PRG's which are provably cryptographically secure is beyond the reach of current knowledge. In fact, since the class P refers to worst-case complexity, and the intractibility assumptions on which cryptographically secure PRG's have been based all concern average-case complexity, it seems that an assumption much stronger than $P \neq NP$ is necessary to establish their existence. In this chapter, and Appendix 2, we present the results of [ILL] and [H]. These results give an exact, natural characterization of the minimal assumption under which cryptographically secure PRG's exist: the existence of a one-way function. (We will later explain the differences between the results in these two papers.)

Besides being useful for probabilistic computation, cryptographically secure PRG's are one of the basic primitives of modern cryptography, with many applications. For example, they can be used to build secure private-key encryption systems ([GM], [GGM], [LR]). Naor [Na] shows how to construct a bit commitment protocol based on a pseudo-random generator. Bit commitment can be used for zero-knowledge proof systems ([GGM]), which can in turn be used for identification ([FS]), among numerous other uses.

A one-way function is, intuitively, one which is easy to compute, but hard to invert, i.e., given $f(x)$, it is difficult to find any x' with $f(x') = f(x)$. A cryptographically secure PRG is easily seen to be a special case of one-way function. Intuitively, if we could, given a pseudo-random string, compute a seed generating the string, then we could distinguish the string from a truly random string. Thus, to be cryptographically secure, a PRG must also be one-way.

While there are many natural problems that are conjectured to be one-way functions, it is hard to think of a natural example of a conjectured cryptographically secure pseudo-random generator. [1] Thus, it is desirable to convert what seems to arise naturally (one-way functions) into a valuable commodity (cryptographically secure pseudo-random generators).

The first construction of a cryptographically secure pseudo-random generator, due to Blum and Micali ([BM]), is based on the intractability of the discrete log problem, a classical problem of number theory. [BBS] give a construction based on the quadratic residuousity problem, another number- theoretic problem. Yao ([Y1]) generalizes this by showing that such a pseudo-random generator can be constructed from any one-way permutation. Levin ([Lev1]) shows that the existence of functions that are one-way on a quadratic number of iterates is necessary and sufficient for the existence of cryptographic pseudo-random generators. [GKL] show that any one-way function for which the preimage sizes of all elements in the range are roughly equal is sufficient. (The actual condition they use is slightly weaker, but more difficult to state. This result allowed PRG's to be constructed based on many non-number theoretic problems, such as subset sum and error correction.)

We now explain the difference between the result proved in this chapter, which appeared in [ILL], and that proved by Hastad in [H], which we present in Appendix 2. There are two natural notions of security for cryptography : uniform and non-uniform. In the uniform model, adversaries or protocol- breaking algorithms are modelled as probabilistic Turing machines; in the non-uniform model, they are circuit families. Corresponding to these two models are two notions of reduction between protocols or primitives. (A protocol is a method

[1] One counter-example, which is somewhat suprising, is the subset sum function, which is a cryptographically secure PRG if and only if it is a one-way function ([IN]).

for different participants to accomplish some cryptographic task; a primitive is a basic building block, such as a one-way function or pseudo-random generator, which can be used in various protocols. We often talk of a adversary to break a protocol or primitive, meaning an algorithm for cheating in the protocol, or to refute the claimed property of the primitive. For example, a breaking algorithm for a one-way function is an algorithm which successfully inverts the function quickly.) We will later discuss the advantages and disadvantages of the two models, but, in general, a uniform reduction is much more desireable than a non-uniform reduction. Intuitively, a non-uniform reduction between protocols or primitives is just an existence proof. To actually be implemented, a cryptographic protocol or primitive must be given a fixed bound on resources to be used, which we will index by a security parameter n. For example, the security parameter for a one-way function might be the length of its input, which implicitly determines an upper bound on the time needed to compute the function. If a non-uniform reduction exists between protocols/primitives X and Y, then, if there is an adversary to break X with security parameter n in time T, then there is an adversary to break Y with security parameter n in time $p(T)$ for a fixed polynomial p. However, a non-uniform reduction does not tell you how, given a description of the algorithm breaking X on security parameter n, to actually find the algorithm which breaks Y on this security parameter. On the other hand, a uniform reduction will not only guarantee the existence of such an algorithm, but will give a method of finding the algorithm in time polynomial in n and T.

In this chapter, we construct a PRG based on any one-way function, and give a non-uniform reduction from inverting the one-way function to distinguishing the PRG's outputs from truly random strings. This result is from [ILL]. Although the reduction is non-uniform overall, the construction is a composition of many small steps; for all but one of these steps, the reduction we give is actually uniform. Our techniques allow us to give simple constructions of PRG's based on many interesting special cases of one-way functions for which uniform reductions exist. Hastad [H] modifies our construction for this step to obtain a uniform reduction for this step as well; replacing our construction for this step with his, and keeping the other steps intact, yields a PRG with a uniform reduction from the one-way function. For completeness, a somewhat simplified version of Hastad's proof is given in Appendix 2.

From now on, we will refer to cryptographically secure PRG's simply as pseudo-random generators, and ignore the existence of special purpose PRG's.

3.2 Security

In this chapter, we consider both uniform and non-uniform models of security. The difference between the two models of security is that, in the uniform model, the adversary is a fast algorithm, whereas, in the non-uniform model, the adversary is a small circuit. At first glance, the non-uniform notion of security seems too strong of a requirement to place on a cryptographic protocol. However, a protocol that is only secure in the uniform sense is susceptible to attack if the adversary has time before the protocol begins to pre-process information. The time and number of processors an adversary has during the pre-processing phase may be much greater than the resources available during the protocol itself, when the adversary must answer questions on line and perhaps is physically restricted to using a "smart card" type computer. The result of the pre-processing, a small circuit breaking the protocol, can then be used to break the protocol within the allowable time on-line. Security in the non-uniform model is equivalent to immunity from this type of attack (see [KL]).

For use in probabilistic algorithms, there are also advantages to the non-uniform model. Pseudo-random generators which are secure in only the uniform sense behave like truly random numbers for most inputs, but may not be good for the worst-case inputs. This is because the worst-case input might code a circuit to break the generator. Thus, Yao [Y1] uses the assumption that a pseudo-random generator with non-uniform security exists in order to conclude that $BPP \subset DTime(2^{n^{\epsilon}})$ for every $\epsilon > 0$.

However, although we claim non-uniform security is needed for cryptosystems to be truly unbreakable, reductions between protocols and/or primitives in the uniform model of security are strictly stronger than the corresponding reductions in the non-uniform model. This is because uniform reductions are of the following form: "From a circuit to break protocol/primitive X on security parameter n with probability ϵ, a circuit to break protocol/primitive Y on security parameter $p(n)$ with probability δ can be constructed *in probabilistic polynomial time*". Non-uniform reductions prove that a small circuit to break Y exists if one exists to break X, but do not provide an effective procedure to construct it.

The convention in theoretical cryptography is to consider any function or protocol which eventually takes a super-polynomial time to break as "secure". This is described by saying that any strategy which breaks the protocol is eventually infeasible. In practice, however, a more quantitative analysis of security is necessary. A model of security which is somewhat more realistic might be as follows. Cryptographic protocols and primitives are designed with a security parameter as an input. For example, for a one-way function, the security parameter would be the length of the input. The total time taken during the protocol or primitive should be polynomial in the security parameter. When implementing a protocol, the protocol designer makes a survey of the resources available to potential adversaries. This should factor in the speed of the adversaries' computers, number of processors available, and the amount of time available to the adversary before breaking the system is worthless, to get a number R (say, the number of gates of a Boolean circuit which can simulate an adversary with these resources). The protocol designer also needs to determine an acceptable risk, ϵ, the probability with which a cheater with the aforementioned resources might cheat successfully. Once these factors are determined, the protocol designer chooses the minimal security parameter k achieving these goals. At the moment, all that can be done is to make educated guesses as to what this parameter should be, based on the best known method for breaking the protocol, but a mature theory of circuit complexity might actually allow us to do so rigorously. (To do this, we would require actual numerical lower bounds, explicitly bounding the number of gates needed to break the protocol with a security parameter k with probability ϵ.) The traditional "super-polynomial" definition translates into being able to use parameter $k = (S/\epsilon)^{\delta}$ for any constant $0 < \delta < 1$. In practice, we usually want something much better than this, more along the lines of $k = \log^{O(1)}(S/\epsilon)$. The reductions given here all preserve security to within a polynomial; if one primitive (say, a one-way function) is secure with parameter $k = f(S, \epsilon)$, the constructed primitive (say, a pseudo-random generator) will be secure using a parameter $poly(k) = poly(f(s, \epsilon))$ for a fixed polynomial $poly$. Although we will not go into the details of calculating the most efficient construction obtainable by out techniques, the construction given here is somewhat inefficient, since the polynomial involved is along the lines of k^4. An open problem is to find a more efficient construction, preferably linear. This is known in the case of a one-way permutation ([GL]), but some inefficiency may be the price of a general construction such as we give here. The method presented here does give us such an efficient construction when the one-way function is regular, i.e., when each output has approximately the same number of inputs mapping to it, and this number can be calculated easily.

In this thesis, we compromise between the unrealistic convention of taking polynomial-time as the measure of feasiblity for an adversary, and the more cumbersome model described above. We will paramatarize the acheivable security by a "resource bound" $R(n)$, but consider feasibility modulo changes by a polynomial, i.e. for any polynomials p and q, $R(p(n))q(n)$ will also be considered feasible limits for an adversary. This brushes over differences, vital in real life, between functions which require fully exponential time to invert, i.e., $2^{O(n)}$, and those which only require time $2^{n^{\gamma}}$, for a constant γ. The conventional definitions are obtained letting $R(n) = n$.

Definition 3.2.1 *A* resource bound *R is a time-computable, monotonic increasing function from N to N. Since any one-way function can be inverted in exponential time by exhaustive search, to avoid trivial cases, we assume, for every $\epsilon > 0$, $R(n) < 2^{n^{\epsilon}}$ for all but finitely many n. A function R' is* feasible *with respect to R if $R'(n) < R(p(n))q(n)$ for some polynomials p and q. A function P from N to $[0, 1]$ is* negligible *with respect to R if, for all polynomials p and q, and all but finitely many n, $P(n) < 1/R(p(n))q(n)$.*

Comment: From now on, we will define security with respect to a fixed but unspecified resource bound. As this resource bound will not affect the actual construction of a PRG based on a one-way function, we will be giving one pseudo-random generator which is secure with respect to all resource bounds the original one-way function was secure for. For example, if the original one-way function takes time $2^O(n)$ to invert, it will be secure against all sub-exponential resource bounds, and so will the resulting pseudo-random generator.

Definition 3.2.2 *A* uniform *algorithm is a probabilistic Turing machine. The* time bound $T(n)$ *for a uniform algorithm is the maximum running time on inputs of length* n*. A* non-uniform *algorithm is a pair of Turing machines* A *and* M*.* A *is a preprocessing algorithm that on input of length* n *produces a string* $A(n)$*. The running time of* $A(n)$ *is not limited. Algorithm* M *accepts as input* x *and* $A(|x|)$*. The* time bound $T(n)$ *for a non-uniform algorithm is maximum running time of* M *over all inputs* x *and* $A(|x|)$*.*

From the results of [KL], a non-uniform algorithm with time bound in resource class R is equivalent to a (recursive) family of circuits with the size of the circuit for inputs of length n bounded by a function $r(n)$ where r is in resource class R.

Definition 3.2.3 *A uniform (non-uniform) algorithm is* feasible *with respect to resource class* R *if the time bound function is feasible with respect to* R*.*

For the remainder of the paper, unless stated otherwise, a feasible adversary algorithm is always with respect to an arbitrary but fixed resource class R. For each cryptographic task that we define, there are implicitly two definitions being made simultaneously, one with respect to the uniform and the other with respect to the non-uniform model of security. Unless otherwise stated, each definition, lemma, proposition and theorem has two versions, one in each model.

3.3 One-way functions

Intuitively, a function f is *one-way* if it is easy to compute but hard to invert. More precisely, given x, the value of $f(x)$ can be computed in polynomial-time, but every feasible algorithm that receives as input $f(x)$ (when x is a randomly chosen string of length n) can output a y such that $f(y) = f(x)$ with only negligible probability. It has not yet been proven that one-way functions exist (if $P = NP$ then they certainly do not exist, but even if $P \neq NP$ it is not clear that they exist), but there are many examples of functions that seem to be one-way in practice and that are widely conjectured to be one-way. Some examples of conjectured one-way functions are factoring a composite number N that is the product of large randomly chosen primes, square roots modulo such an N, discrete log modulo a large randomly chosen prime, problems from coding theory, and the subset sum problem.

Definition 3.3.1 *A* length (function) $l(n)$ *is a monotone increasing function from N to N such that $l(n)$ is computable in time polynomial in n. A* function f *with input length $m(n)$ and output length $l(n)$ specifies for each $n \in N$ a function $f_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^{l(n)}$. For simplicity, we write $f(x)$ in place of $f_n(x)$. We say that f is* polynomial-time computable *if there is a polynomial-time Turing machine that on input $x \in \{0,1\}^{m(n)}$ computes $f(x)$. A* probability ensemble *D with length $m(n)$ assigns to each positive integer n a probability distribution D_n on bit strings of length $m(n)$. The* uniform ensemble *U assigns to each positive integer n the uniform probability distribution U_n on strings of length n. For $X \subseteq \{0,1\}^{m(n)}$, $D[X]$ is the sum over all $x \in X$ of the probability of x with respect to D_n. We use the notation $x \in_D \{0,1\}^{m(n)}$ to mean that x is randomly chosen from $\{0,1\}^{m(n)}$ according to D_n. We say that D is* polynomial-sampleable *if there is a polynomial-time Turing machine M with input length $k(n)$ and output length $m(n)$ such that, for each $n \in N$, $M(x) \in_D \{0,1\}^{m(n)}$ when $x \in_U \{0,1\}^{k(n)}$. Define $f(D)$ to be the probability ensemble with length $l(n)$, where $f(D_n)$ is the probability distribution defined by the random variable $f(x)$ when $x \in_D \{0,1\}^{m(n)}$.*

Note : Hereafter, unless stated otherwise, f is always a polynomial-time computable function with input length n and output length $l(n)$ and D and E are always probability ensembles with length n.

Definition 3.3.2 *We say that f is* somewhat one-way on D *if, for some constant $c > 0$, for every feasible algorithm M, the inverting probability* $\Pr[f(x) = f(M(f(x)))]$ *when $x \in_D \{0,1\}^n$ is at most $1 - 1/n^c$. We say that f is* one-way on D *if, for every feasible algorithm M, the inverting probability* $\Pr[f(x) = f(M(f(x)))]$ *when $x \in_D \{0,1\}^n$ is negligible. We say that f is* one-way *if f is one-way on U.*

Proposition 3.3.3 *If there is a polynomial-sampleable D such that f is one-way on D, then there is a one-way function g.*

The function g is simply the composition of f and the polynomial-time computable sampling function for D. This proposition allows us to just consider one-way functions on the uniform ensemble.

Notation : Let $q(n)$ be a length function. Let D^q be the ensemble with length $n \cdot q(n)$ such that D^q_n is the distribution obtained by independently sampling $q(n)$ times from D_n and concatenating the results. Similarly, let f^q be the function with input and output lengths $n \cdot q(n)$ and $l(n) \cdot q(n)$, respectively, given by: $f^q(x_1 \circ \ldots \circ x_{q(n)}) = f(x_1) \circ \ldots \circ f(x_{q(n)})$, where $x_1, \ldots, x_{q(n)} \in \{0,1\}^n$.

The following is implicit in [Y1].

Proposition 3.3.4 *If f is somewhat one-way on D with associated constant $c > 0$, then f^q is one-way on D^q, where $q(n) = n^{c+1}$.*

3.4 Pseudo-random generators

Informally, a polynomial-time computable function f is *pseudo-random* if $f(x)$ is strictly longer than x and if every feasible algorithm can distinguish $f(x)$ from a truly random string of the same length (when x is chosen randomly) with only negligible probability. Intuitively, $f(x)$ "looks" just like a random string to any feasible algorithm, even though it is generated from a string x that is strictly shorter. This intuition is captured in the following definition of [BM], [Y1].

Definition 3.4.1 *f is a* generator *if $l(n) > n$ for all $n \in N$. The* distinguishing probability *$p(n)$ of algorithm M for f is $|\Pr[M(f(x)) = 1] - \Pr[M(y) = 1]|$ when $x \in_U \{0,1\}^n$ and $y \in_U \{0,1\}^{l(n)}$. We say that f is* pseudo-random *if every feasible algorithm has a negligible distinguishing probability for f.*

The normal definition of a pseudo-random generator insists that the generator can stretch the input by any polynomial amount. The following shows the definition above is equivalent, and follows from [GGM]. Although this is a well-known result, we give the proof, since we will need to refer to it later on.

Proposition 3.4.2 *If there is a pseudo-random generator f then, for every constant $c > 1$, there is a pseudo-random generator with input length n and output length n^c.*

Proof: Without loss of generality, assume $l(n) = n + 1$. (If $l(n) > n + 1$, just use the first $n + 1$ bits of $f(x)$.) Consider the function g computed as follows. On input $x = x_0$, compute $f(x_0)$; call the first bit b_1 and the remaining n bits x_1. Recurse, using x_1 in place of x_0, thus obtaining b_2 and x_2. Repeat for n^c iterations. Output $b_1, ...b_{n^c}$. We claim this is function is cryptographically secure as a PRG. Assume not; let A be any feasible algorithm with a $\epsilon(n)$ distinguishing probability between this function and a truly random sequence of bits. For $0 \leq i \leq n^c$, let D_i be the distribution where $b_1, ...b_i$ are chosen at random, x_i is chosen at random, and $b_{i+1}, ...b_{n^c}$ are generated from x_i as in the recurrence above. Note that D_0 is the distribution output by g on a randomly chosen input, and D_{n^c} is the uniform distribution on $\{0,1\}^{n^c}$. Thus, if we let $A[D]$ denote the probability that $A(x) = 1$ when x is chosen according to D, we have $|A[D_0] - A[D_{n^c}]| > \epsilon(n)$. Thus, for some $i, 0 \leq i \leq n^c - 1$, we have $|A[D_i] - A[D_{i+1}]| > \epsilon(n)/n^c$. This gives us a way of distinguishing between $f(x)$ and a truly random $n + 1$ bit string. On input $y, |y| = n + 1$, pick random bits $b_1, ..b_i$, and let b_{i+1} be the first bit of y, and x_{i+1} be the remaining n bits.

Use x_{i+1} to generate $b_{i+2}, ...b_{n^c}$ as before. Simulate A on $b_1, ..b_{n^c}$. If y is picked truly at random, $b_1, ..b_{n^c}$ are distributed according to D_{i+1}; if $y = f(x)$ for a randomly chosen x, the b_i's are distributed as in D_i. Thus, our algorithm run with parameter i distinguishes between random and pseudo-random strings with probability at least ϵ/n^c. We can either choose i at random, or pick the best value for i by estimating, via random sampling, $A[D_i]$ for each value of i. Thus, since f is, by assumption, pseudo-random, $\epsilon(n)$ must be negligible.

We can now state our main theorem.

Theorem 3.4.3 *If there is a one-way function in the non-uniform model of security then there is a pseudo-random generator in the non-uniform model of security.*

Like all the previous constructions of pseudo-random generators, our construction uses the notion of a hidden bit of a function, introduced in [BM].

Definition 3.4.4 *A* bit (function) *b is a polynomial-time computable function with input length n that outputs a single bit. The* distinguishing probability *$p(n)$ of algorithm M for b on D given f is $|\Pr[M(f(x)) = b(x)] - \Pr[M(f(x) \neq b(x))]|$ when $x \in_D \{0,1\}^n$. b is* hidden *for f on D if every feasible algorithm has negligible distinguishing probability for f on D. b is* hidden *for f if it is hidden for f on U.*

Notation (inner product bit) : Let $x, y \in \{0,1\}^n$. Let $f'(x \circ y) = f(x) \circ y$ and let $D'_n = D_n \circ U_n$. It is easy to see that if f is one-way on D, then f' is one-way on D'. The *inner product* bit is $b(x \circ y) = x \odot y$.

The following is due to Goldreich and Levin ([GL]). It is extensively used in our construction.

Theorem 3.4.5 (Goldreich and Levin) *Assume that f is one-way on D. Let f', D' and bit b for f' be defined in terms of f and D as in the inner product bit definition. Then, b is hidden for f' on D'.*

The idea of a function that hides a bit was introduced in the original construction of a pseudo-random generator [BM] and has been central to all such constructions since that time. Theorem 3.4.5 presents an elegant, simple and general method of obtaining a hidden bit from a one-way function. However, naive use of this theorem with an arbitrary one-way function has difficulties because of the following. Let f be any one-way function. Define one-way function f' as $f'(x \circ y) = f(x)$, where $|y| = |x|$. From Theorem 3.4.5,

$f''(x \circ y \circ r) = f(x) \circ r$, where $|r| = |x \circ y|$, hides the inner product bit. However, for cryptographic purposes this is useless, as the hidden bit is informationally impossible to recover. The hidden bit is only cryptographically useful if it is also *meaningful* as defined below. (We later show how to use Theorem inner product to extract bits which are both hidden and meaningful from one-way functions.)

Definition 3.4.6 *Let b be a bit function for f. Then b is* meaningful *for f on D if there is a constant $c > 0$ so that there is an algorithm (not necessarily a feasible algorithm) which predicts b correctly, given $f(x)$ for $x \in_D \{0,1\}^n$, at least a $1/2 + 1/n^c$ fraction of the time. Equivalently, for $n > 0$ an integer, let S_n be the set of n bit strings x so that $b(x) = 0$, and T_n be the set of n bit strings x so that $b(x) = 1$. Then b is meaningful if, for every n, the distribution on $f(x)$ when x is chosen according to $D_n|S_n$ is statistically distinguishable from the distribution on $f(x)$ when x is chosen according to $D_n|T_n$ by at least $1/n^c$. The bit b is* meaningful *for f if it is meaningful for f on U.*

3.5 Computational Indistinguishablity

Computational indistinguishability is the computational analog of statistical indistinguishability. Intuitively, computationally indistinguishable distributions "look" alike to feasible algorithms. Cryptographically secure PRG's are a special case: a length-increasing, poly-time function is a PRG if its output ensemble is computationally indistinguishable from the uniform ensemble. The following definition appears in [GM], [Y1], and [GMR].

Definition 3.5.1 *Let D and E be probability ensembles with the same length function. The* distinguishing probability function $p(n)$ *of algorithm M for D and E is $|\Pr[M(x) = 1] - \Pr[M(x') = 1]|$ when $x \in_D \{0,1\}^n$ and $x' \in_E \{0,1\}^n$. D is* computationally indistinguishable *from E if every feasible algorithm has a negligible distinguishing probability.*

The following two propositions are the crucial point in this paper where constructions involving the uniform and non-uniform models of security diverge. Although analogs of each other, these two propositions have a subtle difference in addition to the notions of security involved. Both say, intuitively, that, if two ensembles are computationally indistinguishable, then many samples from one ensemble are indistinguishable from many samples of the other. If the ensembles in question are both *polynomial-sampleable*, then Proposition 3.5.2 says this is true with respect to both models of security. However, Proposition 3.5.3 says that in the non-uniform model, this is also true for *arbitrary* ensembles. The reason for this difference is as follows. In the uniform model of security, receiving several samples from an ensemble that is not polynomial-sampleable might give an adversary information that it could not compute itself. This extra information might allow the adversary to be able to distinguish between the two ensembles on the next sample. However, this kind of information can never be helpful to a non-uniform adversary, since it is succinctly describable (being extracted from a polynomial number of polynomial length samples) and hence could be included in the non-uniform "advice" string. This distinction has repercussions throughout the paper and is ultimately the reason why our proof of Theorem 3.4.3 does not hold in the uniform model of security.

The following propositions appear in [GMR]. Let $q(n)$ be a length function.

Proposition 3.5.2 *If D and E are* polynomial-sampleable *probability ensembles that are computationally indistinguishable (in either model of security). then D^q and E^q are computationally indistinguishable (in the same model of security as before).*

Proof: Let A be an algorithm distinguishing between D^q and E^q with distinguishing probability $\epsilon(n)$. Fix a length n. Let the distribution F_i be that given by taking i random samples from D_n and $q-i$ samples from E_n and concatenating them. Thus, $F_0 = D^q{}_n, F_q = E^q{}_n$. Since A distinguishes between F_0 and F_q with probability $\epsilon(n)$, there is some $i, 0 \leq i < q$ so that A distinguishes between F_i and F_{i+1} with probability at least $\epsilon(n)/q(n)$. Let A'_i be the probabilistic algorithm which, on input x generates $i-1$ random samples from $D_n, d_1, ...d_{i-1}$, $q-i$ elements from $E_n, e_{i+1}, ..., e_q$, and simulates A on the input $d_1 \cdot d_2 ... \cdot d_{i-1} \cdot x \cdot e_{i+1} ... \cdot e_q$. If x is drawn from the distribution D, $A'{}_i$ has the same probability of acceptance as does A on F_i, whereas if x is chosen according to E_n, the probability of acceptance is the same as that of A on F_{i+1}. Thus, A'_i distinguishes between D and E with probability at least $\epsilon(n)/q(n)$. Given i, the algorithm A'_i is polynomial-time. To find an appropriate value for i with high probability, for each possible i, run A'_i on random samples from both D_n and E_n and estimate the distinguishing probability; let i be the number with $0 \leq i < q(n)$ which has the greatest estimated distinguishing probability.

Proposition 3.5.3 *If D and E are* arbitrary *probability ensembles that are computationally indistinguishable in the* non-uniform *model of security, then D^q and E^q are computationally indistinguishable in the non-uniform model of security.*

Proof: Let A be a circuit distinguishing between D_n and E_n with probability at least ϵ. We can find an $i, 0 \leq i < q(n)$ and an "algorithm" A'_i as above; however, since D and E are no longer necessarily polynomially sampleable, A'_i is not necessarily feasible. However, since the expected distinguishing probability for A'_i is at least $\epsilon/q(n)$, when $d_1 ... d_{i-1}, e_{i+1}, ... e_q$ are chosen as above, there must be at least one vector of values which achieves the expectation. Since this vector is polynomial size, and since, given such a vector, A_i' is feasible, the vector can be "hard-wired" into A_i' to yield a circuit of size at most that of A .

3.6 Computational Entropy

Intuitively, pseudo-random generators transform a small amount of randomness into a much larger string that is random for all practical purposes. Of course, in information-theoretic terms, no such increase is possible. Applying a fixed function to a string can only decrease its informational content, not increase it. (Formally, if D is a distribution and f a function on the same finite set, the Shannon entropy of D is at least as large as that of the induced distribution $f(D)$.) Thus, in some sense, any pseudo-random generator determines an ensemble that (asymptotically) has a greater *computational* entropy than it has Shannon entropy. In this section, we formalize this intuitive notion of the computational entropy of an ensemble. This definition provides one of the main conceptual tools in this chapter.

Just as Shannon entropy quantifies the amount of randomness in a distribution, computational entropy quantifies the amount of "apparent" randomness (to feasible algorithms) of a distribution. For example, we can relax the notion of a generator f being pseudo-random by allowing its output to be computationally indistinguishable from an ensemble D that is not necessarily the uniform ensemble, but which has more Shannon entropy than the input to f. We call such an f a *pseudo-entropy generator*. In this case, we say that the computational entropy of f is at least the Shannon entropy of D.

The notion of computational entropy is also useful in the case when the Shannon entropy of D is not necessarily greater than that of the input to f. We say that f has *false* entropy if the computational entropy of f exceeds the Shannon entropy of f. (but not necessarily the Shannon entropy of the input to f).

We use computational entropy in our construction of a pseudo-random generator as follows. For convenience, we present our construction in reverse order; instead of starting with a one-way function, and constructing more complicated functions from it, we will present our proof as reducing the task of constructing a cryptographically-secure PRG to that of constructing weaker types of functions. First, we show how to use a pseudo-entropy generator to construct a pseudo-random generator. The next step is to show how to convert any function with false entropy into a pseudo-entropy generator. These steps turn out to be almost immediate consequences of the results proved in Chapter 2. Finally, we prove that any one-way function in the non-uniform sense can be used to construct a function with false entropy in the non-uniform sense, thus completing the proof of Theorem 3.4.3. Hastad's analagous result in the uniform case uses the first two steps, and part of the last step, but must in addition go to quite a bit of trouble (and develop some clever and interesting proof techniques) to give a construction of a *uniform* false entropy generator from a one-way function.

In the following, $s(n)$ is a function from N to the non-negative reals.

Definition 3.6.1 *We say f has* uniform computational entropy at least $s(n)$ *if there is a* polynomial-sampleable *ensemble D such that D is computationally indistinguishable (in the uniform sense) from $f(U)$ and $Ent(D_n) \geq s(n)$.*

Definition 3.6.2 *We say f has* non-uniform computational entropy at least $s(n)$ *if there is an (*arbitrary*) ensemble D such that D is computationally indistinguishable (in the non-uniform sense) from $f(U)$ and $Ent(D_n) \geq s(n)$.*

The difference between these two definitions is necessary so that the following proposition holds in both models of security.

Proposition 3.6.3 *Let $q(n)$ be a length function. If f has computational entropy at least $s(n)$, then f^q has computational entropy at least $s(n) \cdot q(n)$.*

Proof : The uniform case follows from Proposition 3.5.2 and the non-uniform case follows from Proposition 3.5.3. □

Definition 3.6.4 *We say f is a* pseudo-entropy generator *if there is a constant $c > 0$ such that f has computational entropy at least $n + 1/n^c$.*

Definition 3.6.5 *We say f has* false entropy *if there is a constant $c > 0$ such that f has computational entropy at least $Ent(f(U)) + 1/n^c$.*

3.7 Getting Rid of Unknown Parameters

At certain points in our construction, we will need to "pretend" we could compute certain parameters such as, for example, the entropy of the distribution $f(U_n)$. In fact, we cannot guarantee that these parameters are easily computable. However, we can show that, if the parameter has only a polynomial number of possible values, this problem can be surmounted. A function's being computable with access to a parameter with a polynomial number of possible values is equivalent to its being computable in P/log (see [KL]). Intuitively, f is "computable from a parameter" if there is a polynomial-size family of polynomial-time computable functions so that on any length, one member of the family agrees with f.

Definition 3.7.1 *Let f be a function (not necessarily polynomial-time computable). We say that f is computable* from a parameter *in polynomial-time if there is a polynomial-time computable function f' with two inputs, and a constant c, so that, for every length n, there is a $y_n, 1 \leq y_n \leq n^c$ with $f(x) = f'(x, y_n)$ for every $x \in \{0,1\}^n$. We call f' together with the sequence $y_i, i \in N$* a realization *of f, and the constant c, the associated constant for the realization. We define the concepts of one-way functions computable from a parameter, computational entropy of a function computable from a parameter, and cryptographically secure PRG computable from a parameter, as before, except that in each case, the adversary has access to the appropriate value of y_i, for some realization of the function. Also, in the definition of the uniform computational entropy of a function computable from a parameter, we allow the indistinguishable ensemble to also be computable from the same parameter, i.e., the distribution for input length n can be sampled from in polynomial-time by a machine given n and y_n.*

Proposition 3.7.2 *If there is a PRG computable from a parameter, with a realization with associated constant c, then, for any constant $d > 1$, there is a PRG computable from a parameter, with associated constant c, so that $l(n) > n^d$.*

Proof: The proof is exactly the same as for Proposition 3.4.2 ([GGM]).

Lemma 3.7.3 *If there is a PRG computable from a parameter, then there is a PRG.*

Proof: Let f be a PRG computable from a parameter and let $f', \{y_i | i \in N\}$ be a realization of f with associated constant c. By the above proposition, we can assume without loss of generality that $l(n) > n^{c+2}$. Then let g be the polytime computable function which takes as input n^c strings of length n, say $x_1, \ldots. x_{n^c}$, computes, for each $1 \leq i \leq n^c$, $f'(x_i, i)$, and outputs their bitwise exclusive-or. Since g inputs n^{c+1} bits and outputs $l(n) > n^{c+2}$ bits, g is a generator. We claim that g is a PRG. Let M be a feasible adversary; assume that M has a non-negligible distinguishing probability for g. Let $M'(z, j)$ be the probabilistic algorithm that generates a random n bit string x_i for every $1 \leq i \leq n^c, i \neq j$, computes $z_i = f'(x_i, i)$ for each such i, lets w be the bitwise exclusive-or of all the z_i's and z, and simulates M on w. For any fixed j, and in particular, $j = y_n$, the distribution on w when $z = f(x, j)$ for a randomly chosen n bit string x is exactly the distribution on outputs of g. On the other hand, if z is chosen uniformly at random, then w is also uniformly distributed. Thus, the distinguishing probability of M' for f is exactly the same as that of M for g. But since f is a PRG computable from a parameter, this probability must be negligible, contrary to assumption. □

Because of the above Lemma, it suffices to construct a pseudo-random generator computable from a parameter from a one-way function in order to prove our main theorem.

3.8 Construction of a PRG from a one-way function

Our construction is composed of three steps: first, use the one-way function to construct a function with false entropy; second, convert the false entropy into pseudo-entropy; and finally, convert a pseudo-entropy generator into a pseudo-random generator. Since the first step is the most difficult, we present the construction in reverse order. In other words, instead of showing the construction starting from a one-way function and building up to a PRG, we instead present the proof as a gradual weakening of our final goal.

We first show that it suffices to construct only a pseudo-entropy generator, rather than a pseudo-random generator; i.e., given the former, we can construct the latter. Both types of generator produce outputs which in some sense have more randomness than the amount of bits input; the difference is that the randomness of the output of a pseudo-entropy generator may be that of an arbitrary, hard to understand distribution. We must somehow take this unknown random source and use it to obtain (almost) the same amount of randomness in a uniformly distributed form. This general problem, the use of an unknown random source to obtain truly random bits, is the subject of many papers ([B1], [SV], [VV], [CG], etc.). In general, it is impossible to obtain even one random bit from such a source alone ([SV]). However, we will use the Leftover Hash Lemma from Chapter 2 to show that a source of truly random bits can be used to extract almost all the randomness from an unknown source, without compromising the truly random bits. This will suffice for our construction. The intuition is that the Leftover Hash Lemma provides a way of converting distributions with a certain amount of inherrent randomness to almost the same amount of uniform randomness. Applying the same technique to a distribution which looks like it has inherrant randomness will produce bits that look uniform. A technical problem is that the Leftover Hash Lemma holds for distributions with a low collision probability, rather than for distributions with high entropy. The following shows that this problem can be avoided by taking many samples from the distribution, rather than just one.

Lemma 3.8.1 *Let D be a distribution on strings of length n, and let E be the entropy of D. Let $k > 8$. Then there is a distibution C so that: the collision probability of C is at most $2^{-(kE-2nk^{2/3}-nk2^{-n}-1)}$, and C is statistically indistinguishable from D^k within $2k2^{-n}+2e^{-k^{2/3}/2}$.*

Proof : Let X be the set of strings x of length n with $D(x) > 2^{-2n}$. Since there are at most 2^n strings not in X, and each has probability at most 2^{-2n}, we must have $D(X) \geq 1-2^{-n}$. Let $D' = D|X$. A simple calculation shows that D and D' are statistically indistinguishable within 2^{-n+1}. Thus, D^k and ${D'}^k$ are statistically indistinguishable within $2k2^{-n}$. Let E' be the entropy of

D'. Then $E = D(X)E' + D(0,1^n - X)Ent(D|0,1^n - X) \leq E' + n2^{-n}$, since no distribution on $0,1^n$ has more than n bits of entropy.

Consider the random variable $V = -\log(D'^k(d_1, d_2, ...d_k) = -\log(D'(d_1)) + -\log(D'(d_2)) + ... - \log(D'(d_k))$ where $d_1, ...d_k$ are chosen acoording to D'. Then V is the sum of k independent random variables, each with expectation E', and each bounded by $2n$, since no element of X has probability less than 2^{-2n} under D'. Therefore, applying Chernoff Bounds, we have that $V \geq kE' - 2nk^{2/3}$ with probability at least $1 - e^{-k^{2/3}}$. Let Y be the set of sequences $d_1, ...d_k$ where this is true, i.e. for which $D'^k(d_1, ...d_k) \leq 2^{-kE' - 2nk^{2/3}}$. Let C be the distribution $D^k|Y$. Since $D'^k(Y) \geq 1 - e^{-k^{2/3}/2}$, C is statistically indistinguishable from D'^k within $2e^{-k^{2/3}/2}$, and hence from D^k within $2k2^{-n} + 2e^{-k^{2/3}/2}$. Also, for every sequence $d_1, ..d_k \in Y$, $C(d_1, ..d_k) \leq 2^{-kE' - 2nk^{2/3}}/D'^k(Y) \leq 2^{-kE' - 2nk^{2/3}}/1/2 \leq 2^{-(kE - nk2^{-n} - 2nk^{2/3} - 1)}$. Since C assigns no element a probability greater than this amount, the collision probability of C is at most this amount, since if we take two random elements from C, whatever we picked for the first element, the conditional probability that the second is the same is at most this amount. $\square$

Corollary 3.8.2 *Let D be a distribution on strings of length n with entropy E. Let $k > 8$ and let $m \leq kE - 4nk^{2/3} - nk2^{-n} - 1$. Let H be a family of universal hash functions from kn bits to m bits. Then the distribution $(h, h(d_1 \odot ...d_k))$ where h is chosen randomly from H and $d_1...d_k$ are chosen according to D, is quasi-random within $2k2^{-n} + 2e^{-k^{2/3}/2} + 2^{-nk^{2/3}}$.*

Proof : Combine the above Lemma with the Leftover Hash Lemma. $\square$

Lemma 3.8.3 *If there is a pseudo-entropy generator (computable from a parameter), then there is a pseudo-random generator (computable from a parameter).*

Proof : Let f be the (uniform/non-uniform) pseudo-entropy generator, and let D be the (polynomial-sampleable/arbitrary) probability ensemble with $Ent(D_n) \geq n + n^{-c}$ that is computationally indistinguishable from $f(U)$ (with length $l(n)$). Let $k(n) = n^{3c+2}$. By Proposition 3.5.2 / 3.5.3, $f^k(U)$ is computationally indistinguishable from D^k. Note that $Ent(D_n^k) = k(n) \cdot Ent(D_n) \geq k(n) \cdot (n + n^{-c})$. Let $j(n) = k(n) \cdot (n + n^{-c}) - 2n \cdot k(n)^{2/3}$. Let $h \in_U H_{k(n) \cdot l(n), j(n)}$. Let D'_n be the probability distribution defined by $h \circ h(y_1 \circ \ldots \circ y_{k(n)})$, where, for all $i = 1, \ldots, k(n)$, $y_i \in_D \{0,1\}^{l(n)}$ independently. By the above Lemma, D'_n is quasi-random within an exponentially small in n amount. Let the generator be defined by $g(h \circ x_1 \circ \ldots \circ x_{k(n)}) = h \circ h(f(x_1) \circ \ldots \circ f(x_{k(n)}))$. Let E'_n be the probability distribution defined by the output of g when the input is $h \in_U H_{k(n) \cdot l(n), j(n)}$ and, for all $i = 1, \ldots, k(n)$, $x_i \in_U \{0,1\}^n$ independently.

Then, since $f^k(U)$ is computationally indistinguishable from D^k, it follows that E' is computationally indistinguishable from D'. (If M' has a non-negligible distinguishing probability for E' and D', let M be the algorithm which given an input x of length $l(n) \cdot k(n)$ picks a random $h \in_U H_{k(n) \cdot l(n)}, j(n)$, and simulates M' on $h, h(x)$. If x is chosen according to $f^k(U)$, the input to M' is distributed according to E'; if x is chosen according to D^k, the input to M' is distributed according to D'. Thus, M and M' have the exact same distinguishing probabilities.)

Because D' is quasi-random within an exponential small in n amount, and because by choice of $k(n)$ the output of g is longer than the input, g is a pseudo-random generator. $\Box$

Our next goal is to transform a function f with false entropy into a pseudo-entropy generator g. The major obstacle is that f could be many-to-one. In this case, even though the output of f seemingly has more entropy than it really has, the Shannon entropy of the output of f may be much less than the length of the input; intuitively the application of f to the input may cause more of a loss in Shannon entropy than the corresponding gain in false entropy. However, we can use the method in Chapter 2 (Theorem 2.6.4) to sample from the distribution $f(x)$ a polynomial number of times using only slightly more than the Shannon entropy number of bits per sample. The computational entropy of a distribution is independent of the method used to sample from it; so, by Proposition 3.6.3, the distribution has more computational entropy per element generated than Shannon entropy. Hence, our method of generating many samples from the distribuiton has a greater computational entropy than the number of bits it uses, and is hence a pseudo-entropy generator. In order to use the sampling method from Chapter 2, we need a good estimate of the entropy of the distribution. This is one point where we need to use a parameter. Since we will not need the exact entropy, but only a good approximation, we can treat this as a parameter with only polynomially many possible values.

Lemma 3.8.4 *Assume there is a polynomial-time computable (from a parameter) function f with at least n^{-c} bits of false entropy. Then there is a pseudo-entropy generator computable from a parameter.*

Proof : Let $E(n)$ be the entropy of the distribution $f(U_n)$ rounded up to the nearest $1/n^{c+1}$. From Theorem 2.6.4, there is a polynomial $k(n)$ and a polynomial time computable function g which, given n and $E(n)$, outputs $k(n)$ samples from the distribution $f(U_n)$ using a total of $k(n)(E(n) + 1/n^{c+1})$ random bits, so that the distribution $g(y)$ is statistically indistinguishable from the distribution $f^k(U_n)$ within an exponentially small amount. The latter distribution, by Proposition 3.6.3, has computational entropy at least $k(n)(Ent(f(U_n)) + 1/n^c) \geq k(n)(E(n) + 1/n^c - 2/n^{c+1}) \geq k(n)(E(n) + 1/2n^c)$ for all but finitely many n. Therefore, g also has this amount of computational entropy, and is hence a pseudo-entropy generator, since this amount is larger than the number of random bits g inputs for all but finitely many n.

As a technical point, we must show that knowing $E(n)$ does not help the adversary. This is handled differently in the uniform and non-uniform cases. In the non-uniform case, $E(n)$ could be "hard-wired" into an adversary's circuit. In the uniform case, there is a polynomial-time algorithm for sampling from the indistinguishable ensemble. The adversary tries all possible values for $E(n)$, and with each calculates the distinguishing probability obtained by trying the resulting algorithm on many samples from D and from $f(U_n)$. The adversary picks the value for $E(n)$ yielding the best result. $\square$

Corollary 3.8.5 *If there is an f (computable from a parameter) that has false entropy at least n^{-c} for some constant $c \geq 0$, then there is a pseudo-random generator.*

We have reduced the problem of constructing a pseudo-random generator to the problem of finding a function with false entropy. We now show, in the non-uniform model, how to construct a function g with false entropy from *any* one-way function f. Let D be a probability ensemble that is computationally indistinguishable from $g(U)$. The construction has the property that if there is a (uniform/non-uniform) feasible algorithm for distinguishing D from $g(U)$ then there is a (uniform/non-uniform) feasible algorithm for inverting f. The only problem is that D is not polynomial-sampleable. In particular, even if $g(U)$ is computationally indistinguishable from D in the uniform model, we see no way to prove that $g^q(U)$ is computationally indistinguishable from D^q in the uniform model (see Proposition 3.5.2).

Lemma 3.8.6 *If there is a one-way function f (in the non-uniform sense), then there is a polynomial-time computable function g with false entropy at least $1/3n$ (in the non-uniform sense).*

Proof : Let $l(n) = \lceil \log(n+1) \rceil$ and let $k(n) = 2^l(n) - 1$. Define g by

$$g(r \circ i \circ h \circ x) = r \circ i \circ h \circ f(x) \circ (h(x) \uparrow (i + l(n))) \circ r \odot x,$$

where $r, x \in \{0,1\}^n$, $i \in \{0,1\}^{l(n)}$ (i is to be thought of as a number between 0 and $k(n)$) and $h \in H_{n,k(n)+l(n)}$. Let $m(n) = |r \circ i \circ h \circ x|$. Let U'_n be the uniform probability distribution strings of length $m(n)$. We describe a probability ensemble D that is computationally indistinguishable from $g(U')$ such that $Ent(D_n) \geq Ent(g(U'_n)) + 1/3n$. Although D is not going to be polynomial-sampleable, it is easiest to think of D as being generated from $r \circ i \circ h \circ x \in_{U'} \{0,1\}^{m(n)}$ and from independently chosen bit $\beta \in_U \{0,1\}$. Define

$i(x) = \lfloor \log(|f^{-1}(f(x))|) \rfloor$. $D(r \circ i \circ h \circ x \circ \beta)$ is exactly the same as $g(r \circ i \circ h \circ x)$ except for possibly the last bit, which in g is always $r \odot x$. The last bit of $D(r \circ i \circ h \circ x \circ \beta)$ is $r \odot x$ unless $i = i(x)$, in which case it is β.

Let ensemble E be such that E_n is the distribution on $i(x) \circ h \circ x$ when $x \in_U \{0,1\}^n$ and $h \in_U H_{n,k(n)+l(n)}$. We define

$$g'(i \circ h \circ x) = i \circ h \circ f(x) \circ (h(x) \uparrow (i + l(n))).$$

Claim : g' is one-way on E.

Proof of Claim : Assume there is an algorithm M with time bound $T(n)$ that inverts $g'(i \circ h \circ x)$ with probability at least $p(n) = 1/T(n)$ for some function $T(n)$ in resource class R when $i \circ h \circ x$ is randomly chosen according to E_n, i.e. on input $i(x) \circ h \circ f(x) \circ (h(x) \uparrow (i(x) + l(n)))$, M finds y such that $f(y) = f(x)$ and $h(y) \uparrow (i(x) + l(n)) = h(x) \uparrow (i(x) + l(n))$. We use only the fact that $f(y) = f(x)$ in the proof of the claim. We construct an algorithm M' with time bound polynomial in $T(n)$ and n that inverts $f(x)$ with probability at least $p(n)/2$. Let $j(n) = 8\log(T(n))$. On input $f(x)$, M' runs through all possible $i = 0, \ldots, k(n)$. For each i, M' chooses $h \in_U H_{n,k(n)+l(n)}$ and $s \in_U \{0,1\}^{i-j(n)}$. For each $t \in \{0,1\}^{j(n)+l(n)}$, M' forms $u = s \circ t$ and simulates M on input $i \circ h \circ f(x) \circ u$. We claim that with probability at least $p(n)/2$ there is a round, i.e. an i and a t, where this simulation yields a y with $f(y) = f(x)$. The round in question is when $i = i(x)$. We know that the probability that M, on input $i(x) \circ h \circ f(x) \circ (h(x) \uparrow (i(x) + l(n)))$, for x and h randomly selected, yields such a y, is at least $p(n)$. If instead of trying the one value $h(x) \uparrow (i(x) + l(n))$ we try $(h(x) \uparrow (i(x) - j(n))) \circ t$ for all possible extensions $t \in \{0,1\}^{j(n)+l(n)}$, the chance that M finds such a y can only increase. Fix $f(x)$ and consider the distribution $h \circ (h(x') \uparrow (i(x) - j(n)))$ for x' randomly and uniformly chosen from $f^{-1}(f(x))$. The distribution on x' has min-entropy $\log\left(|f^{-1}(f(x))|\right) \geq i(x)$. Hence, by the Leftover Hash Lemma, the distribution $h \circ (h(x') \uparrow (i(x) - j(n)))$ is statistically indistinguishable within $3p(n)^2$ from the distribution $h \circ s$ when $h \in_U H_{n,k(n)+l(n)}$ and $s \in_U \{0,1\}^{i(x)-j(n)}$. Thus the probability that, for at least one t, M on input $i(x) \circ h \circ f(x) \circ s \circ t$ finds a preimage of $f(x)$ for s chosen at random differs from the probability when s is given by $h(x) \uparrow (i(x) - j(n))$ by at most $3p(n)^2$. Thus, this probability for a random s is at least $p(n) - 3p(n)^2 \geq p(n)/2$. From this contradiction of the one-wayness of f, we conclude that g' is one-way on E. □

We now conclude the proof of Lemma 3.8.3. To distinguish D from $g(U')$ is equivalent to being able to predict $r \odot x$, for $i(x) \circ h \circ x$ randomly chosen according to E_n and and $r \in_U \{0,1\}^n$, given $r \circ g'(i(x) \circ h \circ x)$. From the above claim and from the Goldreich-Levin inner product bit Theorem, Theorem 3.4.5, it follows that this task is computationally infeasible, and thus $g(U')$ and D are computationally indistinguishable.

All that remains to be shown is that $Ent(D_n) > 1/3n + Ent(g(U'_n))$. This follows from the fact that with probability at least $1 - 1/n$, when $x \in_U \{0,1\}^n$ and $h \in_U H_{n,k(n)+l(n)}$, x is uniquely determined by $f(x)$ and $h(x) \uparrow (i(x)+l(n))$. When x is fixed and $i \in_U \{0,1\}^{l(n)}$, $i = i(x)$ with probability at least $1/2n$. Thus, with probability at least $(1-1/n)1/2n \geq 1/3n$ the last bit of $g(r \circ i \circ h \circ x)$ is completely determined by the preceding bits, whereas in $D(r \circ i \circ h \circ x \circ \beta)$ the last bit β is always independent of the preceding bits, and thus there is one extra bit of entropy in D_n. Furthermore, in all cases, $r \odot x$ adds at most one bit of entropy to the preceding bits of $g(r \circ i \circ h \circ x)$. $\square$

Theorem 3.4.3 If there is a one-way function in the non-uniform model of security then there is a pseudo-random generator in the non-uniform model of security.

Proof : Combine Lemma 3.8.6 and Corollary 3.8.5. $\square$

3.9 Conclusion

The result presented in this chapter gives a unified foundation for much of theoretical cryptography. When combined with other work ([H], [GGM], [LR], [GMW], [Na] [FS]), it shows that applications ranging from private key encryption to zero-knowledge proofs can be based on any one-way function. In Chapter 4, we present results from [IL2] showing that most cryptographic applications require a one-way function. Recently, [Rom] shows how to base a signature scheme on any one-way function. Thus, the existence of a one-way function is a necessary and sufficient assumption for much of modern cryptography. This will be discussed in more detail in the next chapter.

One important issue concerning cryptographic PRG's is that of efficiency; the construction we give here for a pseudo-random generator based on any one-way function increases the size of the seed by a large polynomial amount. For practical applications, it would be nice to have a much more parsimonious construction. Another general construction which could use a more efficient version is the [GGM] construction of a pseudo-random function-generator based on a one-way function. Besides efficient general constructions, it is interesting to look at especially efficient constructions of pseudo-random generators and pseudo-random function generators based on specific one-way functions. Of course, "efficiency" here would depend on how secure one assumes the original one-way function is. [IN] and [SSh] give highly efficient PRG based on the subset-sum and discrete log problems. However, we know of no "practical" way of constructing a pseudo-random function generator, needed for private key cryptography, based on any reasonable assumption. (What is desired here is an alternative to DES which is theoretically appealing.)

Another open question concerns cryptographically secure pseudo-random generators implementable quickly in parallel (i.e., NC). In particular, is this implied by the existence of a one-way function in NC? Suprisingly, the crux of this question is not converting such a function to a PRG whose output is slightly larger than its input (in fact, the construction presented here is implementable in parallel if the original one-way function is), but in converting such a PRG to one whose output is much longer than its input. The construction given for this in [GGM] seems inherrently sequential. In particular, no NC PRG based on factoring is known, although the [IN] PRG based on subset sum is computable in NC.

Chapter 4

One-way Functions are Necessary for Complexity-Based Cryptography

4.1 Introduction

In the previous chapter, we showed that, given a one-way function, it is possible to construct a pseudo-random generator; hence, the existence of a one-way function is a sufficient condition for much of modern cryptography. Here, we show that this is also a necessary condition for almost all cryptographic tasks.

In much of modern cryptography, the security of protocols that have been designed for a variety of different tasks is based on the intractability of some problem, for example, factoring randomly chosen large numbers. To date, the problems assumed to be intractable all have the same form; they are based on a one-way function, i.e. one that is easy to compute but hard to invert. An intuitive reason for this is that such a function represents a source of hard problems which can be generated together with a solution, thus breaking the symmetry of knowledge between the participants. By picking a random input, and computing a one-way function on the input, one participant distinguishes herself from the other participants and the adversary, since only she knows the inverse of the function on this output. ([AABFH] give a formal notion of a hard problem for which instances can be generated together with solutions; the existence of such a problem is equivalent to the existence of a one-way function.) However, this intuition does not seem to translate directly into a rigorous proof; our objective for this section is to develop a general technique for formally stating and proving statements of the kind, "Cryptographic task X requires a one-way function."

Also, although the previous discussion suggests that it is reasonable that some form of one-way function is necessary for cryptography, it is not *a priori* clear which exact definition of one-way function is the correct one. Many different definitions of one-way function have been proposed; sometimes, stronger assumptions about the form of the function, e.g., that the function be a permutation, are made; sometimes the notion of what successfully inverting the function means has been strengthened or weakened. Our results show that the definition given in Chapter 3, where there are no restrictions on the form of the function and inverting the function means finding any pre-image, is the "correct" one (or at least, equivalent to any correct one) in that the existence of such a one-way function is a necessary and sufficient condition for many tasks in cryptography, and is a necessary condition for almost all such tasks. In fact, the only commonly discussed tasks we could find which do not provably require one-way functions in the sense of this standard definition are those with information-theoretic (based on no complexity assumptions) solutions, such as secret-sharing ([Sham],[Bla]). As an informal conjecture, we feel that this will be true in general; no assumption weaker than the existence of one-way functions will be useful in cryptography.

Our results concern cryptography based solely on computation and transfers of information between participants. It is also possible to conjecture that the laws of physics will allow us to build devices with special features which can be used to perform cryptographic tasks without any complexity-theoretic assumptions. One example of this is Quantum Cryptography, which constructs a physical channel for information which is secure if the Heisenberg Uncertainty Principle is true (see, e.g., [BBR]). Another example of work along these lines is the Noisy Channel model of [CK], where oblivious transfer is shown to be possible if one is given an information channel which interupts a message with a fixed probability p. The impossibility results we prove here implicitly assume that the only physical devices available to the participants are insecure communication channels and private, unbiased coins.

To repeat, we will show that for most of the standard cryptographic tasks, any secure protocol for the task can be converted into a one-way function in the usual sense, and thus the security of any proposed protocol for these tasks is implicitly based on a function being "one-way". Thus, the usual definition of a one-way function is robust; any definition of one-way function from which a secure cryptographic protocol for any of these tasks can be based can be used to construct a one-way function in the usual sense. Thus, the standard notion of one-way function is in a formal sense the correct one.

However, to prove these claims, we introduce a (seemingly) weaker concept of one-way function, called a distributionally one-way function, which we feel is of some independent interest. Intuitively, a function is distributionally one-way if it is easy to compute, but, given an output, it is difficult to uniformly generate random pre-images. Although a distributionally one-way function seems much weaker than the standard kind of one-way function, we prove that the existence of such a function implies that of a one-way function in the standard sense. Thus, to show that the usual sort of one-way function is necessary for the task, it suffices to construct a function which is one-way in this weaker sense.

Because there are too many cryptographic tasks in existence to discuss in a single paper (or even a thesis!), we choose to cover a spectrum of the more central cryptographic applications: private-key encryption, identification/authentication, bit commitment and coin-flipping by telephone. These tasks were chosen for simplicity, importance to cryptography, and because they run the gamut of cryptographic applications. For most other standard cryptographic tasks, the analagous result can be obtained either by a direct reduction to one of the four theorems proved here, or by imitating the proofs of these theorems. As an example, we will show how secret agreement and electronic signatures can both be used for identification; hence, that these tasks require one-way functions follows immediately from the analagous statement about secure identification protocols. (This result for secret agreement was also obtained by [BCG], using the same techniques, as outlined in [IL1].)

Our results allow us, for any of these four tasks, to give a "normal form" for protocols for the task; i.e., for each task there is a universal format so that any given secure protocol for the task can be used to design a secure protocol in the universal format. For example, we show that any secure protocol for private-key encryption, (even one with many communication rounds that is probabilistic and has some chance of error in decryption) yields a pseudo-random generator. Combining this with [GGM] and [LR], we have a way of converting any private-key encryption scheme into a "pseudo-random block cipher" (a provably secure private-key cryptosystem of the same form as DES). This resolves an open problem stated in [Rac2]. (Interestingly, [Rud2] presents strong evidence, in the form of relativized results, that this type of result will not be provable for secret agreement.)

The results given here are an expanded version of [IL2]; some of the results presented here were announced in [IL1]. Many of the proofs rely on [ILL] and [H]. (See Chapter 3 and Appendix 2, respectively.)

4.2 Distributionally One-Way Functions

We will use what we call "distributionally one-way functions" (defined below) as a stepping stone in our constuctions of one-way functions from various cryptographic tasks. The notion of a distributionally one-way function was inspired by [Rud1] (see also [IR]) and was defined under the name "informationally one-way functions" in [IL1].

Before giving the formal definition, we will try to motivate the idea of "distributionally one-way" as a natural generalization of the concept of "one-way".

Consider a one-way function f which is not computed directly but instead via a probabilistic polynomial time algorithm A that can make mistakes a small fraction of the time, i.e. for most r, $A(x,r) = f(x)$. For example, to compute $f(x)$, A might use a *BPP* sub-routine; if the sub-routine makes errors, the value A outputs might be arbitrary. Let (x,r) be a random pair and let $y = f(x)$. Even though no adversary can find, given y, an x' such that $f(x') = y$, there might be an adversary that can find a pair (x', r') such that $A(x', r')$ is mistakenly equal to y (instead of to $f(x')$). Thus, the new function A might not be one-way in the standard sense. However, it is intuitively clear that A still has a kind of "one-wayness". More precisely, although it is possible to find an inverse of y with respect to A, the inverses found all have a special form, easily distinguishable from a random inverse. The following definition of distributionally one-way insists that an adversary must produce random preimages of $y = f(x)$ approximately uniformly. Thus, even though the A described above is not one-way, it is distributionally one-way.

Definition 4.2.1 *We say that f is* distributionally one-way *if, for some constant $c > 0$, for every feasible algorithm M, for $x \in_U \{0,1\}^n$, the distribution defined by $x \circ f(x)$ and the distribution defined by $M(f(x)) \circ f(x)$ are statistically distinguishable by at least n^{-c} for all but finitely many lengths n.*

If f is a distributionally one-way function, then it is computationally infeasible to randomly generate preimages of $f(x)$. With this definition, we fix a parameter c of distinguishability: an algorithm that randomly generates preimages of $f(x)$ in a manner statistically close but still n^{-c} distinguishable from the uniform distribution is considered to be unsuccessful. This is a weak form of a one-way function, but the following lemma shows that such a one-way function can be used to construct a one-way function in the usual sense.

Theorem 4.2.2 *If there is a distributionally one-way function, then there is a one-way function.*

Proof : Let f be a distributionally one-way function with associated constant c. Let $H_{n,m}$ be a family of hereditarily universal hash functions from n bit strings to m bit strings [Carter, Wegman]. Let $g(h \circ i \circ x)$ be $h \circ i \circ f(x) \circ h(x)_{\leftarrow i}$, where $i \in \{1, \cdots, n + (6c+6)\log n\}$ and $h \in H_{n,n+(6c+6)\log n}$. We claim g is a somewhat one-way function. By Proposition 3.3.4, this suffices to construct a one-way function.

Assume we had an algorithm A which inverted g on random inputs where x is of length n with probability $1 - 1/n^{4c+10}$. Then let A' be the following algorithm to generate almost uniformly random pre-images of f. Given input n and $y = f(x)$, we first wish to approximate the size of $f^{-1}(y)$ to within a factor of n^{2c+2}. We do this as follows: for each value of i, $1 \leq i \leq n + 2c\log n$, pick at random n hash functions h_j and n random strings of length i, s_j. For each $j, 1 \leq j \leq n$, simulate A on $h_j \circ i \circ y \circ s_j$. "Accept" i if, for some j, A finds an inverse of g, i.e., for $z_j = A(h_j \circ i \circ y \circ s_j)$, we have $f(z_j) = y$ and $h_j(z_j)_{\leftarrow i} = s_j$. Let i^* be the greatest value for i which is accepted. 2^{i^*} is our approximation. We claim that this approximation is within the prescribed bounds with probability at least $1 - 1/n^{2c}$; i.e., with this probability, $\log |f^{-1}(y)| \leq i^* \leq \log |f^{-1}(y)| + (2c+2)\log n$.

To see that the upper bound holds with high probability, let $i > \log |f^{-1}(y)| + (2c+2)\log n$. Then, for any fixed j and h_j, there are at most $|f^{-1}(y)|$ strings of the form $h_j(z)_{\leftarrow i}$ for $z \in f^{-1}(y)$, as compared to $n^{2c+2}|f^{-1}(y)|$ strings of the same length. Thus, when we select s_j at random, the probability that it is in this set is $1/n^{2c+2}$. Thus, since no inverse of g can be found by A if none exists, for each $j, 1 \leq j \leq n$, the probability that a pre-image is found is at most $1/n^{2c+2}$ for a total probability of at most $1/n^{2c+1}$. Since there are at most n different possibilities for i as above, this gives a bound of at most a $1/n^{2c}$ probability of any such i being accepted.

For the other direction, we show that $i = \log |f^{-1}(y)| + 1$ is accepted with high probability. A fails, for a random image of g, with probability at most $1/n^{4c+10}$. Therefore, with probability at least $1 - 1/n^{2c+5}$, for $y = f(x)$ for a randomly chosen x, the conditional probability that A succeeds on $g(h, i, x')$ is at least $1 - 1/n^{2c+5}$, where h and i are random, and x' is a random element of $f^{-1}(y)$. In this case, there can be no i so that the conditional probability of failure given y and i exceeds $1/(n^{2c+4})$, since each i is chosen with probability $1/n$. In particular, this will be true for the value of i above. Now, for a fixed $x', x'' \in f^{-1}(y)$, and a randomly chosen h, the probability that $h(x')_{\leftarrow i} = h(x'')_{\leftarrow i}$ is exactly $1/2^i$. Thus, for a fixed x', the probability that there is such an x'' is at most $|f^{-1}(y)|/2^i = 1/2$. Therefore, the conditional probability that A is successful given that no such x'' exists must be at least $1 - 2/n^{2c+4} > 1/2$. In other words, we expect to have at least $1/4|f^{-1}(y)| = 1/8(2^i)$ strings $x' \in f^{-1}(y)$ where there are no such x'''s, and where A is successful on $g(h, i, x') = h \circ i \circ y \circ h(x')_{leftarrowi}$. Since each such x' uniquely determines the string $h(x')_{leftarrowi}$, we expect A to be successful on at least $1/8$ of the inputs of the form $h \circ i \circ y \circ s$, for s an i bit string. Thus, each time we pick a random h_j and s_j, we have at least a $1/8$ probability of A's being successful; doing this n times independently at random, we have an exponentially small chance of not accepting i.

Now that we know roughly how large $f^{-1}(y)$ is, we want to pick an element of this set (almost) uniformly at random. Let $i' = i^* + (4c+4)\log n$; i' is an overestimate of the number of bits we need to hash an element of $f^{-1}(y)$ to in order to, with high probability, uniquely define the element. We pick a random hash function h; we then pick random strings s_j of length i' until A successfully inverts g when given $h \circ i \circ y \circ s_j$. When this happens, we output the value of x given by A; if this fails to happen after n^{8c+8} attempts, we halt and output some default value. We need to show that this produces, with high probability, an almost uniformly distributed element of $f^{-1}(y)$. As before, we have that with probability $1 - 1/n^{2c+4}$ over the choice of y and h, the conditional probability that A succeeds in inverting $g(h,i,x')$ for $x' \in f^{-1}(y)$ is at least $1-1/n^{2c+5}$. Also, if $i^* \geq \log|f^{-1}(y)|$, as we have argued happens with high probability, then $i' \geq \log|f^{-1}(y)| + (4c+4)\log n$, and for any $x \in f^{-1}(y)$, the probability over a random h that $h(x)_{\leftarrow i'}$ uniquely determines x' in the set $f^{-1}(y)$ (i.e., there is no $x' in f^{-1}(y)$ with $h(x)_{\leftarrow i'} = h(x')_{\leftarrow i'}$) is at least $1-1/n^{4c+4}$. Thus, for a random h, with probability at least $1-1/n^{2c+1}$, the fraction of uniquely defined x' is at least $1-1/n^{2c+3}$. Assume that both of these likely events occur. Let S be the set of $x \in f^{-1}(y)$ so that $h(x)_{\leftarrow i'}$ uniquely determines x, and so that A succeeds on $g(h,i',x) = h \circ i' \circ y \circ h(x)_{\leftarrow i'}$, and let $T = h(S)$. Then $|T| = |S| \geq (1-2/n^{2c+1})|f^{-1}(y)|$. Therefore, the uniform distribution on S is indistinguishable from the uniform distribution on $f^{-1}(y)$ within $2/n^{2c+1}$. Note that the conditional distribution on our outputs given that some element of S is output is the uniform distribution on S. Thus, it remains to show that with high probability, our algorithm outputs an element of S. Let $S' = f^{-1}(y) - S$, and $T' = h(S')$. Then $|T'| \leq |S'| \leq |f^{-1}(y)|2/n^{2c+1}i \leq |T|3/n^{2c+1}$. Since A can only invert g on elements which have inverses, which in this case means $s_j \in T \cup T'$, this implies that the conditional probability that we output an element of S, given that we output something, is at least $1-3/n^{2c+1}$. To see that we output something with high probability, note that, in each round, we have a $|T|/2^{i'}$ chance of outputting an element of S. If $i \leq \log|f^{-1}(y)| + (2c+2)\log n$, as happens with high probability, $i^* \leq \log|f^{-1}(y)| + (6c+6)\log n$, so this probability is at least $1/2n^{6c+6}$. Since we pick n^{6c+8} strings s_j, one of these strings will be in T with probability exponentially close to 1. □

In our results, we prove that a secure protocol yields one of the following: (1) a distributionally one-way function; (2) a one-way function; (3) a false entropy generator; or (4) a pseudo-random generator. By Theorem 4.2.2, and the results of Chapter 3 and [H] (Appendix 2), all of these are equivalent.

4.3 Protocols

In order to define the notion of a secure protocol for the various tasks we wish to consider, we must first develop some formalistic notations for describing protocols. As all the tasks considered here involve two participants, we will not define a multi-participant protocol. The participants will in general have three kinds of inputs: a "security parameter" n governing the complexity of the protocol in terms of the number of communication rounds allowed, bits of communication per round, and time taken to execute the protocol; "random tapes", which allow for probabilistic moves on the part of the participants; and "real inputs", which are an inherrent part of the task to be completed (e.g., a message to be sent or "signed" electronically). Once these inputs have been provided, the participants should act deterministically; for our purposes, the only allowable action is to send a message to the other participant. Without loss of generality, we assume the participants alternate sending messages, and that the total number of messages sent depends only on the security parameter. We often wish to say that at the end of a protocol, one or the other participant "knows" a message or fact; we model this by having the participants have a private output, depending on their private inputs and the conversation, where they can "sum up" the gist of the protocol. Often, this output is only a single bit meaning "Accept" or "Reject". All complexity measures, including the number of message rounds, length of messages sent, number of random bits, and computation time, should be bounded by a polynomial in the security parameter.

Definition 4.3.1 *A* two participant protocol *consists of a* message length *function, $m(n)$, a* number of rounds *function, $r(n)$, a* length function for the first participant's random tape, *a(n), a* length function for the second participant's random tape, *$b(n)$, and two deterministic Turing machines, A and B, known as the* participants, *"Alice" and "Bob". m and r should be bounded by a polynomial in n, and, on an input of the form $(n, z_1, .., z_k)$, A and B should both output a string of length no more than $m(n)$. If, in addition, A and B are polynomial time machines, and $a(n)$ and $b(n)$ are bounded by a polynomial in n, we call the protocol* polynomial-time. *An input n, x, R_A for Alice and n, y, R_B for Bob, where $|R_A| = a(n)$ and $|R_B| = b(n)$, determine a* run *of the protocol with* security parameter *n, consisting of a* conversation *and two* private outputs, *one for each participant, which are defined as follows. The conversation between Alice and Bob is defined by $C_{AB}(n, x, R_A, y, R_B) = m_1, m_2, \ldots, m_{r(n)}$, where $m_i = A(n, x, R_A, m_1, \ldots, m_{i-1})$ if i is even, and $m_i = B(n, y, R_B, m_1, \ldots, m_{i-1})$ if i is odd. Alice has private output $Alice_{AB}(n, x, R_A, y, R_B) = A(n, x, R_A, C_{AB}(n, x, R_A, y, R_B)$, and similarly Bob has a private output denoted $Bob_{AB}(n, x, R_A, y, R_B)$. In the above situation, we call n the* security parameter, *x* Alice's private input, *R_A* Alice's random tape, *y,* Bob's private input, *and R_B,* Bob's random tape. *When the other parameters are unambiguous, we will denote to $C_{A,B}(n, x, R_A, y, R_B)$ as simply $C(R_A, R_B)$ or even C; and we will denote by $C_i(R_A, R_B)$, or simply C_i, the first i messages of the conversation, $m_1, ..m_i$ defined as above. In some protocols, one or both of the participants may have no private input, or no random tape; in this case, we suppress mention of these non-existent inputs.*

A well known fact about protocols is:

Proposition 4.3.2 *If, for a particular protocol, security parameter, and inputs, we have $C(R_A, R_B) = C(R'_A, R'_B)$, then $C(R_A, R_B) = C_(R_A, R'_B) = C(R'_A, R_B) = C(R'_A, R'_B)$. In other words, the set of pairs R_A, R_B that yield the same conversation is $(x, y) \in X \times Y$, where X is a set of random tapes for Alice and Y is a set of random tapes for Bob.*

4.4 Identification

Intuitively, an identification protocol allows Alice to convince Bob that she is the same person he was talking to at an earlier time, as opposed to being an eavesdropper who overheard their earlier conversation.

Definition 4.4.1 *An* identification protocol *actually consists of two polynomial-time protocols: P^1, called the* introduction *protocol, and P^2, called the* recognition *protocol. In the introduction protocol, Alice (A^1) has random input R_A and Bob (B^1) has random input R_B, which produce conversation $C_1(R_A, R_B)$. (Here, we are suppressing mention of the security parameter; there are no private inputs in the introduction protocol.) In the recognition protocol, the private input for Alice (A^2) is R_A and $C_1(R_A, R_B)$. Similarly, the input for Bob (B^2) is R_B and $C_1(R_A, R_B)$. At the conclusion of the recognition protocol, Bob either outputs "ACCEPT" or "REJECT".*

(Without loss of generality, we can assume that the second protocol requires no additional random coin flips, R_A and R_B providing the randomness for the second protocol as well as the first. This will simplify notation in the following discussion.)

A secure identification protocol *has the following two properties:*

- *For every security parameter n, Bob accepts Alice in the recognition protocol with a high probability, i.e.,*

  $$Prob[Bob_{A^2B^2}(n,(R_A,C_1(R_A,R_B)),(R_B,C_1(R_A,R_B))) = \text{``}ACCEPT''] \geq .9$$

  (over random tapes R_A and R_B).

- *For any feasible algorithm E, and for all but finitely many security parameters n,*

  $$Prob[Bob_{EB^2}(n,(R_E,C_1(R_A,R_B)),(R_B,C_1(R_A,R_B))) = \text{``}ACCEPT''] \leq .1$$

  i.e., Bob rejects any feasible adversary (called the "eavesdropper"), that has access to the conversation from the introduction protocol but does not have the value of R_A that Alice has. (The random input R_E to E is allowed to be of any feasible length, so R_E may be much longer than R_A).

  Intuitively, any listener who heard the introduction conversation between Alice and Bob and who tries to impersonate Alice in the recognition protocol is caught with high probability by Bob.

Theorem 4.4.2 *Any secure identification protocol can be used to construct a distributionally one-way function.*

Proof : Let $C_1 = C_1(R_A, R_B)$ abbreviate $C_{A^1B^1}(n, R_A, R_B)$. We prove, by contradiction, that if the identification protocol is secure, then C_1 is a distributionally one-way function. By Proposition 4.3.2, for a fixed R_A and R_B, the set of pairs R'_A, R'_B with $C_1(R_A, R_B) = C_1(R'_A, R'_B)$ is of the form $X_{C_1} \times Y_{C_1}$. Assume that C_1 is not distributionally one-way. Let $M(n, C_1, r)$ be the feasible algorithm that, on input $C_1(R_A \circ R_B) = C_1$ and a randomly chosen r, produces a pair (R'_A, R'_B) such that the distribution on triples R'_A, R'_B, C_1 is statistically indistinguishable within $1/n^2$ of the distribution R_A, R_B, C_1. Then, with probability $1 - 1/n$ over R_A, R_B, the distribution given by $M(n, C_1, r)$ (for r randomly chosen) is indistinguishable within $1/n$ from the uniform distribution on $X_{C_1} \times Y_{C_1}$. The eavesdropper E, who knows C_1 (but not R_A), will simulate M on C_1 to obtain R'_A, R'_B, and will then use R'_A to simulate Alice in the recognition protocol, i.e., for each message to be sent by E in the recognition protocol with B^2, E simulates A^2 on input (n, R'_A, C_1) (and the previous messages in the recognition protocol). We claim that B^2 accepts in the recognition protocol between B^2 and E with almost the same probability as in the recognition protocol between B^2 and A^2. The distribution R'_A, R'_B, C_1 is indistinguishable within $1/n^2$ of the distribution R_A, R_B, C_1. In this latter distribution, given C_1, R_A and R_B are uniformly distributed on the sets X_{C_1} and Y_{C_1}, respectively. Thus, with probability $1-1/n$, fixing C_1, the distribution on pairs R'_A, R'_B must be indistinguishable within $1/n$ of the uniform distribution on $X_{C_1} \times Y_{C_1}$; in particular, R'_A will be indistinguishable, within $1/n$, from a random element of X_{C_1}. Call such a C_1, "good". Since, once a good C_1 is fixed, R'_A is independent of R_B, this means that the distribution on R'_A, R_B is also indistinguishable from the uniform distribution on $X_{C_1} \times Y_{C_1}$, and hence from R_A, R_B. Thus, the probability, given any good C_1, that B^2 (using random tape R_B) accepts in the recognition protocol with A^2 (using random tape R_A) is within $1/n$ of the probability that B^2 accepts in the protocol with E (who is simulating A^2 using random tape R'_A). Thus, the total difference in the probabilities of acceptance, is at most $2/n$, adding the $1/n$ probability that C_1 is not good to the $1/n$ bound on the difference given C_1 is good. This is a contradiction, since a difference in probabilities of acceptance of at least .8 is required for a secure identification protocol. □

Identification, as defined here, is a "least common denominator" for many types of cryptographic tasks. Secure protocols for many other cryptographic tasks can be used to construct a secure identification protocol. One such example is a secret agreement protocol between two parties, where the parties involved agree on a common secret (with reasonable probability) which an eavesdropper cannot feasibly deduce from the conversation. An identification scheme based on secret agreement can be designed as follows. In the introduction protocol, Alice and Bob agree on a common secret. To recognize Alice, Bob asks Alice to send her this secret. This special case, that a secret agreement protocol implies a one-way function, was independently proved in [BCG] (using Theorem 4.5 of [IL1], which is Theorem 4.2.2 in this paper).

4.5 Bit Commitment

Intuitively, a bit commitment protocol allows Alice to "commit" herself to a secret bit without allowing her to change her mind later in the protocol. Alice does not want Bob to know the value of her bit until a later time, and thus Alice sends encrypted information to Bob about the bit from which Bob is unable to determine its value with probability significantly greater than 1/2. On the other hand, when Alice finally does release the value of her bit by releasing information about the encryption, Bob would like to be sure that Alice can't change her mind about the original value to which she committed herself.

Definition 4.5.1 *A* bit commitment protocol *consists of two polynomial-time protocols, P^1, called the* commit *protocol, and P^2, called the* release *protocol. In the commit protocol, Alice (A^1) has random input R_A and a one-bit "real" input b, and Bob (B^1) has a random input R_B. As before, to simplify notation, we will assume that R_A and R_B also contain the random bits for the release protocol. Also as before, we will abbreviate $C_{A^1B^1}((n, R_A, b), (n, R_B))$ as C_1 or $C_1(R_A, b, R_B)$. In the release protocol, Alice (A^2) has input R_A, b, and C_1, and Bob (B^2) has input R_B and C_1. At the end of the release protocol, Bob either outputs "REJECT" or a bit b'. If Alice and Bob obey the protocol, Bob should accept and output $b' = b$ with high probability in the release protocol. Informally, the protocol is secure if: no feasible cheating strategy for Bob can allow him to guess the value of b, before the release protocol has begun, with significantly more than a 1/2 probability; and, every feasible cheating strategy for Alice has, with high probability after the commit protocol is over, only one choice of bit b' she can make Bob output. Formally, a bit commitment protocol is secure if*

1. $$Bob_{A^2B^2}((n, R_A, b, C_1), (n, R_B, C_1)) = b$$
   with probability at least .99.

2. *Let B' be any feasible algorithm (for the commitment protocol). Then, $Prob[Bob_{A^1B'}((n, R_A, b), (n, R_{B'})) = b] - 1/2$ is negligible as a function in n.*

3. *Let D^1 and D^2 be any feasible algorithms. (D^1 is a cheating strategy for Alice in the commit protocol, and D^2, in the release protocol.) Let D^1 have random input R_D , and D^2 have inputs a bit b', R_D, and $C_1 = C_{D^1B^1}((n, R_D), (n, R_B))$. Then the probability that both $Bob_{D^2B^2}((n, 0, R_D, C_1), (n, R_B, C_1)) = 0$ and $Bob_{D^2B^2}((n, 1, R_D, C_1), (n, R_B, C_1)) = 1$ is negligible. In other words, no cheating strategy for Alice can arrange to be able to be able to reveal both a 0 and a 1.*

The above definition of bit commitment includes as special cases bit commitment versus a strong receiver [BC], [Cha] and versus a strong committer [GMW]. If we remove the word "feasible" in the second clause above, we call the protocol "secure vs. a strong receiver", and if we remove the word "feasible" from the third clause, we call the protocol "secure vs a strong committer."

Combining the results here with Chapter 3, [H] and [Na], we get that the existence of any bit commitment protocol implies a bit commitment protocol versus a strong committer. [GMR] defines the notion of a zero-knowledge proof system, and [GMW] shows how any bit commitment protocol that is secure versus a strong committer can be used to construct a zero-knowledge proof system for any problem in *NP*. Putting this all together, any bit commitment scheme can be used to construct a zero-knowledge proof system for any problem in *NP*. The analogous statement for a strong receiver, which would imply that any bit commitment could be used for minimum disclosure proofs ([BC], [Cha]), is an open question.

Theorem 4.5.2 *Any secure bit commitment protocol can be used to construct a distributionally one-way function.*

Proof : One proof would be to show that any bit-commitment scheme can be used for identification. In the introduction protocol, Alice commits to several randomly chosen bits. In the recognition protocol, she releases these bits. This can be made formal, but we will instead give a direct proof.

Our choice for a distributionally one-way function will again be $C_{A^1B^1}((n, b, R_A), (n, R_B))$ which we will abbreviate as $C_1(b, R_A, R_B)$. Again, we will prove that C_1 is distributionally one-way by contradiction. Let $M(n, C_1, r)$ be a feasible, probabilistic algorithm producing triples b', R'_A, R'_B so that the distributions b', R'_A, R'_B, C_1 and b, R_A, R_B, C_1 are indistinguishable. As before, for a fixed value C for C_1, let X_C and Y_C be such that $X_C \times Y_C$ is the set of pairs $(b', R'_A), R'_B$ with $C_1((b', R'_A), R'_B) = C$. Call C *balanced* if, for a random $(b', R'_A) \in X_C$, $1/4 < Prob[b' = 0] < 3/4$, and *unbalanced* otherwise. If, with a significant probability, C_1 is unbalanced, then Bob can use M to give a good guess as to the value of the committed bit b, Otherwise, if C_1 is balanced, Alice can commit to a 0, and later use M to find a random $(1, R'_A) \in X_{C_1}$; pretending that this was the random tape she used in the commit stage, she can reveal a "1" to Bob.

More formally, we have as before that, with probability $1 - 1/n$ over the value C of $C_1((b, R_A), R_B)$, (b', R'_A) is statistically indistinguishable within $1/n$ from the uniform distribution on X_C. Assume C is unbalanced with probability at least $1/2$. Then the following method allows Bob to predict b with a greater than $1/2$ probability. B' follows the protocol exactly during the commit stage, then simulates M on $C = C_1((b, R_A), R_B)$ to obtain b', R'_A, R'_B. B' guesses b'. We claim $b' = b$ with probability at least $3/4$. Call C "good" if, given $C_1((b, R_A), R_B) = C$, the distribution on (b', R'_A) is indistinguishable within $1/n$ from the uniform distribution on X_C. All but a $1/n$ fraction of $C's$ are good. For a fixed C, let $p = Prob[b = 0]$, and $p' = Prob[b' = 0]$. If C is good, $|p - p'| \leq 1/n$, and, since, for a fixed C, b and b' are independent, the probability that $b = b'$ is $pp' + (1 - p)(1 - p') \geq p^2 + (1 - p)^2 - 1/n$. Thus, this probability is always at least $1/2 - 1/n$, and, if C is unbalanced, is at least $5/8 - 1/n$. Thus, the overall probability that $b = b'$ is at least

$(1/2-1/n)(1/2-1/n)+1/2(5/8-1/n)>13/16-2/n>3/4$ (the worst case being when all "bad" C's are unbalanced.)

Otherwise, C is unbalanced with probability at least 1/2. Consider the following strategy D for Alice. In the commit protocol, D commits to a randomly chosen b, obeying the protocol (i.e., simulating A_1). In the reveal protocol, D is given a bit b_0 to reveal. If $b_0=b$, D follows the reveal protocol. Otherwise, D runs M n times on input $C=C_1((b,R_A),R_B))$, and lets R''_A be R'_A for the first output of M of the form b', R'_A, R'_B with $b'=b_0$, if any such exists. If no such triples exist, D quits. D^2 simulates A^2 on input b_0, R''_A, C in the reveal protocol. Call C *excellent* if C is good, balanced, and if, for the real reveal protocol, B^2 outputs b with probability at least .9 given that $C=C_1((b,R_A),R_B)$. Since Bob outputs b in the reveal protocol with probability at least .99, this last condition holds with probability at least .9. Since C is bad with at most probability $1/n$ and unbalanced with probability at most 1/2, C is excellent with probability at least $.4-1/n>1/3$. If C is excellent, the distribution on (b', R'_A) is within $1/n$ of the uniform distribution on X_C, so $|p-p'|<1/n$. Since C is balanced, both p and $1-p$ are at least 1/4, so p' and $1-p'$ are at least $1/4-1/n$; thus, D only quits with an exponentially small probability. Also, let $X_{b_0}=\{R_A|(b_0,R_A)\in X_C\}$. Then R''_A is indistinguishable within $24/n$ from the uniform distribution on X_{b_0}. For, let $S\subset X_{b_0}$, and let $s=|S|/|X_{b_0}|$, and, without loss of generality assume $b_0=0$. Then $Prob[R''_A\in S]=Prob[R'_A\in S]/p'\leq (|S|/|X|+1/n)/(p-1/n)\leq (sp+1/n)/(p-1/n)\leq s+24/n$, the last inequality holding because $p\geq 1/4$. Also, the probability that B^2 outputs b_0 after the reveal protocol when A^2 uses input b_0, R_A, C, for R_A randomly chosen from X_{b_0} is at least .6, since for a random element $(b,R_A)\in X_C$, $b=b_0, R_A\in X_{b_0}$ with probability at least 1/4, and the overall probability that Bob does not output b is at most .1. Combining the two, and since R_B is independent of R''_A, we have that Bob outputs b_0 when A^2 is run on input b_0, R''_A, C with probability at least $.6-24/n>1/2$. Since there is at least a 1/3 chance of $C's$ being excellent, and since there is a 1/2 chance of *Bob* outputting the opposite of the bit committed to in the reveal protocol if C is excellent, there is at least a 1/6 chance of D^2 being able to convince Bob to output the reverse of the bit committed to. Since there is only a .01 chance of Bob's not outputting the bit committed to, this means that D has a $1/6-.01$ probability of being able to succeed for either desired bit. □

4.6 Private Key Encryption

Private key encryption is a system by which two people who have previously met and privately selected a random n bit key k, can send messages of length longer than n securely over an insecure channel. In other words, a computationally limited listener who does not know k should not be able to deduce any information about which message (of length at least $n+1$) was sent from overhearing the conversation on the insecure channel. This notion of an interactive probabilistic private key encryption system was suggested in [Rac2]. More formally:

Definition 4.6.1 *Let $l(n) > n$ be polynomially bounded in n. The idea is that Alice and Bob share a secret key k of length n, and Alice wants to send to Bob a message m of length $l(n)$ securely, where m is randomly chosen. i.e. so that no feasible listener L (who does not know k) can determine anything about m. The key is also selected according to some protocol, but, as the conversation during this protocol is private and between trusted parties, we may as well assume that one party generates the key (according to some polynomially samplable distribution) and tells the other participant the key. Formally, a* private key encryption protocol *consists of a* key selector *which is a polynomially sampleable probability ensemble, K, and a* message sending *protocol, a polynomial protocol A,B, where A (Alice) has as inputs a* key $k, |k| = n$, *and a* message $m, |m| = l(n)$, *as well as random tape R_A, and B (Bob) has as inputs k and a random tape R_B. A* random run *of such a protocol consists of choosing a key k according to K_n, and choosing a random message m and random R_A,R_B, and running the message sending protocol using k, m, R_A, R_B. At the end of the protocol, B outputs an $l(n)$ bit string m'; with high probability, this should be the message m. On the other hand, no eavesdropper, who knows the conversation generated by the protocol, but not the key used, should be able to guess the message, or any partial information about the message. More formally, a* secure private key encryption protocol *satisfies the following properties:*

- *With probability at least .99, Bob outputs the message m at the end of the protocol, i.e.,*

$$Prob[Bob_{AB}((n, k, m, R_A), (k, R_B)) = m] \geq .99.$$

  for a random run of the protocol.

- *No feasible algorithm L can learn anything about m from the conversation between A and B, i.e. the ensemble defined by $C_{AB}((n, k, m, R_A), (k, R_B)) \circ m$ (for a random run of the protocol with security parameter n) is computationally indistinguishable from the distribution defined by $C_{AB}((n, k, m, R_A), (n, k, R_B)) \circ x$, where k, R_A, R_B and m are chosen as in a random run of the protocol, and x is an independently and uniformly chosen $l(n)$ bit string.*

Theorem 4.6.2 *Any secure weak private key encryption protocol can be used to construct a pseudo-random generator.*

Proof :
We will use the result from Chapter 3 that any function with false entropy at least $1/n^c$ can be used to construct a pseudo-random generator. Let $K(n)$ be produced as $S(n, r)$, where S is a deterministic polynomial time computable function, and r is a randomly chosen string (of length some polynomial in n).

Then let $f(n, R_A \circ R_B \circ r \circ m) = C_{AB}((n, S(n,r), m, R_A), (n, S(n,r), R_B)) \circ m$. By the definition of secure private key encryption protocol, this distribution is computationally indistinguishable from that where we replace m by an independently chosen $l(n)$ bit string x. Thus, if we can show that the entropy of the latter distribution is significantly greater than that of the first. we are done. (Since both distributions are polynomially sampleable, this holds in both uniform and non-uniform models of security.) The intuition is that, for Bob to learn the string m with high probability, m has to be determined by the key $k = S(n,r)$ and the conversation. Thus, the conditional entropy of m given the conversation is at most the entropy of the key, which is at most n. On the other hand, x is completely undetermined by the conversation, so its conditional entropy will be $|x| = l(n) \geq n+1$. The analysis is slightly complicated by Bob's only needing to get the message 99 % of the time, rather than always.

Formally, let $k = S(n,r)$ be a random variable. For convenience, we drop mention of the security parameter n. We claim that the entropy of the distribution $C_{AB}((k, m, R_A), (k, R_B)) \circ m$ is at least .09 less than that of the distribution $C_{AB}((k, m, R_A), (k, R_B)) \circ x$. Thus, f has false entropy. The theorem follows from Corollary 3.8.5.

First, we show that the $l(n)$ bit string m is determined with probability at least .99 by the conversation and the n bit string k, i.e. there is an algorithm (not necessarily feasible) that given the conversation and k produces m with probability at least .99. The algorithm randomly generates a R'_B so that Bob's behavior on input (k, R'_B) is consistent with the conversation. Note that, for a fixed conversation and key k, the distribution on (m, R'_B) is the same as the distribution on (m, R_B), where R_B is the original random tape for Bob. Therefore, the probability that the algorithm outputs m is the same as the corresponding probability for B, .99. There is some random tape for the algorithm that achieves this probability. Fixing this random tape, we have a deterministic algorithm that, given $C_{AB}((k, m, R_A), (k, R_B))$, and k, outputs m with probability .99.

We now need to define and prove a well-known fact about conditional entropies.

Definition 4.6.3 *Let $X \in S$ and $Y \in T$ be random variables determined by t, which is distrubuted according to D. For $x \in S$, let $Y|X = x$ denote the distribution on Y given that X is x, i.e., $Y(t)$ for t chosen according to the distribution $D|\{t, X(t) = x\}$. Then the* conditional entropy *of Y given X for the distribution D, denoted $Ent_D[Y|X]$, or just $Ent[Y|X]$ when D is unambiguous, is the expectation, for t chosen according to D, of $Ent[Y|X = X(t)]$.*

Proposition 4.6.4 *Let $X \in S$ and $Y \in T$ be random variables in t, which is distributed according to D. Then $Ent[< X, Y >] = Ent[X] + Ent[Y|X]$.*

Proof: $Ent[< X, Y >]$ is the expectation, for t chosen according to D, of $-\log(Prob[X = X(t) \text{ and } Y = Y(t)]) = -\log(Prob[X = X(t)]Prob[Y =$

$Y(t)|X = X(t)]) = -\log Prob[X = X(t)] - \log Prob[Y = Y(t)|X = X(t)]$. Since the sum of expectations is the expectation of the sum, $Ent[< X, Y >] = Exp_D[-\log Prob[X = X(t)]] + Exp_D[-\log Prob[Y = Y(t)|X = X(t)]] = Ent[X] + Ent[Y|X]$. □

Letting C be the random variable $C_{AB}((k, m, R_A), (k, R_B))$, we wish to prove to prove that the entropy of the distribution $< C, m >$ is less than that of $< C, x >$. From the proposition, it suffices to prove the same difference for $Ent[m|C]$ and $Ent[x|C]$. Since x is a randomly and independently chosen $l(n)$ bit string, $Ent[x|C] = l(n)$. To calculate $Ent[m|C]$, fix a value C_0 for the conversation C. Since the algorithm described above, on input C, k, outputs m with total probability at least .99, with probability at least .9, the conditional probability of this event is at least .9. Call such a C_0 "good". If C_0 is not good, we still have an upper bound of $l(n) = |m|$ on the entropy of the distribution $m|C = C_0$. On the other hand, if C_0 is good, divide the distribution on m up into the conditional distribution given that the algorithm succeeds, and that given that the algorithm fails. Call the event that the algorithm succeeds S. Applying the proposition, we have: $Ent[m|C_0] \leq Ent[< m, S > |C = C_0] = Ent[S|C = C_0] + Ent[m|C = C_0 \textit{ and } S] = -Prob[S = 1|C = C_0] \log Prob([S = 0|C = C_0]) - Prob[S = 0|C = C_0] \log(Prob[S = 0|C = C_0]) + Prob[S = 1|C = C_0]Ent[m|C = C_0, S = 1] + Prob[S = 0|C = C_0]Ent[m|C = C_0, S = 0]$. Now, $Prob[S = 1|C = C_0] \leq .9$ since C_0 is good, and $Ent[m|C = C_0, S = 1]$ is at most $n = |k|$, since, given that the algorithm succeeds, m is completely determined by C and k. Also, $Ent[m|C = C_0, S = 0]$ is at most $|m| = l(n)$. Thus, the previous bound for $Ent[n|C_0]$ is itself bounded by $-.9 \log(.9) - .1 \log(.1) + .9n + .1l(n) \leq .8 + .9(l(n) - 1) + .1l(n) = l(n) - .1$. Thus, since C_0 is good with probability at least .9, the overall expectation of $Ent[m|C = C_0]$ is at most $.9(l(n) - .1) + .1(l(n)) = l(n) - .09$. □

4.7 Coin Flipping on the Phone

Coin flipping by telephone ([B2]) is a method for two participants who do not trust each other to choose a random bit. Alice and Bob are flipping a coin to decide an issue; Alice wants the coin to be heads, Bob, tails. Since Alice and Bob are only communicating with electronic messages, and cannot see each other's coins being flipped, they do not trust each other. To be secure, the protocol must be such that, if Bob follows the protocol, no feasible strategy for Alice can cause "Heads" to be decided on more than 50 % of the time; similarly, if Alice follows protocol, she will win at least 50 % of the time. (Actually, 50 %-epsilon, for a negligible $\epsilon(n)$.)

The fact that coin flipping by telephone requires a one-way function is slightly suprising, in that coin flipping does not seem, *a priori*, to involve any restrictions on the information flow between the parties. However, as a closer analysis will show, any protocol for coin-flipping must have rounds where one party has more information about the coin to be flipped than the other, but cannot use this information to influence the outcome. This fact is what we will use to construct our one-way function.

A realistic definition of coin flipping by telephone would also have to include a procedure for enforcing the decision, and to decide what happens if one party refuses to continue in the protocol (see [B2] and [Cle]). We make the unrealistic assumption that an impartial judge (i.e., some polynomial time computable predicate) can hear the entire protocol, and makes the final decision. However, as before, this unrealistic assumption only makes our result stronger, since any protocol for a "realistic" definition could be used for coin-flipping in our unrealistic model.

Definition 4.7.1 *A* coin flipping protocol *is a polynomial protocol, together with a deterministic poly-time computable predicate P (which determines the winner). There are no real inputs to the protocol; A has a random input R_A and B a random input R_B. In the following, we will omit mention of the security parameter, letting $C_{AB}(R_A, R_B)$ or simply C abbreviate $C_{AB}((n, R_A), (n, R_B))$. If $P(C_{AB}(R_A, R_B)) = 1$, we say the output of the protocol is heads, 0, tails. The protocol is* secure *if, for any polynomial p and any feasible strategy for Alice, A', $Prob[P(C_{A'B}(R'_A, R_B)) = 1] < 1/2 + 1/p(n)$ for all but finitely many n, and, likewise, for any feasible strategy for Bob, B', $Prob[P(C_{AB'}(R_A, R'_B)) = 1) > 1/2 - 1/p(n)$ all but finitely often.*

Theorem 4.7.2 *If there is a secure coin-flipping protocol, then there is a distributionally one-way function.*

Proof: The distributionally one-way function is constructed as follows: g takes as input n, i, R_A and R_B, where R_A and R_B are as in the protocol, and $i \in \{1, .., r(n)\}$. g simulates the protocol to obtain $C = C_{AB}(R_A, R_B)$ and outputs C_i, the first i messages sent during the protocol.

Assume g is not distributionally one-way. Then it is feasible to generate pre-images of g almost uniformly at random. In particular, one can use this to, given R_A and C_i, estimate closely the conditional probability that $P(C) = 1$. To do this, generate random (or almost random) pairs R'_A, R'_B which are preimages of C_i. Note that R'_B is distributed as a random tape for Bob given R_A and C_i (because the pre-images of C_i are a cross-product of possible R_A's and possible R_B's). Then simply simulate the protocol for the remaining steps using R_A and R'_B, and see whether the resulting conversation yields heads or tails. Repeat for a sufficiently large polynomial number of times; the fraction of heads will be close (with high probability) to the conditional probability. Similarly , it is possible to estimate the conditional probability of heads given C_i and R_B.

Without loss of generality, assume Alice sends the odd messages, Bob the even. Let A' be the following cheating strategy for Alice: A' picks a random tape R_1 and a random number of messages i. A' simualtes A using random tape R_1 for the first i messages. Then A' estimates the conditional probability of heads given R_1 and C_i. Next, A' generates (almost) at random a pre-image R'_A, R'_B of C_i, and estimates the conditional probability of heads given R'_A and C_i. The value with the highest estimated conditional probability is chosen; call it R_2. Then A' simulates A using random tape R_2 for the rest of the protocol. B' is the similar strategy for Bob, substituting R'_B's for R'_A's and letting R_2 be the R'_B with the lowest conditional probability.

Claim: Let $r(n)$ be the number of messages sent in the protocol with parameter n. Then either $Prob[P(C_{A'B}(R_{A'}, R_B) = 1] > 1/2 + 1/10r(n)$ or $Prob[P(C_{AB'}(R_A, R_{B'}) = 1] < 1/2 - 1/10r(n)$.

Proof: Fix R_A and R_B, and hence $C = C_{AB}(R_A, R_B)$. Let p_i be the conditional probability of heads given C_i, the first i messages of C. p_0=1/2, and $p_{r(n)} = 1$ or 0, depending on whether $P(C) = 1$ or 0. Thus, the expectation over a random i (and a random conversation C) of $|p_i - p_{i-1}|$ is at least $1/2r(n)$. Therefore, the same is true either for a random even i or a random odd i. Without loss of generality, assume it is true for a random odd i; i.e., after a random message sent by Alice. In this case, we show that A' successfully cheats with the prescribed probability; in the other case, a symmetric argument shows that B' achieves this probability.

Let a_i be the conditional probability of heads given R_A and C_i. Then for i odd, $a_i = a_{i-1}$, since Alice doesn't learn anything by sending a message. Since $|a_{i-1} - p_{i-1}| + |a_i - p_i| = |p_{i-1} - a_i| + |a_i - p_i| \geq |p_{i-1} - p_i|$, and this last quantity has expectation at least $1/2r(n)$ for a random odd i, we have that the expectation of $|a_i - p_i|$ for i, R_A, R_B chosen at random is at least $1/4r(n)$. But , given a fixed C_i, p_i is the expectation of a_i over random choices of R_A, i.e., if we let $a_i(R'_A)$ be the conditional probability of heads given C_i and R'_A, and let r be the number of possible such R'_A's given C_i, we have $p_i = \sum_{R'_A} a_i(R'_A)/r$. Therefore, $|a_i - p_i| \leq \sum_{R'_A} 1/r|a_i - a_i(R'_A)| = Exp(|a_i - a_i(R'_A)|$. Thus, the expectation of $|a_i(R_A) - a_i(R'_A)|$ for R_A and R_B chosen at random, i chosen at random, and R' chosen at random given $C_i(R_A, R_B)$, is at least $1/4r(n)$. Since R_A and R'_A are identically distributed, we conclude that the expectation of $\max\{0.a_i(R') - a_i(R)\}$ is at least $1/8r(n)$, i.e., $a_i(R'_A)$ is equally likely to be the larger, and this case contributes an equal share to the expectation of the absolute value of the difference. Thus, by picking the best of R_A and R'_A, Alice improves the probability of heads by at least $1/8r(n)$.

In the above analysis, we assumed that the procedure for uniformly generating pre-images of g was perfect; it is easy to show that allowing a small error will not significantly affect the performance of the algorithm. Thus, in the above claim, the perfect algorithm caused a difference of at least $1/8r(n)$, while we claimed only $1/10r(n)$ to allow for imperfections in the pre-image generating function. □

4.8 Summary

Our results, combined with other recent results, give a coherrent picture of the strength of cryptographic assumptions. The picture that emerges is of a three tiered structure. The bottom tier contains those cryptographic tasks which are solvable *information theoretically*; that is, without any complexity assumptions. These include one-time pads (i.e., private key cryptosystems where the length of the message is the length of the key), and, less trivially, secret sharing ([Bla],[Sham]), as well as a few others. The next level, which we will refer to as "low cryptography", consists of tasks which are possible if, and only if, one-way functions exist. The results in this chapter, Chapter 3, [H] and several other recent papers in theoretical cryptography can be summarized as proving that the following tasks and primitives are on this level: distributionally one-way function; function with false entropy; pseudo-random generators; pseudo-random function generator ([GGM]); pseudo-random permutation generator ([LR]); weak identification ; strong (zero-knowledge) identification ([FS]); bit commitment; bit commitment versus a strong committer ([Na]);zero-knowledge computation on committed inputs ([GMW]); coin flipping by telephone ([B2]); electronic signatures ([NY],[Rom]); and universal one-way hash functions ([NY], [Rom]).

On the other hand, ([IR]) presents strong evidence (e.g., a relativized separation) that it will not be possible to base secret agreement on an arbitrary one-way function, or on any assumption which is true of a random permutation. Thus, there seems to be a third tier, "high" cryptography, of primitives and tasks which imply the possibility of secret agreement. A partial list of these would contain: trapdoor one-way functions, oblivious transfer ([B3], [Rab2]), public key cryptography ([DH]), voting ([Ben]), and zero-knowledge multi-party computation ([Y2],[GHY], [CDG], [GMW2]). The exact structure of this tier is still mysterious. It is possible that,, unlike "low" cryptography, "high" cryptography will not admit a simple characterization in terms of necessary and sufficient assumptions for the tasks involved. For example, [Rud2] gives relativized worlds where secret agreement is possible in k rounds of communication, but not in $k-1$ rounds. Since any natural kind of cryptographic primitive that allows secret agreement is likely to do so in a fixed number of rounds, this seems to argue that there will be no natural primitive whose existence can be proved necessary and sufficient for secret agreement. Many open problems remain in the classification of assumption strength of tasks on this tier. For example, can any protocol for secret agreement yield a protocol for oblivious transfer (and hence, by [Ki], multi-party zero-knowledge computation)?

There are also a few open problems regarding classifying tasks which seem to fall between the second and third tiers. These include bit commitment vs. a strong receiver ([BC], [Cha]), and non-interactive zero-knowledge ([BFM]). Both of these tasks are possible with a truly random permutation, and so do not seem to be as strong as secret agreement, but are not known to be possible based only on a one-way function.

Bibliography

[AKS] Ajtai, M., Komlos, J., and Szemeredi, E., "Deterministic Simulation of Logspace", *Proceedings of 19th STOC, 1987.*

[AABFH] Abadi, Alander, E., Broder, A., Feigenbaum, J., and Hemachandra, L., "On Generating Solved Instances of Computational Problems," *Proceedings of Crypto, 1988.*

[AKLLR] Aleliunas, R.., Karp, R., Lipton, R., Lovasz, L., and Rackoff, C., "Random walks, universal traversal sequences, and the complexity of maze problems", *Proceedings of 20th FOCS, 1979*, pp. 218-223.

[B1] Blum, M., "Independent, unbiased coin flips from a correlated source: a finite state markov chain," *Proceedings of 25'th FOCS, 1984*, pp. 425-433.

[B2] Blum, M., "Coin Flipping by Phone," *IEEE Spring COMPCOM, Feb. 1982*, pp. 133-137.

[Ba] Bach, E., "Realistic Analysis of Some Randomized Algorithms", *Proceedings of 19'th STOC, 1987.*

[BBR] Bennett, C., Brassard, G., and Robert, J., "Privacy Amplification By Public Discussion," SIAM J. Comput., Vol 17, No.2, 1988.

[BBS] Blum, L., Blum, M., and Shub, M., "A Simple Unpredictable Pseudo-Random Number Generator," SIAM J. Comput., Vol. 15, No. 2, 1986.

[BC] Brassard, G., Crepeau, C., "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond," *Proceedings of 27'th FOCS, 1986.*

[BCG] Bellare, M., Cowen, L., Goldwasser, S., "Secret Key Exchange", unpublished manuscript, May 1989.

[Ben] Benaloh, J., (formerly, Cohen, J.) "Verifiable Secret Ballot Elections," Ph. D. Thesis, Yale University, Sept., 1987. Also Tech Report YALEU/DCS/TR-561.

[BFM] Blum, M., Feldman, P., and Micali, S., "Non-Interactive Zero-Knowledge and its Applications," *Proceedings of 20'th STOC, 1988*, pp. 103-112.

[BM] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits," *SIAM J. on Computing*, Vol. 13, 1984, pp. 850-864, *FOCS 1982*.

[BNS] Babai, L., Nisan, N., Szegedy, M., "Multiparty Protocols and Logspace-Hard Pseudorandom Sequences," *Proceedings of 21'st STOC, 1989*, pp. 1-11.

[C] Chaum, D., "Demonstrating that a public predicate can be satisfied without revealing any information about how," *Proceedings of CRYPTO 1986*.

[CDG] Chaum, D., Damgard, I., and van de Graaf, J., "Multi-party Computations Ensuring Secrecy of Each Party's Input and Correctness of the Output," *Proceedings of CRYPTO '87*.

[CG1] Chor, B., Goldreich, O., "On the power of two points based sampling", J. of Complexity, Vol. 5, 1989.

[CG2] Chor, B., and Goldreich, O., "Unbiased bits from sources of weak randomness and probabilistic communication complexity", *Proceeedings of 26'th FOCS, 1985*

[CGHFRS] Chor, B., Goldreich, O., Hastad, J., Friedman, J., Rudich, S., and Smolensky, R., "The bit extraction problem or t-resilient functions", *Proceedings of 26'th FOCS, 1985*, pp. 396-407.

[Co] Cook, S. A., "The Complexity of Theorem Proving Procedures", *Proceedings of 3'rd STOC, 1971*, pp. 151-158.

[CW] Carter, J., and Wegman, M., "Universal Classes of Hash Functions," *JCSS*, 1979, Vol. 18, pp. 143-154.

[CWi] Cohen, A., and Wigderson, A., "Dispersers, Deterministic Amplification, and Weak Random Sources," *Proceedings of 30'th FOCS, 1989*.

[DFK] Dyer, M., Frieze, A., and Kannan, R., "A Random Polynomial Time Algorithm for Approximating the Volume of Convex Bodies," *Proceedings of 21'st STOC, 1989*.

[FKL] Frieze, D., Kannan, R., and Lagarias, J. C., "Linear Congruential Generators Do Not Produce Random Sequences," *Proceedings of 25'th FOCS, 1984*.

[FS] Fiat, A., and Shamir, A., "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," *Proceedings of CRYPTO 1986*.

[GG] Gabber, O., and Galil, Z., "Explicit Construction of Linear-sized Superconcentrators," *Proceedings of 20'th FOCS, 1979.*

[GHY] Galil, Z., Haber, S. and Yung, M., "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystem," *Proceedings of 26'th FOCS, 1985*, pp. 360-371.

[GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions," *J. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807, *FOCS 1984*.

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

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

[GM] Goldwasser, S. and Micali, S., "Probabilistic Encryption," *JCSS*, Vol. 28, No. 2, April 1984, pp. 270-299, *STOC 1982*.

[GMR] Goldwasser, S., Micali, S. and Rackoff, C., "The Knowledge Complexity of Interactive Proof Systems," *SIAM J. on Computing*, Vol. 18, No. 1, 1989, pp. 186-208, *STOC 1985*.

[GMW] Goldreich, O., Micali, M. and Wigderson, A., "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design," *Proceedings of the 27'th FOCS, 1986*, pp. 174-187. Also, *Tech. Report TR498*, Comp. Sci. Dept., Technion, submitted to *JACM*.

[GMW2] Goldreich, O., Micali, S., Wigderson, A., "How to Play Any Mental Game," *Proceedings of 19'th STOC, 1987*, pp. 218-229.

[H] Hastad, J., "Pseudo-Random Generators under Uniform Assumptions," Proceedings of 22'nd STOC, 1990 pp. 395-404.

[HILL] Hastad, J., Impagliazzo, R., Levin, L., and Luby, M., "Pseudo-random generators from one-way functions," in preparation, 1991.

[ILL] Impagliazzo, R., Levin, L., Luby, M., "Pseudo-random generation from one-way functions," *Proceedings of STOC 1989.*

[IL1] Impagliazzo, R., Luby, M., "Pseudo-Random Number Generation from any One-way Function," International Computer Science Institute technical report *TR-89-002*, February, 1989.

[IL2] Impagliazzo, R., Luby, M., "One-Way Functions are Essential for Complexity-Based Cryptography," *Proceedings of 30'th FOCS, 1989.*

[IN] Impagliazzo, R., and Naor, M., "Efficient Cryptosystems Provably as Secure as Subset Sum," *Proceedings of 30'th FOCS, 1989*

[IR] Impagliazzo, R., and Rudich, S., "Limits on the Provable Consequences of One-Way Permutations," *Proceedings of 21'st STOC, 1989*, pp. 44-61.

[IZ] Impaglaizzo, R., and Zuckerman, D., "How to Recycle Random Bits," *Proceedings of 30'th FOCS, 1989.*

[JVV] Jerrum, M., Valiant, L., and Vazirani, V., "Random Generation of Combinatorial Structures from a Univorm Distribution," Theoretical Comp. Sci., 43: 169-188 (1986).

[Ki] Killian, J., "Founding Cryptography on Oblivious Transfer," *Proceedings of 20'th STOC, 1988*, pp. 20-31.

[Kn] Knuth, D., *The Art of Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley Series in Computer Science and Information Processing (R.S. Varga and M.A. Harrison, Consulting Editors), Addison-Wesley Publishing Co., 1969.

[KL] Karp, R., and Lipton, R., "Turing Machines That Take Advice," L'Enseignement Mathematique, Vol. 28, pp. 191-209 (1982). See also STOC 1980.

[KPS] Karp, R., Pippenger, N., and Sipser, M., "A time-randomness tradeoff," AMS Conference on Computational Complexity, Durham, N.H. 1985.

[KR] Karloff, H., and Raghavan, P., "Randomized Algorithms and Pseudorandom Numbers," *Proceedings of 20'th STOC, 1988.*

[KW] Karp, R., and Wigderson, A., "A Fast Parallel Algorithm for the Maximal Independent Set Problem," JACM, 32:4, 762-773 (Oct., 1985).

[Lev1] Levin, L. A., "One-Way Functions and Pseudo-random Generators,"Combinatorica, Vol. 7, No. 4, pp. 357-363, (1987) Also in STOC, 1985.

[LLS] Lichtenstein, D., Linial, N., and Saks, M., "Imperfect random sources and discrete control processes," *Proceedings of 19'th STOC, 1987*, pp. 169-177.

[LR] Luby M., and Rackoff, C., "How to Construct Pseudorandom Permutations From Pseudorandom Functions," *SIAM J. on Computing*, Vol. 17, 1988, pp. 373-386. Also in *STOC 1986.*

[McI] McInnes, J., "Cryptography Using Weak Sources of Randomness," Tech Report 194/87, University of Toronto, 1987.

[Mil] Miller, G., Riemann's hypothesis and tests for primality, JCSS, 13: 300-317 (1976).

[Na] Naor, M., "Bit Commitment from Pseudo-random Generators," *Proceedings of CRYPTO, 1989.*

[Ni] Nisan, N., "Pseudo-random Generators for Space-Bounded Computation," *Proceedings of 22'nd STOC, 1990* pp. 204-212.

[NY] Naor, M., and Yung, M., "Universal One-Way Hash Functions and Their Cryptographic Applications," *Proceedings, 21'st STOC, 1989*, pp.33-43.

[P] Plumstead, J., (now Boyar, J.), "Inferring a Sequence Generated by a Linear Congruence," *Proceedings of 24'th FOCS, 1983.*

[Rab] Rabin, M. O., "Probabalistic algorithm for testing primality," J. of Number Theory, 12: 128-138 (1980).

[Rac1] Rackoff, C., personal communication, 1989.

[Rac2] Rackoff, C., "A Basic Theory of Public and Private Cryptosystems," *Proceedings of CRYPTO 1988.*

[Ren] Renyi, A., *Probability Theory*, North-Holland, Amsterdam, 1970.

[Rom] Rompel, J., "One-Way Functions are Necessary and Sufficient for Secure Signatures," *Proceedings of 22'nd STOC, 1990*, pp. 387-394.

[Ru] Rudich, S., "Limits on the Provable Consequences of One-Way Functions," *Ph.D. Thesis*, Department of Computer Science, U.C. Berkeley, 1988.

[Sa] Santha, M., "On Using Deterministic Functions to Reduce Randomness in Probabilistic Algorithms," Imformation and Computation, 74: 241-249 (1987).

[Shan] Shannon, C., "A Mathematical Theory of Communication," *Bell Systems Technical Journal*, 27, 1948, pp. 379-423 and pp. 623-656.

[Sham] Shamir, A., "How to Share a Secret," *CACM*, 1979, 22, pp. 612-613.

[Si1] Sipser, M., "A Complexity Theoretic Approach to Randomness," *Proceedings of 15th STOC, 1983.*

[Si2] Sipser, M., "Expanders, Randomness, or Time Vs. Space," *Proceedings of Structure in Complexity, 1986.*

[Sp] Spencer, J., *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia,1987.

[SS] Solovay, R., and Strassen, V., A Fast Monte Carlo Algorithm for Primality, SIAM J. on Computing, 6:84-85 (1977).

[SSh] Schrift, A., and Shamir, A., "The Discrete Log is Very Discreet," *Proceedings of 22'nd STOC, 1990.*

[SV] Santha, M., and Vazirani, V., "Generating quasi-random sequences from slightly randomsources," *Proceedings of 25'th FOCS, 1984*, pp. 434-440.

[V1] Vazirani, U., "Efficiency Considerations Using Semi-Random Sources," *Proceedings, 19'th STOC, 1987*

[V2] Vazirani, U., "Randomness, Adversaries, and Computation," Ph.D. Thesis, U. of California, Berkeley, 1986.

[VV] Vazirani, U., and Vazirani, V., "Random polynomial-time is equal to semi-random polynomial-time," *Proceedings of 26'th FOCS, 1985*, pp. 417-428.

[Y1] Yao, A.C., "Theory and Applications of Trapdoor Functions," *Proceedings of 23'rd FOCS, 1982*, pp. 80-91.

[Y2] Yao, A. C., "How to Generate and Exchange Secrets," *Proceedings of the 27'th FOCS, 1986*, pp. 162-167.

Appendix A

Probability Theory From Scratch

Many of the classical results in probability theory concern to what extent and with what probability a random variable is close to its expectation. Throughout this thesis, and especially in Section 1, we have used such results, in particular Chernoff Bounds and the Martingale Tail Inequality[Sp]. In this appendix, we present alternate derivations for some of these results. Although our methods do not yield as tight numerical bounds as do the standard methods, we feel that they are interesting in that they have a very concrete, computational flavor.

We start by examining a very simple distribution, n tosses of a fair coin. Let the random variable H denote the number of coins which come up heads, and, similarly, T denote the number of tails. It is well-known that $|H - T|$ is usually around $n^{1/2}$. Why? Although not a fully intuitive explanation, the following proof relates this number to the probability $1/n$ of picking the same coin twice when two coins are selected at random. (The proof is a disguised way of computing the variance of the distribution.) The proof applies equally well to the more general case when the coins are not truly random and independent, but merely "pairwise independent", i.e., if c_i denotes the value of the i'th coin, we have $Pr[c_i = \text{“}Heads''|c_j = \text{“}Heads''] = Prob[c_i = \text{“}Heads''|c_j = \text{“}Tails''] = 1/2$ for every $i \neq j$.

Proposition A.0.1 *Let D be a pairwise independent distribution on n coins. Then $Exp[(H-T)^2] = n$.*

Proof: Consider the following probability: flip the coins according to D, then select two coins at random, with replacement, and see if they have the same value. The conditional probability of this event once we have flipped the coins is $(H/n)^2 + (T/n)^2 = (1/2(1 + (H-T)/n))^2 + (1/2(1 - (H-T)/n))^2 = 1/2(1 + (H-T)^2/n^2)$, so the overall probability is $1/2(1 + Exp[(H-T)^2]/n^2)$.

On the other hand, the probability remains the same if we pick the two coins first, and the conditional probability is 1 if the same coin was picked twice, $1/2$ otherwise, i.e., $1/n + (1-1/n)*1/2 = 1/2(1+1/n))$. Equating and solving for $Exp[(H-T)^2]$, we are done. $\Box$.

In the case of truly random coin tosses, the probability of greatly exceeding this expected value is very small. The following argument gives an intuitive reason why, although the version as stated below is circular in nature. We will return and show how to formalize this argument to get a slightly weaker version of Chernoff bounds.

Assume for now that n is even. Let's divide the coins into two groups of size$n/2$, and let H_1, H_2, T_1, T_2 denote the number of heads/tails in the two groups. Now $H-T = H_1 - T_1 + H_2 - T_2 = H_1 - (n/2 + H_1) + n/2 - T_2 - T_2 = 2(H_1 - T_2)$. Since H_1 is independent from T_2, and since "heads" and "tails" are totally symetric concepts, we might as well think of this as flipping $n/2$ coins twice, and doubling the difference between the two numbers of heads. Consider any individual coin in this set of $n/2$. The chance that it comes up the same in both trials, and, hence, contributes nothing to the final difference, is exactly $1/2$. If not, it is equally likely to add 1 to the difference, or subtract 1 from it. Thus, the distribution of the difference is exactly the same as the following quantity: Flip $n/2$ coins. Those that come up tails, ignore; those that come up heads, reflip. Take the difference between the number of reflipped coins that are heads, and the number of such coins that are now tails. Assuming what we are trying to prove, that with very high probability, k coins flipped have $k/2 + -k^{1/2+\epsilon}$ heads, this value should be upper bounded by looking at the number of heads less the number of tails when $n/4 + n^{1/2+\epsilon}$ coins are flipped. Thus, ifwe let $P(n,k)$ be the probability that $H-T > k$ when n coins are flipped, we have the fuzzy inequality, $P(n,k) <\approx P(n/4 + n^{1/2+\epsilon}, k/2) \approx P(n/4, k/2)$. Then we have $P(n,k) \approx P(n/4^i, k/2^i)$. Setting $i = logn - logk$, we have $P(n,k) \approx P(k^2/n, k^2/n) = 2^{-k^2/n}$.

This intuitive but non-formal argument gives us the same form of expression as Chernoff bounds; Chernoff shows $P(n,k) < e^{-k^2/2n}$. Unfortunately, in many cases Chernoff bounds are tight, so our "bound" must be wrong in these cases. We now use the intuition expressed above into a formally correct proof of a Chernoff-like bound, but which gives a weaker result. Since this gets a little complicated, and the idea of this appendix is to give simple proofs from first principles, the casual reader may wish to skip to Proposition A.0.7.

Part of the above argument was formally correct; we can summarize this part as :

Proposition A.0.2 *$P(2n, 2k) \leq P(n, 2d) + P(n/2 + d, k)$, for all integers n, k and d.*

We will use this proposition to "bootstrap" to be able to obtain a sequence of increasingly tight bounds on $P(n,k)$ of $exp(-O((k/n^\delta)^2))$ for any $\delta > 1/2$.

In particular, for $k = n^{1/2+\gamma}$, the bound is exponentially small. In order to start the bootstrapping process, we use a brute force calculation to get a bound on $P(n, cn)$ for a fixed constant $0 < c < 1$.

Proposition A.0.3 $n! \geq (n^n)2^{-2n}$.

Proof:

$n! = n(n-1)...1 > (n/2)^{n/2}(n/4)^{n/4}... = n^n/(2^{n/2}4^{n/4}8^{n/8}...) = n^n/2^{n(1/2+2/4+3/8+...)} = n^n 2^{-2n}$

.□

Proposition A.0.4 $P(n, 3/4n) \leq n2^{-3/8n}$.

Proof: Since $H-T \geq 3/4n$ when $T \leq 7/8n$, we have $P(n, 3/4n) = 2^{-n}\sum_{1\leq i\leq 1/8n}\binom{n}{i} \leq 2^{-n}(n/8)\binom{n}{n/8} \leq 2^{-n}(n/8)((n)^{n/8}/(n/8)!) \leq 2^{-n}(n/8)n^{n/8}/((n/8)^{n/8}2^{-n/4}) = n2^{-3/8n}$.

Proposition A.0.5 *For every $c, 0 < c < 1$, there is a $d, 0 < d < 1$ with $P(n, cn) \leq 2^{-dn}$ for sufficiently large n.*

Proof: It suffices to prove the proposition for $c = 2/3, 1/2, 1/3, 2/9, 4/27....$, which we do inductively. By the propositions A.0.2 (letting $d = n/16$ and A.0.4, we have $P(n, 2/3n) \leq P(n/2, 3/4n) + P(n2^{-3/16n} + P(7/16n, n/3)$. Since $n/3 = 16/21(7/16n) \geq 3/4(7/16n)$, applying proposition again, we get the claim for c=2/3. Similarly, we get the claim for $c = 1/2$, and $c = 1/3$. Assuming we have proved the claim for $c \leq 1/3$, we prove it for $2/3c$. $P(n, 2/3cn) \leq P(n/2, cn/2) + P(((1+c)/4)n, 1/3cn)$. Both of these two terms grows exponentially small by induction, the second because $(cn/3)/((1+c)n/4) = 4/3c/(1+c) \geq c$.□

We now use Proposition A.0.5 with a fixed value of c to get a bound for $P(n, k)$ for arbitrary k. The smaller we choose c, the better the bound (assymptotically).

Proposition A.0.6 *Let $1 > \delta > 1/2$. Then $P(n, k) \leq \log n 2^{(-d(k/n^{\delta})^2)}$ for some constant $d > 0$ and all but finitely many n.*

Proof: Let $a = 2^{-1/\delta}$, let $c = 4a - 1$, and let b be such that $P(n, cn) \leq 2^{-bn}$. Note, since $1/2 < \delta < 1$, $1/4 < a < 1/2$, so $0 < c < 1$, so $b > 0$ exists by Proposition A.0.5. Then applying Proposition A.0.2, using $d = cn/4$, we have $P(n, k) \leq P(n/2, cn/2) + P(an, k/2)$. We prove, by induction on n, that $P(n, k) \leq logn2^{-(b/2)\alpha^2}$, for $\alpha = k/n^{\delta}$. We have $P(n/2, cn/2) \leq 2^{-b/2n}$ by choice of b, and $P(an, k/2) \leq \log an2^{-\alpha'^2}$ for $\alpha' = k/2an^{\delta}$ by induction. But by our choice of a, $\alpha' = \alpha$. Also, $a < 1/2$, so $\log an < \log n - 1$, and $k < ni$ so $\alpha^2 < n$. Thus $P(n, k) \leq 2^{-b/2n} + (\log n - 1)2^{-b/2\alpha^2} \leq \log n2^{-b/2\alpha^2}$, which is what we wanted to show.

Note : The factor of $\log n$ can be removed by a more careful argument.

The above argument used many calculations; it is hard to claim that it is simpler than the standard proof of Chernoff bounds. However, the calculations involved were merely using the simple recurrence given by Proposition A.0.2, which is simple and intuitive, repeatedly. It is still conceivable that this method could yield a much simpler proof, or perhaps a tighter bound, than Chernoff's.

We now give a method for calculating bounds on the probabilities of diverging from an expectation for a large class of distributions. We directly reduce the problem for these distributions to that concerning flips of a fair coin.

We start with distributions determined by flipping independent, but biased, coins. Let the biases be $p_1, ...p_n$. We show that the probability that the number of coins flipped differs from the expected value by k is not much more than for fair coins. If the expected value is small, the probabilitywill be even smaller than for fair coins. Instead of directly proving that, with high probability, the number of heads is close to the expected number, we show first that, if the coins were flipped twice, with high probability, the number of heads would be approximately the same.

Proposition A.0.7 *Let $p_1, ...p_n$ be numbers between 0 and 1. Then, if two sets of n independent biased coins are flipped, with the biases $p_1, ..p_n$, the probability that the number of heads in the two sets differ by k or more is at most $P(n, k)$.*

Proof: Instead of flipping each coin twice, the difference between the two numbers can be generated as follows. For each coin i, flip a coin of bias ${p_i}^2 + (1-p_i)^2$ to decide whether the coin has the same value for the two trials, or different values. If they would be the same, the coin contributes nothing to the difference; if different, the coin contributes 1 to the difference with probability $1/2$, and -1 with probability $1/2$. Thus, after how many coins are the same and how many different has been determined, the conditional probability of the difference exceeding k is exactly $P(m, k)$, where m is the number of differences. Since $m \leq n$, this conditional probability is always less than $P(n, k)$.

Proposition A.0.8 *Let $p_1, ...p_n$ be numbers between 0 and 1; let $E = \sum p_i$. If coins are flipped independently with these biases, and if H is the random variable given by the number of heads, the probability that $|H - E| > 2k + 2nP(n, k)$ is at most $2P(n, k)$.*

Proof: By Proposition reffixed probs, if H_1 and H_2 are given by two independent samplings from the distribution, the probability that $|H_1 - H_2| > k$ is less than $2P(n, k)$. Thus, there is some value V for H_1 so that the conditional probability of this event is at most $2P(n, k)$, given $H_1 = V$. Then the expected number of heads, E, is at most $(V + k)(1 - 2P(n, k)) + n2P(n, k)$ and at least $(V - k)(1 - 2P(n, k))$, i.e., $|V - E| < k + 2P(n, k) \max V, n - V < k + 2nP(n, k)$.

Note that, if $k > c(n \log n)^{1/2}$, the $2nP(n, k)$ in the expression is less than 1.

For the case $E << n$, we can improve this bound.

Proposition A.0.9 *Let δ be fixed constants > 0 . Let $p_1, ..p_n$ be numbers between 0 and 1, let $E = \sum p_i$, and let $A = max(E, c(n \log n)^{1/2})$. Then, if independent coins are flipped with biases $p_1...p_n$, and X is the number of heads, the probability that $|X - E| > (A)^{1/2+\delta}$ is exponentially small, i.e. less than* $\exp -n^\epsilon$ *for some constant $\epsilon > 0$ and all but finitely many n.*

Proof: As before, consider $|X_1 - X_2|$ for two random samplings from this distribution. In the proof of Proposition A.0.7, apply Proposition A.0.8 to the number of coins where the two differed. The expectation of this number is $\sum 2p_i(1-p_i) < 2E$. Thus, the probability that there are more than $2E + 2l + 1$ such coins is at most $2P(n, l)$, for $l > c(n \log n)^{1/2}$. Pick $l = A^{1+\delta}$. Then $P(n, l)$ is exponentially small. Thus, with all but exponentially small probability, we flip fewer than $2E + 2l + 1 < 4l + 1$ fair coins to determine the final difference. In this case, the probability that the difference is more than $A^{1/2+\delta}$ less than $exp(-A^{1+2\delta}/8l)$ by Chernoff bounds. This last quantity is $exp(-A^\delta/8)$ which is exponentially small. That the same type of bound holds for the difference between one sampling and the expectation follows from an argument like that for Proposition A.0.8.

A more careful analsis along these lines will yield tighter bounds. In particular, one can to some extent bootstrap by using Proposition A.0.9 instead of 8 to bound the probability that $2E + 2l + 1$ coins are different.

A distribution on sequences of numbers $a_1, ...a_n$ has the Martengale Property if, for every $1 \le i \le n$, the expectation of a_i is independent from $a_1, ...a_{i-1}$, i.e., for every possible initial sequences $b_1, ...b_{i-1}$, the expectation of a_i given that $a_1 = b_1, ...a_{i-1} = b_{i-1}$ is the expectation of a_i. For Martengale distributions on bounded intervals, we directly reduce the probability that the sum of the $a_1, ..a_n$ differs from its expectation by a certain amount to the corresponding probability for biased coins, and, hence, for fair coins.

Proposition A.0.10 *Let $a_1, ...a_n$ be selected according to a Martengale distribution satisfying $0 \le a_i \le 1$ for each $1 \le i \le n$. Let e_i be the expectation of a_i, and $E = \sum e_i$. Let X be the random variable $X = \sum a_i$. Then the probability that $|E - X| > 4k + 2$ is at most $4P(n, k)$ for any $k > c(nlogn)^{1/2}$.*

Proof: Consider the following probability distribution. Pick $a_1, ...a_n$ according to the Martengale distribution. Then flip n independent coins of biases $a_1...a_n$. We claim the resulting distribution on sequences of heads and tails is the same as that obtained by flipping independent coins of biases $e_1, ...e_n$. For, whatever $a_1, ..a_{i-1}$ are, the expected value of a_i and hence of a coin flipped with bias a_i is exactly e_i by the Martengale property. Applying Proposition 8 then, the probability that, if H is the number of heads, $|H - E| > 2k + 1$ is at most $2P(n, k)$. However, we can also apply Proposition 8 to the coins flipped with biases a_i; whatever a_i's are chosen, the probability that $|H - \sum a_i| = |H - X| > 2k + 1$ is at most $2P(n, k)$. $\square$.

Corollary A.0.11 *If $a_1, ...a_n$ are selected according to a Martengale distribution satisfying $r \leq a_i \leq s$, then the probability, for E, X as above, that $|X - E| > (2k+1)(s-r)$ is at most $4P(n,k)$, for k as above.*

Proof: Let $b_i = (a_i - r)/(s-r)$. $b_1, ...b_i$ is Martengale, and $0 \leq b_i \leq 1$. Applying Proposition A.0.10 to $b_1, ...b_n$ yields the desired bound.

Corollary A.0.12 *Let $a_1, ...a_n$ be selected according to any distribution with $r \leq a_i \leq s$. Let X be as before, and let E be the random variable given by $E = \sum E(a_i|a_1, ...a_{1-1})$. Then the probability that $|X - E| > 2(s-r)(2k+1)$ is at most $4P(n,k)$ for k as above.*

Proof: Let $b_i = a_i - E(a_i|a_1, ...a_{i-1})$. Then, no matter what the values of $a_1, ...a_{i-1}$, and hence of $b_1, ...b_{i-1}$, the expected value of b_i is 0. Thus, $b_1, ...b_n$ is a Martengale distribution. Also, $r - s \leq b_i \leq s - r$, so applying Proposition A.0.10 to $b_1 ...b_n$ yields the desired bound.

This last corollary says that if we predict a_i according to our best guess given the previous a_j's, the sum of our errors is small. Our optimism will cancel with our pessimism.

Appendix B

Pseudo-Random Generators from One-Way Functions With Uniform Security

Here, we give a simplified version of Hastad's proof ([H]) that, under the uniform definition of security, any one-way function can be used to construct a cryptographically secure pseudo-random generator. Many of the results from Chapter 3 also hold in the uniform model. We will use the following results from Chapter 3:

Lemma B.0.13 *In the uniform definition of security, it is possible to construct a cryptographically secure pseudo-random generator from any function computable in polynomial time (from a parameter) which has false entropy at least* $1/n^c$ *for any constant* c*.*

Lemma B.0.14 *Let* f *be any one-way function. Then the function* f^* *defined by* $f^*(x,h,i) =< h, i, f(x), h(x)_i >$*, where* h *is a hash function from* $n = |x|$ *bits to* n *bits and* $0 \leq i \leq n$*, is one-way on the uniform distribution on the set* S *of triples* x, i, h *where* $i \leq \log|f^{-1}(f(x))| + 2\log n$*, and is one-to-one on all but an* $1/n$ *fraction of the set* S' *of triples* x, i, h *where* $i \geq \log|f^{-1}(f(x))| + \log n$*.*

In light of B.0.13 it suffices to construct, from a one-way function f, a function with false entropy (of the uniform type – i.e., there is a *polynomially sampleable* distribution with substantially more entropy than the generator, but which is computationally indistinguishable.) Also, it suffices for the function to be only computable from a parameter, by Chapter 2, Section 7, i.e., a $O(\log n)$ bit advice string (in other words, a polynomial size family of possible generators, one of which is good.) The construction uses the function f^* as in B.0.14. Then we use the Goldreich-Levin inner product bit to add some false entropy, as we did in Section 2. This gives us some false entropy when the input to f' is in $S \cap S'$, since then the inner product bit is almost certainly determined information-theoretically, but still computationally hidden. The problem is that it may be difficult to predict when this false entropy is occuring, i.e., when the input is in $S \cap S'$. The intuition behind Hastad's construction is that we do not try to predict this event. Instead, by picking many inputs to f' at random, we are guaranteed that, with very high probability, a proportional number of such inputs were picked. Then the collection of inner product bits taken as a whole will have a certain amount of false computational entropy (intuitively). We then use a random hash function, as we did before, to convert this false entropy to an almost equal amount of what look like uniformly distributed random bits. This will be our uniform false entropy function, since the distribution it will be indistinguishable from is simple and constructive: merely many copies of f' on random inputs, followed by a hash function and a certain number of random bits. Hastad proves a much stronger result than we give here; we prove only as much as we need for the main result.

Let $f'(x, i, h, r) = f^*(x, i, h) \circ r$, where $|r| = n$, and $b(x, i, h, r) = x \cdot r$. For simplicitly, we will denote the quadruple $< x, i, h, r >$ as y, and let T be the set of such quardruples with $< x, i, h > \in S$. From h B.0.14 above and the Goldreich-Levin Theorem, b is hidden for f' when r is selected at random and the other inputs are selected according to the uniform distribution on S, i.e., when y is a randomly chosen element of T. Moreover, both proofs can be thought of as giving strong reductions between the problems involved: there is an expected polytime algorithm, that, given as input a circuit C which predicts b given f' on this distribution with probability $1/2 + \delta$ on inputs with $|x| = n$, outputs a circuit C' which inverts f with probability at least a fixed polynomial in δ and $1/n$. (Here, time is polynomial in n, $1/\delta$ and $|C|$.) Thus, even though we may not be able to generate random elements of T, we can test whether a circuit C is a $1/2 + \delta$ predictor for $b(y)$ given $f'(y)$ for y picked randomly from T, by using the above algorithm and then testing whether the algorithm actually inverts $f(x)$ with the prescribed probability. Call this testing algorithm "TEST (C, δ)".

Let $d = Exp[\log(|f^{-1}(f(x))|)]$. Then the probability p that a random y is in T is $(d + 2\log n)/n$, and the probability of y not being in T' is $(d + \log n)/n = p - \log n/n$. We assume we know d within 1; this will be the "parameter", since there are only n possible values. Hastad's false entropy generator is given by: $g(y_1, y_2, \ldots, y_k, M) = M \circ f'(y_1) \circ \cdots \circ f'(y_k)\ M(b(y_1) \circ \cdots \circ b(y_k))$, where $k = k(n)$ is an appropriately chosen polynomial in n and M is a hash function mapping k bits to $pk - 2k^{2/3}$ bits.

Lemma B.0.15 *The distribution given by $g(y_1, \ldots, y_k, M)$ is indistinguishable from the distribution where we replace $M(b(y_1), \ldots, b(y_k))$ with $pk - k^{2/3}$ truly random bits. Moreover, there is an expected polynomial time algorithm which, given a circuit C which distinguishes the two distributions with probability δ,* *produces a circuit which inverts f with a polynomially related probability.*

Since f' is one-to-one on a set of size at least $p' - 1/n$, an easy calculation shows that the second distribution has entropy at least $k(p - p' - 1/n) - 2k^{2/3} = k(\log n - 1)/n - 2k^{2/3}$ more than the first. Letting $k = n^3$, this is at least n^2. Thus, since both distributions are polytime sampleable, this shows that, for $k = n^3$, g has false entropy at least n^2 in the uniform sense. So it suffices to prove the above Lemma in order to obtain:

Lemma B.0.16 *¿From any one-way function which is secure in the uniform sense, we can construct a function computable in polynomial-time from a parameter which has false entropy in the uniform sense.*

and, thus,

Theorem B.0.17 *¿From any one-way function which is secure against uniform algorithms, we can construct a pseudo-random generator which is secure against uniform algorithms.*

It remains to prove B.0.15.

Proof : For a circuit C and a distribution D , let $C(D)$ be the probability that $C(x) = 1$ when x is chosen according to D. Let D denote the distribution determined by g, and let D' denote the other distribution, where $M(b(y_1), \ldots, b(y_k))$ is replaced by an equal number of independent random bits. We are given a circuit C which distinguishes the two distributions with probability δ, i.e., $C(D) - C(D') = \delta$. Let $\epsilon = \delta/(20k)$. Our algorithm proceeds in k stages. At stage $i + 1$, we will have fixed values for $y_1, \ldots, y_i$; we will also, for each $1 \leq j \leq i$, have chosen either to "randomize" b_j, or to "set" b_j to $b(y_j)$. To avoid confusion, we will call the fixed value for y_i, "z_i". An outline of the program in each stage is as follows. First, a candidate circuit C_{i+1} is produced, and tested for predicting the bit b from f' for random elements of T with probability ϵ, using the procedure $TEST$ described before. If C_{i+1} is accepted by $TEST$, we are done. Otherwise, we flip a coin with bias p to decide whether to "randomize" b_{i+1}, or to "set" it. Finally, we must pick a good value for z_{i+1}; we do so by picking many candidates, and testing each. We pick the candidate with the best conditional distinguishing probability.

Let ${D^{set}}_i$ be the following distribution. For each $i + 1 \leq j \leq k$, pick a value for y_j at random, and let $b_j = b(y_j)$. For $1 \leq j \leq i$, let $y_j = z_j$. Let $b_j = b(z_j)$ if the j'th place has been set; if the j'th position has been randomized, pick $b_j \in \{0, 1\}$ at random. Pick matrix M at random. Output $f'(y_1) \circ \ldots \circ f'(y_k) \circ M \circ M(b_1, \ldots b_k)$. Let ${D^{rand}}_i$ be the same distribution, except b_{i+1} is chosen at random. Let ${E^{set}}_i$ be ${D^{set}}_i$ given that $y_{i+1} \in T$, and similarly for ${E^{rand}}_i$. In a similar manner, we define the distributions ${F^{rand}}_i$ and ${F^{set}}_i$, the conditional distributions given $y_{i+1} \notin T$. Finally, let D'_i be that distibution obtained from either of the D_i's (i.e, ${D^{set}}_i$ or ${D^{rand}}_i$) by replacing $M(b_1, \ldots b_k)$

by an equal number of random bits, and analagously for E'_i and F'_i. Let $\delta^D{}_i$ be the random variable $C(D^{set}{}_i) - C(D'_i)$, and analagously for $\delta^E{}_i$ and $\delta^F{}_i$. Note that $\delta^D{}_0 = \delta$, and that $\delta^D{}_i = p\delta^E{}_i + (1-p)\delta^F{}_i$. (Both distributions $D^{set}{}_i$ and D'_i are the "weighted sum" of the conditional distibutions $E^{set}{}_i$ (E'_i) and $F^{set}{}_i$ (F'_i), where the "E" distribution has weight p, the probabiltiy that a random $y_{i+1} \in T$, and the "F" distribution has weight $1-p$.)

Intuitively, the distributions $E^{set}{}_i$ and $E^{rand}{}_i$ are computationally indistinguishable, since $b(y_{i+1})$ is hard to predict from $f'(y_{i+1})$ given that $y_{i+1} \in T$. Formally, if $C(E^{set}{}_i) - C(E^{rand}{}_i) > 2\epsilon$, then the following probabilistic algorithm $P_i(C)$ predicts $b(y)$ from $f'(y)$ for a random $y \in T$ with probability at least $1/2+\epsilon$. Pick a random element of $D^{set}{}_i$, but change $f'(y_{i+1})$ to $f'(y)$, and try both 0 and 1 as possible values for $b(y_{i+1})$. This gives us two possible inputs to C, w_0, where we guess that $b(y) = 0$, and w_1, where we guess $b(y) = 1$. If $C(w_0) \neq C(w_1)$, P predicts the guess b with $C(w_b) = 1$. Otherwise, P outputs a random bit. (This is the standard way to prove that a hidden bit is computationally indistinguishable from an independently picked bit. We omit the details.)

Our algorithm first runs procedure $TEST$ on $P_i(C)$. If $P_i(C)$ passes $TEST$, we have found a circuit which inverts f with probability a fixed polynomial in ϵ and $1/n$, so we are done. Otherwise, we flip a coin of bias p, i.e, choose a bit that is 1 with probability p and 0 with probability $1-p$. If the coin is a 1, we will randomize the $i+1$'st position; otherwise, we will set it to $b(z_{i+1})$. In either case, we pick $n/(p(1-p)\epsilon^2)$ possible values for z_{i+1}. For each possibility, we estimate the corresponding $\delta^D{}_{i+1}$ by sampling many times from both distributions, $D^{set}{}_{i+1}$ and D'_{i+1}. (By sampling $1/\epsilon^3$ times from each distribution, our estimate will be within ϵ of the correct value with probability exponentially close to 1.) We choose the permanent value for z_{i+1} to be the candidate which yields the highest value of $\delta^D{}_{i+1}$. We then proceed to stage $i+2$.

Intuitively, what is going on is that we would like to pick z_{i+1} at random, but knowing whether z_{i+1} is in T; if it is, we are entitled to randomize b_{i+1}, since $b(z_{i+1})$ is then computationally hidden. In other words, we would like, with probabilityn p, to pick a random $z_{i+1} \in T$ and randomize b_{i+1}, and with probability $1-p$, to pick a random $z_{i+1} \notin T$ and set b_{i+1}. However, in general, we can't pick a random $z_{i+1} \in T$. On the other hand, if we pick many candidates, and take the best one (given that we have decided to randomize), with high probability, one of these candidates will be in T. Thus, the z_{i+1} we picked will either be a random element of T, or, if not, will be *at least as good as such an element for our purposes.* The same is true if we decide to set b_{i+1}.

Now that we have described the algorithm, we prove that it succeeds with a non-negligible probability.

Claim : With probability ϵ over runs of the algorithm, the above algorithm succeeds in finding a circuit which inverts f with probability a fixed polynomial in ϵ and $1/n$.

An outline of the proof of the claim is as follows. First, we show that, if we don't find a inverting circuit, then the expected value of δ_k is close to δ. But, with all but exponentially small probability, δ_k is exponentially small. To see this last, note that, since at each stage we randomize position j with probability p, with all but expontially small probability, at least $pk - k^{2/3}$ positions are randomized. In this case, the distributions D_k and D'_k will be statistically indistinguishable within an exponentially small amount by the Leftover Hash Lemma (Section 1); the only difference between the distributions is that a random hash of a random element of a set of size $2^{pk-k^{2/3}}$ is replaced by $pk - 2k^{2/3}$ random bits. (Aside: If we use the matrix hash for M, this part of the argument can be seen directly without recourse to the Leftover Hash Lemma. If the submatrix of M consisting of columns corresponding to positions which have been randomized is non-singular, then the distribution obtained by replacing these places with random bits and applying M to the result gives a uniform output. Since the number of rows of M is significantlyless than the number of places randomized, this submatrix will be non-singular with high probability.)

Let $\gamma_i = \delta^D{}_i$ if we have not found an inverting circuit by stage i, and let $\gamma_i = 1$ otherwise.

Sub-Claim 1: $Exp[\gamma_i] \geq \delta - 6i\epsilon$

Proof of Sub-Claim 1: If $P_i(C)$ is a good predictor for $b(y)$ on S, then with high probability, the procedure $TEST$ will find an inverting circuit in the $i+1'st$ stage, so $Exp[\gamma_{i+1}]$ is almost 1 in this case. Thus, we need to handle the case where a good predicter is not found.

Sub-Claim 1.1: Fix a run of the algorithm up to the i'th stage. If $P_i(C)$ does not predict $b(y)$ from $f(y)$ on T with probability at least $1/2 + \epsilon$, then $Exp[\delta^D{}_{i+1}] \geq \delta^D{}_i - 6\epsilon$.

Proof of Sub-Claim 1.1: If $P_i(C)$ does not predict $b(y)$ from $f(y)$ on T with probability $1/2+\epsilon$, then, as we have observed earlier, $C(E^{set}{}_i) - C(E^{rand}{}_i) < 2\epsilon$. We have also observed that $\delta^D{}_i = p\delta^E{}_i + (1-p)\delta^F{}_i$. We will show that, given that we decide to randomize b_{i+1}, $\delta^D{}_{i+1}$ is within 5ϵ of $\delta^E{}_{i+1}$, and that, given that we decide to set b_{i+1}, $\delta^D{}_{i+1}$ is within 5ϵ of $\delta^F{}_{i+1}$. Since the former event happens with probability p and the latter with probability $1-p$, we can use the aforementioned identity to show that the expectation of $\delta^D{}_{i+1}$ is within 6ϵ of $\delta^D{}_i$.

Sub-Claim 1.1.1: Given that the algorithm randomizes b_{i+1}, with probability at least $1-\epsilon$, $\delta^D{}_{i+1} \geq \delta^E{}_i - 5\epsilon$.

Proof of Sub-Claim 1.1.1: Let $\delta(z)$ be the value of $\delta^D{}_{i+1}$, given that $z_{i+1} = z$ and b_{i+1} is randomized. The algorithm in stage $i+1$ picks many ($n/p(1-p)\epsilon$) candidates z and estimates $\delta(z)$ for each candidate, picking the candidate with the largest estimate. The number of samples we use to estimate this value is such that with probability exponentially close to 1, all estimates are within ϵ of their actual value. Thus, with the same probability, $\delta(z_{i+1})$ for the value of z_{i+1} actually selected is within 2ϵ of the best among the candidates. On the other hand, for $z \in T$ chosen at random, the expectation of $\delta(z)$ is precisely $C(E^{rand}) - C(E') \geq C(E^{set}) - 2\epsilon - C(E') = \delta^E{}_i - 2\epsilon$. Since $\delta(z) \leq 1$, there must be at least an ϵ fraction of $z \in T$ which are within ϵ of this expectation, so, for a random z, $\delta(z) \geq \delta^E - 3\epsilon$ with probability at least $p\epsilon$. For the number of samples the algorithm takes, with probability exponentially close to 1, one such z is on the list of candidates, and so the best on the list also achieves this value of $\delta(z)$. $\Box$.

Sub-Claim 1.1.2: Given that the algorithm sets b_{i+1}, then with probability at least $1 - \epsilon$, $\delta_{i+1} \geq \delta^F{}_i - 5\epsilon$.

Proof of Sub-Claim 1.1.2: The proof is similar to Sub-Claim 1.1.1. Let $\delta(z)$ now mean the value of $\delta^D{}_{i+1}$ given that $z_{i+1} = z$ and b_{i+1} is set. Then, as before, with high probability, for the value of z_{i+1} actually chosen, $\delta(z_{i+1})$ is within 2ϵ of the largest $\delta(z)$. On the other hand, the expectation of $\delta(z)$ for $z \notin T$ is exactly $\delta^F{}_i$. Thus, at least a $(1-p)\epsilon$ fraction of z's are within ϵ of this value, and so with high probability, the best z on our list of candidates is. $\Box$.

Combining Sub-Claims 1.1.1 and 1.1.2, the expectation of $\delta^D{}_{i+1}$ is at least $p(1-\epsilon)(\delta^E{}_i - 5\epsilon) + (1-p)(1-\epsilon)(\delta^F{}_i - 3\epsilon) \geq p\delta^E{}_i + (1-p)\delta^F{}_i - 6\epsilon = \delta^D{}_i - 6\epsilon$. This completes the proof of Sub-Claim 1.1. $\Box$

Since, with high probability, γ_{i+1} is either 1, if we have found an inveting circuit, or $\delta^D{}_{i+1}$ if we have not, we have immediately that, fixing the first i stages of the algorithm, $Exp[\gamma_{i+1}] \geq \gamma_i - 6\epsilon$. Sub-Claim 1 follows by simply noting that the expectation of a variable is the expectation of its conditional expectation given other variables. Thus, we have $Exp[\gamma_{i+1}] \geq Exp[\gamma_i] - 6\epsilon$, so $Exp[\gamma_i] \geq \gamma_0 - 6i\epsilon = \delta - 6i\epsilon$. $\Box$.

The special case of Sub-Claim 1 when $i = k$ gives us: $Exp[\gamma_k] \geq \delta - 6k\epsilon \geq \delta/2$. Since γ_k is bounded above by 1, there must be at least a $\delta/4$ fraction of runs where γ_k is at least $\delta/4$. But, as we have argued previously, the fraction of runs where δ_k is greater than an exponentially small amount is exponentially small. Therefore, there must be a $\delta/8$ fraction of runs where $\gamma_k \neq \delta_k$; this only occurs if $P_i(C)$ is a good predictor for some i, at which point we have an exponentially close to 1 chance of finding an inverting circuit. This proves the Claim and the Theorem . $\Box$.