PSEUDORANDOMNESS
AND
COMPUTATIONAL DIFFICULTY

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF SCIENCE

HUGO KRAWCZYK

SUBMITTED TO THE SENATE OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY

SHVAT 5750 HAIFA FEBRUARY 1990

A mis padres, Aurora y Bernardo,
a quienes debo mi camino.

A Leonor, compañera inseparable.

A Liat, dulzura y continuidad.

This research was carried out in the Computer Science Department under the supervision of Professor Oded Goldreich.

To Oded, my deep gratitude for being both teacher and friend.
For the innumerable hours of stimulating conversations.
For his invaluable encouragement and support.

I am grateful to all the people who have contributed to this research and given me their support and help throughout these years.
Special thanks to Benny Chor, Shafi Goldwasser, Eyal Kushilevitz, Jacob Ukelson and Avi Wigderson. To Mike Luby thanks for his collaboration in the work presented in chapter 2.
I also wish to thank the Wolf Foundation for their generous financial support.

Finally, thanks to Leonor, without whom
all this would not have been possible.

Table of Contents

Abstract

In recent years, *randomness* has become a central notion in diverse fields of computer science. Randomness is used in the design of algorithms in fields as computational number theory, computational geometry, parallel and distributed computing, and it is crucial for cryptography. Since in most cases the interest is in the behavior of efficient algorithms (modeled by polynomial-time computations), the fundamental notion of *pseudorandomness* arises. Pseudorandom distributions are probability distributions on sets of strings that cannot be efficiently distinguished from the uniform distribution on the same sets. In other words, any efficient probabilistic algorithm performs essentially as well when substituting its source of unbiased coins by a sequence sampled from a pseudorandom distribution. In this thesis we investigate the existence of pseudorandom distributions and the computational difficulty of generating them.

Pseudorandomness is practically beneficial if pseudorandom sequences can be generated more easily than "truly random" ones. This gives rise to the notion of a *pseudorandom generator* - an efficient deterministic algorithm which *expands* truly random strings into longer pseudorandom sequences. The existence of pseudorandom generators is not yet proven. Such a proof of existence would imply the solution of the most important open problem in theoretical computer science. It would imply the existence of one-way functions and, in particular, that $P \neq NP$. Thus, as long as we cannot settle these questions, the existence of (polynomial-time) pseudorandom generators can be proven only under intractability assumptions. In this thesis, we present a new sufficient condition for the existence of such generators. We show that pseudorandom generators can be constructed using *regular one-way functions*. Regular functions are functions that map the same number of elements to every element in the range of the function (the actual condition is more general). The novelty of our work is both in weakening previous sufficient conditions for constructing pseudorandom generators, and in presenting a new technique for iterating a (regular) one-way function while preserving its one-wayness during the repeated iterations. In particular, this result allows the construction of pseudorandom generators based on specific intractability assumptions that were not known to be sufficient for this task. Examples are the (conjectured) intractability of general factoring, the (conjectured) intractability of decoding random linear codes, and the (conjectured) average-case difficulty of some combinatorial problems (e.g. subset-sum).

We also investigate the existence of pseudorandom distributions when decoupled from the notion of efficient generation. We prove, without relying on any unproven assumption, the existence and samplability of *sparse* pseudorandom distributions, which are substantially different from the uniform distribution. We demonstrate the existence of non-polynomial generators of

pseudorandomness achieving optimal expansion rate. These algorithms have also "optimal" complexity measures (as running time or circuit size), in the sense that improving these measures would lead to major breakthroughs in Complexity Theory.

We prove the existence of pseudorandom distributions which are *evasive*, that is, any efficient algorithm trying to find an element in the support of the distribution (i.e. elements assigned with non-zero probability), will succeed to do so with only negligible probability. This result allowed us to resolve two open problems concerning the composition of zero-knowledge proof systems. We prove that the original definition of zero-knowledge (involving uniform verifiers without no auxiliary input) is not robust under sequential composition, and that even the strong formulations of zero-knowledge are not closed under parallel composition. Other results on the round complexity of zero-knowledge interactive proofs, with significant implications to the parallelization of zero-knowledge protocols, are also presented.

Finally, we investigate whether some classical number generators, called *congruential number generators*, are pseudorandom generators. These algorithms are extensions of the well-known linear congruential generator, and are of interest because of their simplicity and efficiency. We prove that these number generators are not pseudorandom since they can be efficiently predicted. We present an efficient algorithm which, on input a prefix of the generated sequence, guesses the next element in the sequence with a good probability of success. This extends previous results on the predictability of congruential generators and, in particular, it implies an affirmative answer to the open question of whether multivariate polynomial recurrences are efficiently predictable.

Chapter 1:

Introduction

In recent years *randomness* has become a central notion in the theory of computation. The ability of algorithms to "toss coins" enables them to break the limitations of determinism and allows finding more efficient solutions to many problems. For some applications, randomness is even more essential. A traditional example is the field of computer based simulations. Another such a field, central to this thesis, is cryptography.

The amount of randomness consumed by an algorithm is measured in terms of the number of coins tossed by the algorithm. These coin flips are represented by a string of random bits fed into the algorithm. Generating such bits is in many cases an expensive process, and thus randomness becomes a new resource, in addition to classical resources as time and space. Economizing on the amount of random bits required by an application becomes a natural concern.

It is in this light that the notion of *pseudorandomness* and, in particular, of a *pseudorandom generator* arises. Pseudorandom generators are deterministic algorithms which expand short random strings into much longer "pseudorandom" sequences. Informally, the concept of a pseudorandom sequence means that such a sequence, which is clearly not really random, is "as good" as truly random bits for computational purposes. For many years the concept of pseudorandomness was treated as a vague notion lacking clear definitions. In that approach, most of the effort was concentrated on showing that for specific families of sequences, some statistical properties of random sequences do hold. A drawback of this approach is that in practice one must analyze these sequences for each new application according to the characteristics and needs of that application. From the theoretical point of view this approach is unsatisfactory as it does not suggests a uniform definition of pseudorandomness.

A breakthrough in the study of pseudorandomness was achieved in the works by Blum and Micali [BM] and Yao [Y]. These works present a uniform treatment of the concept of pseudorandomness, suitable for *any* efficient (i.e. polynomial-time) application. In this approach a probability distribution is associated to the set of binary strings of a given length. Loosely speaking, this distribution is called *pseudorandom* if it cannot be efficiently distinguished from the uniform distribution on strings of the same length. In other words, efficient probabilistic algorithms perform essentially as well when substituting its source of unbiased coins by a pseudorandom sequence. Thus, for any practical purposes there is no difference between an ideal source of randomness and the pseudorandom source. Moreover, algorithms can be analyzed assuming they

use unbiased coin tosses, and later implemented using pseudorandom sequences. In this approach a pseudorandom generator is an efficient deterministic algorithm which expands random strings into longer ones and which induces on its output a pseudorandom distribution.

The above definition of pseudorandomness is the strongest possible as long as efficient computations are concerned. But, do such sources of pseudorandomness exist. Can they be effectively generated? In this thesis we investigate the existence of pseudorandom distributions and the computational difficulty of generating them. An application of our results on pseudorandomness to the theory of zero-knowledge interactive proofs is also presented. The following sections overview our results.

1.1 Sufficient conditions for the existence of pseudorandom generators.

The existence of (efficient) pseudorandom generators is not yet proven. A limitation in our actual capability to prove such a claim follows from the fact that the existence of pseudorandom generators implies the existence of *one-way functions* (functions which are easy to evaluate but infeasible to invert). Whether such functions do exist is an outstanding open problem in Complexity Theory. In particular, it implies that $P \neq NP$. Thus, as long as we cannot settle these questions we also cannot prove the existence of pseudorandom generators without relying on some intactability assumptions.

A basic question is what are the minimal assumptions we need in order to prove the existence of pseudorandom generators, as well as to be able to construct these generators. The study of this question was initiated in the works by Blum and Micali [BM] and Yao [Y]. They showed that the existence of one-way permutations is a sufficient condition. (A permutation is a length-preserving bijective function). Moreover, given a one-way permutation one can use it for explicitly constructing a pseudorandom generator. The basic scheme for this construction, proposed in [BM], repetitively applies the one-way permutation, outputting one pseudorandom bit per each application. The basic property of permutations in this context is that they preserve the uniform distribution on the domain of application of the function. This property guarantees the difficulty of inverting the permutation even after repeated iterations.

Levin [L] proposed a weaker sufficient condition, namely the existence of functions which are "one-way on the iterates" (i.e., functions that remain one-way after repeated applications). Although this condition (for the existence of pseudorandom generators) is also a necessary one, it is somewhat cumbersome and difficult to check for specific functions (not being permutations). Furthermore, it did not lead to finding new natural functions on which one can base the construction of pseudorandom generators.

In this thesis we present new sufficient conditions for the existence of pseudorandom generators. We present a construction of pseudorandom generators based on the existence of any one-way function which is one-to-one. Moreover, a wider family of one-way functions called *regular* functions do suffice. These are functions in which every image of an n-bit string has the same number of preimages of length n. (Actually, an even weaker condition suffices).

Our condition has several significant implications regarding the construction of pseudorandom generators. First, it constitutes a new sufficient condition for the existence of pseudorandom generators, which is weaker than the one-way permutations condition. In particular, it gets rid of the length-preservation property. Second, it is the first construction that successfuly deals with functions that are not necessarily one-way on the iterates (recall that one-way permutations always have this property). Our construction transforms any regular one-way function into a function which is one-way on the iterates. Third, the new condition allows basing the construction of pseudorandom generators on specific functions which were not known before to be suitable for this task. We show how to construct pseudorandom generators based on different intractability assumptions. Examples are the intractability of general factoring, the conjectured intractability of decoding random linear codes, and the assumed average-case difficulty of some combinatorial problems (e.g. subset-sum). Finally, our results and techniques inspired the works by Impagliazzo, Levin and Luby [ILL] and Hastad [Ha] (which proved the sufficiency of any one-way function for constructing pseudorandom generators), and the work by Naor and Yung (which based digital signatures on one-way permutations [NY]).

1.2 The existence of sparse pseudorandom distributions.

As long as we cannot prove the conjectures on which the construction of pseudorandom generators has to be based, we cannot give a definite proof of existence of such generators. A natural question is what can be proved if we renounce to the requirement of *efficient* generation. A prime concern is whether the existence of pseudorandom distributions can be proved without relying on unproven assumptions.

The above question must be carefully formulated since trivial examples of pseudorandom distributions do exist. Uniform distributions (i.e., truly random sources) are such a case. On the other hand, distributions produced by pseudorandom generators are much more "interesting". These distributions are *sparse* as they concentrate their mass on a very small subset of strings. For example, consider a generator which expands n-bit strings into $2n$-bit sequences. The support (i.e. the set of elements assigned non-zero probability) of the induced distribution contains at most 2^n strings, which is a negligible fraction of the 2^{2n} possible strings of length $2n$. Clearly, these distributions are essentially different from the uniform distribution.

In this thesis we prove the existence of *sparse* pseudorandom distributions, independently of any intractability assumption. Moreover, we show that sparse pseudorandom distributions can be uniformly generated by probabilistic algorithms (that run in non-polynomial time). These generating algorithms use less random coins than the number of pseudorandom bits they produce. Viewing these algorithms as generators which expand randomly selected short strings into much longer pseudorandom sequences, we can exhibit (non-polynomial) generators achieving subexponential expansion rate. This expansion is optimal as we show that no generator expanding strings into exponential longer ones can induce a pseudorandom distribution (which passes non-uniform tests). On the other hand, we use the subexponential expansion property in order to construct non-uniform generators of size slightly super-polynomial. An improvement to this result, namely, a proof of existence of non-uniform polynomial-size generators would separate non-uniform-P (P/poly) from non-uniform-NP (NP/poly), which would be a major breakthrough in Complexity Theory.

We also prove the existence of sparse pseudorandom distributions that cannot be generated or even approximated by efficient algorithms. Namely, there exist pseudorandom distributions that are statistically far from any distribution which is induced by any probabilistic polynomial-time algorithm. In other words, even if efficiently generable pseudorandom distributions exist, they do not exhaust (nor even in an approximative sense) all the pseudorandom distributions.

Finally, we introduce the notion of *evasive* probability distributions. These probability distributions have the property that any efficient algorithm will fail to find strings in their support (except with a negligible probability). Certainly, evasive probability distributions are sparse, and cannot be efficiently approximated by probabilistic algorithms. We show the existence of *evasive pseudorandom* distributions.

Interestingly, we have applied the above "abstract-flavored" results in order to resolve two open questions concerning the sequential and parallel composition of zero-knowledge interactive proofs. This application is described in section 1.4.

1.3 The predictability of congruential generators.

In section 1.1 we have seen that the construction of pseudorandom generators running in polynomial-time is possible under some intractability assumptions. As pointed out, these assumptions are not a weakness of the specific known constructions but are unavoidable for achieving the requirements of pseudorandomness. On the other hand, for many practical applications the proposed generators work too slowly. This fact calls for finding more efficient pseudorandom generators. Natural candidates to be considered are some well-known and simple generators as the linear congruential generator (see bellow) and its generalizations. Our prime concern

is whether these generators produce pseudorandom sequences.

A natural requirement from a pseudorandom sequence is to pass the *predictability test*. This means that seeing a prefix of the generated sequence should not help guessing its continuation. More precisely, no efficient algorithm should predict the next bit in the sequence with a probability which is significantly better than 1/2. Interestingly, Yao [Y] has shown that this property, which is crucial for cryptographic applications, is not only a natural requirement from pseudorandom sequences but it is equivalent to the indistinguishability condition in the definition of pseudorandomness. In other words, a distribution of strings is pseudorandom if and only if it passes the predictability test.

In this part of the thesis we deal with *number* generators which produce, from an initial input, a sequence of integer numbers. In this setting, we consider predicting algorithms which interact with the generator. For every element in the sequence, the predictor outputs its guess of the next number before it gets the correct value from the generator. The efficiency of the predicting algorithm is measured both by the number of prediction mistakes and the time it takes to compute each prediction. Clearly, an efficient predicting algorithm in this sense implies a next-bit predictor as referred above.

An example of a number generator is the *linear congruential generator* which on input integers a, b, m, s_0 outputs a sequence $s_1, s_2, \cdots$ where $s_i \equiv a\, s_{i-1} + b \,(mod\ m)$. Boyar [B] proved that this generator is efficiently predictable, even when a, b and m are unknown. She presented a predicting algorithm which errs on at most $O(log\ m)$ elements and computes each guess in polynomial-time. Her method was extended to deal with more general cases. In particular, Boyar proved the predictability of the multilinear congruential generator ($s_i \equiv \alpha_1\, s_{i-k} + \cdots + \alpha_k\, s_{i-1}\ (mod\ m)$) and Lagarias and Reeds [LR] proved a similar result for polynomial recurrences ($s_i \equiv p(s_{i-1})(mod\ m)$ for an unknown polynomial p of fixed degree). A natural generalization of the above examples are *multivariate polynomial recurrences*, that is, generators for which $s_i \equiv P(s_{i-n}, \ldots, s_{i-1})(mod\ m)$ for a polynomial P in n variables. Finding efficient predictors for these generators remained an open problem.

In this thesis we study a wide family of number generators called *general congruential generators*. This family, introduced by Boyar, includes as special cases all the above examples. These generators are defined by modular recurrences consisting of a linear combination of arbitrary functions working on the past sequence elements (e.g., in the case of multivariate polynomial recurrences the functions are the corresponding monomials). The predictor knows these basis functions but not the coefficients of the linear combination or the modulus of the recurrence. Boyar's predicting method applies to a subclass of these generators. Here, we extend these results showing how to predict any efficient congruential generator. We require that the

basis functions are computable in polynomial time when working over the integers. In particular, we show that multivariate polynomial recurrence generators are efficiently predictable.

Our predicting technique is based on ideas from Boyar's method, but our approach to the prediction problem is somewhat different. Boyar's method tries to simulate the generator by "discovering" its secrets, that is, the modulus and the coefficients that the generator works with. Instead, our algorithm uses only the knowledge that these coefficients exist, but does not try to explicitly find them.

1.4 The composition of zero-knowledge interactive-proofs.

We present an application of our results on the existence of sparse and evasive pseudorandom distributions (see section 1.2) to the theory of zero-knowledge proof systems. In particular, we resolve two open problems concerning the sequential and parallel composition of zero-knowledge interactive proofs.

Zero-knowledge proof systems, introduced by Goldwasser, Micali and Rackoff [GMR1], are efficient interactive proofs which have the remarkable property of yielding nothing but the validity of the assertion. Namely, whatever can be efficiently computed after interacting with a zero-knowledge prover, can be efficiently computed on input a valid assertion. Thus, a zero-knowledge proof is computationally equivalent to an answer by a trusted oracle.

A natural requirement from the notion of zero-knowledge proofs is that the information obtained by the verifier during the execution of a zero-knowledge protocol will not enable him to extract any additional knowledge from subsequent executions of the same protocol. That is, it is desirable that the *sequential composition* of zero-knowledge protocols would yield a protocol which is itself zero-knowledge. Such a property is crucial for the utilization of zero-knowledge proof systems as subprotocols inside cryptographic protocols (otherwise, the security of the entire protocol would be compromised by the serial execution of these subprotocols).

Soon after the introduction of the notion of zero-knowledge, several researchers noticed the importance of the preservation of zero-knowledge under sequential composition. It was conjectured that the original formulation of zero-knowledge is probably not closed under sequential composition. Consequently, stronger formulations of zero-knowledge were proposed for which the preservation property was proved to hold (see [F, GMR2, O, TW]). Feige and Shamir [F] suggested a protocol which supports the above conjecture. Here, we use ideas from this protocol and our results on evasive pseudorandom distributions, to prove that indeed the original formulation of zero-knowledge is not closed under sequential composition.

The *parallel composition* of two (or more) interactive proofs is a protocol resultant from the concurrent execution of these proofs. Parallel composition of interactive proofs is widely

used as means for decreasing the error probability of proof systems, while maintaining the number of iterations they involve. Of course one would be interested to apply these advantages of parallelism also to zero-knowledge protocols. This would be possible if the parallel composition of interactive proofs preserves zero-knowledge. Unfortunately, we show that this is not the case. We present two protocols which are (computational) zero-knowledge with respect to the strongest known definitions, yet their parallel composition is not zero-knowledge (not even in the "weak" sense of the original [GMR1] formulation). These protocols use pseudorandom collections which are evasive against non-uniform polynomial-time machines, and whose existence is proven in this thesis.

The above result rules out the possibility of proving that particular interactive proofs are zero-knowledge by merely arguing that they are the result of parallel composition of various zero-knowledge protocols. But this does not resolve the question whether concrete cases of composed interactive proofs are zero-knowledge. In particular, since the early works on zero-knowledge it was repeatedly asked whether the "parallel versions" of the zero-knowledge proofs presented for Quadratic Residuosity [GMR1], Graph Isomorphism and for any language in NP [GMW1] are also zero-knowledge.

Our results concerning this question are reported in chapter 6 of this thesis. We prove that these "parallel" interactive proofs cannot be proven zero-knowledge using *black-box simulation*, unless the corresponding languages are in BPP. We say that an interactive proof *is proven zero-knowledge using black-box simulation* if there exists a universal simulator which using any verifier V^* as a black box, successfully simulates the conversations of (the same) V^* with the prover. Not only that *all known* zero-knowledge interactive proofs are proven zero-knowledge using a black-box simulation, but it is hard to conceive an alternative way of proving the zero-knowledge property of such an interactive proof.

The "parallel versions" of the above examples constitute interactive proofs of 3 rounds. The impossibility to prove them black-box zero-knowledge follows from our general result stating that *only BPP languages have 3-round interactive proofs which are black-box simulation zero-knowledge.* Moreover, we prove that *languages having constant-round Arthur-Merlin proofs which are black-box simulation zero-knowledge are in BPP.* (Arthur-Merlin proofs [Ba] are interactive proofs with "public coins", i.e. in which all the messages sent by the verifier are the outcome of his coin tosses).

Other consequences of these results are a proof of optimality for the round complexity of various known zero-knowledge protocols, and a structure theorem for the hierarchy of Arthur-Merlin zero-knowledge languages. In particular, these results can be viewed as a support to the conjecture that "secret coins" help in the zero-knowledge setting.

Chapter 2:

On the Existence of Pseudorandom Generators

2.1. INTRODUCTION

In this chapter we present our results concerning the sufficiency of regular one-way functions in order to construct pseudorandom generators.

Pseudorandom generators are efficient deterministic algorithms which expand short seeds into longer bit sequences which are polynomially-indistinguishable from the uniform probability distribution. Formally, we have the following definition.

Definition 2.1.1: A *pseudorandom generator* G is a deterministic polynomial time algorithm which on input a string of length k outputs a string of length $k' > k$ such that for every polynomial time algorithm (distinguishing test) T, any constant $c > 0$, and sufficiently large k

$$\left| Prob(T(G(U_k))=1) - Prob(T(U_{k'})=1) \right| \leq k^{-c},$$

where U_m is a random variable assuming as values strings of length m, with uniform probability distribution.

It follows that the strings output by a pseudorandom generator G can substitute the unbiased coin tosses used by any polynomial time algorithm A, without changing the behavior of algorithm A in any noticeable fashion. This yields an equivalent polynomial time algorithm, A', which randomly selects a seed, uses G to expand it to the desired amount, and then runs A using the output of the generator as the random source required by A.

The notion of a pseudorandom generator was first suggested and developed by Blum and Micali [BM] and Yao [Y]. The theory of pseudorandomness was further developed to deal with function generators and permutation generators and additional important applications to cryptography have emerged [GGM, LuR, N]. The existence of such seemingly stronger generators was reduced to the existence of pseudorandom (string) generators.

In light of their practical and theoretical value, constructing pseudorandom generators and investigating the possibility of such constructions is of major importance. A necessary condition for the existence of pseudorandom generators is the existence of one-way functions (since the generator itself constitutes a one-way function). On the other hand, stronger versions of the one-wayness condition were shown to be sufficient. Before reviewing these results, let us recall the definition of a one-way function.

Definition 2.1.2: A function $f:\{0,1\}^* \to \{0,1\}^*$ is called *one-way* if it is polynomial time computable, but not "polynomial time invertible". Namely, there exists a constant $c > 0$ such that for any probabilistic polynomial time algorithm A, and sufficiently large k

$$Prob\Big(A(f(x),1^k) \notin f^{-1}(f(x)) \Big) > k^{-c}, \tag{2.1.1}$$

where the probability is taken over all x's of length k and the internal coin tosses of A, with uniform probability distribution.
(Remark: The role of 1^k in the above definition is to allow algorithm A to run for time polynomial in the length of the preimage it is supposed to find. Otherwise, any function which shrinks the input by more than a polynomial amount would be considered one-way.)

2.1.1. Previous Results

The first pseudorandom generator was constructed and proved valid, by Blum and Micali, under the assumption that the discrete logarithm problem is intractable on a non-negligible fraction of the instances [BM]. In other words, it was assumed that exponentiation modulo a prime (i.e. the 1-1 mapping of the triple (p, g, x) to the triple $(p, g, g^x \bmod p)$, where p is prime and g is a primitive element in Z_p^*) is one-way. Assuming the intractability of factoring integers of the form $N = p \cdot q$, where p and q are primes and $p \equiv q \equiv 3 \bmod 4$, a simple pseudorandom generator exists [BBS, ACGS]. [1] Under this assumption the permutation, defined over the quadratic residues by modular squaring, is one-way.

Yao has presented a much more general condition which suffices for the existence of pseudorandom generators; namely, the existence of one-way permutations [Y].[2]

Levin has weakened Yao's condition, presenting a necessary and sufficient condition for the existence of pseudorandom generators [L]. Levin's condition, hereafter referred to as *one-way on iterates*, can be derived from Definition 2.1.2 by substituting the following line instead of line (2.1.1)

$$(\forall i, 1 \le i < k^{c+2}) \quad Prob\Big(A(f^{(i)}(x),1^k) \notin f^{-1}(f^{(i)}(x)) \Big) > k^{-c},$$

where $f^{(i)}(x)$ denotes f iteratively applied i times on x. (As before the probability is taken uniformly over all x's of length k.) Clearly, any one-way permutation is one-way on its iterates. It

[1] A slightly more general result, concerning integers with all prime divisors congruent to 3 mod 4, also holds [CGG].

[2] In fact, Yao's condition is slightly more general. He requires that f is 1-1 and that there exists a probability ensemble Π which is invariant under the application of f and that inverting f is "hard on the average" when the input is chosen according to Π.

is also easy to use any pseudorandom generator in order to construct a function which satisfies Levin's condition. Unfortunately, this condition is somewhat cumbersome. In particular, it seems hard to test the plausibility of the assumption that a particular function is one-way on its iterates.

2.1.2. Our Results

In this thesis we consider "regular" functions, in which every element in the range has the same number of preimages. More formally, we use the following definition.

Definition 2.1.3: A function f is called *regular* if there is a function $m(\cdot)$ such that for every n and for every $x \in \{0,1\}^n$ the cardinality of $f^{-1}(f(x)) \cap \{0,1\}^n$ is $m(n)$.

Clearly, *every* 1-1 function is regular (with $m(n)=1, \forall n$). Our main result is

Theorem 2.1.1: *If there exists a regular one-way function then there exists a pseudorandom generator.*

A special case of interest is of 1-1 one-way functions. The sufficiency of these functions for constructing pseudorandom generators does not follow from previous works. In particular, Yao's result concerning one-way permutations does not extend to 1-1 one-way functions.

Regularity appears to be a simpler condition than the intractability of inverting on the function's iterates. Furthermore, many natural functions (e.g. squaring modulo an integer) are regular and thus, using our result, a pseudorandom generator can be constructed assuming that any of these functions is one-way. In particular, if factoring is weakly intractable (i.e. every polynomial time factoring algorithm fails on a non-negligible fraction of the integers) then pseudorandom generators do exist. This result was not known before. (It was only known that the intractability of factoring a special subset of the integers implies the existence of a pseudorandom generator.) Using our results, we can construct pseudorandom generators based on the (widely believed) conjecture that decoding random linear codes is intractable, and on the assumed average case difficulty of combinatorial problems as subset-sum.

Theorem 2.1.1 is proved essentially by transforming any given regular one-way function into a function that is one-way on its iterates (and then applying Levin's result [L]).

It is interesting to note that not every (regular) one-way function is "one-way on its iterates". To emphasize this point, consider the following example of a one-way function ($\bar{f}$) which is trivially invertible on the distribution obtained by iterating the function *twice*: Let f be any one-way function and assume for simplicity that f is length preserving (i.e. $|f(x)|=|x|$). Let

$|x|=|y|$ and define $\bar{f}(xy)=0^{|y|}f(x)$. Clearly, $\bar{f}$ is one-way. On the other hand, for every $xy \in \{0,1\}^{2n}$, $\bar{f}(\bar{f}(xy))=0^n f(0^n)$ and $0^n f(0^n) \in \bar{f}^{-1}(0^n f(0^n))$. Also notice that if f is regular then so is $\bar{f}$.

The novelty of our work is in presenting *a direct way to construct a function which is one-way on its iterates from any regular one-way function (which is not necessarily one-way on its iterates).*

2.1.3. Subsequent Results

Recent results of Impagliazzo, Levin and Luby [ILL] and Hastad [Ha] extend our results showing the sufficiency, for constructing pseudorandom generators, of any one-way function. Thus, the equivalence of pseudorandom generators and one-way functions is stated.

2.2. THE CONSTRUCTION OF PSEUDORANDOM GENERATORS

2.2.1. Preliminaries

In the sequel we make use of the following definition of *strongly* one-way function. (When referring to Definition 2.1.2, we shall call the function *weak* one-way or simply one-way).

Definition 2.2.1: A polynomial time computable function $f:\{0,1\}^* \to \{0,1\}^*$ is called *strongly one-way* if for any probabilistic polynomial time algorithm A, any positive constant c, and sufficiently large k,

$$Prob\left(A(f(x),1^k) \in f^{-1}(f(x))\right) < k^{-c},$$

where the probability is taken over all x's of length k and the internal coin tosses of A, with uniform probability distribution.

Theorem (Yao [Y]): There exists a strong one-way function if and only if there exists a (weak) one-way function. Furthermore, given a one-way function, a strong one can be constructed.

It is important to note that Yao's construction preserves the regularity of the function. Thus, we may assume without loss of generality, that we are given a function f which is strongly one-way and regular.

For the sake of simplicity, we assume f is *length preserving* (i.e. $\forall x, |f(x)|=|x|$). Our results hold also without this assumption (see subsection 2.2.7).

Notation: For a finite set S, the notation $s \in_R S$ means that the element s is randomly selected from the set S with uniform probability distribution.

2.2.2. Levin's Criterion: A Modified Version

The proof of Theorem 2.1.1 relies on the transformation of a function which is one-way and regular into a function which satisfies a variant of Levin's one-way on iterates condition. The modified condition relates to functions which leave the first part of their argument unchanged. It requires that the function is one-way on a number of iterates which exceeds the length of the second part of its argument. (Levin requires that the function is one-way on a number of iterations exceeding the length of the *entire* argument.)

More precisely, we consider functions $F(\cdot,\cdot)$ defined as

$$F(h, x) = (h, F_0(h, x))$$

That is, F applies a function F_0 on its arguments and concatenates the first argument h to this result. We prove the following condition.

Lemma 2.2.1: A sufficient condition for the existence of a pseudorandom generator is the existence of a function F of the form

$$F(h, x) = (h, F_0(h, x))$$

such that F is strongly one-way for $|x|+1$ iterations.

Before proving Lemma 2.2.1, let us recall Blum-Micali scheme for the construction of pseudorandom generators [BM]. This scheme uses two basic elements. The first is a (strongly) one-way function f, and the second is a boolean predicate $b(\cdot)$ called a "hard-core" of the function f. (Roughly speaking, a Boolean function $b(\cdot)$ is a *hard-core predicate of* f, if it is polynomial time computable, but no polynomial time probabilistic algorithm given $f(x)$, for randomly selected x, can compute the value of $b(x)$ with a probability significantly better than 1/2). A pseudorandom generator G is constructed in the following way. On input x (the seed), the generator G applies iteratively the one-way function $f(\cdot)$ on x for t (= $poly(|x|)$) times (i.e. $f(x)$, $f^{(2)}(x),\ldots, f^{(t)}(x)$). In each application of f, the predicate $b(f^{(i)}(x))$ is computed and the resultant bit is output by the generator. That is, G outputs a string of length t. Blum and Micali show that the above sequence of bits is unpredictable when presented in reverse order (i.e. $b(f^{(t)}(x))$ first and $b(f^{(1)}(x))$ last), provided that the boolean function $b(\cdot)$ is a hard-core predicate on the distribution induced by the iterates $f^{(i)}, 0 \le i \le t$. The unpredictability of the sequence is proved by showing that an algorithm which succeeds to predict the next bit of the sequence with probability better than one half can be transformed into an algorithm for "breaking" the hard-core

of the function f. Finally applying Yao's result [Y] that unpredictable sequences are pseudorandom we get that the above G is indeed a pseudorandom generator.

The crucial ingredient in the proof of Levin's condition, as well as of our modified version, is the existence of a hard-core predicate for any (slightly modified) one-way function. A recent result of Goldreich and Levin [GL] greatly simplifies the original proof in [L]. This result states that any function $f'(x,r)=(f(x),r)$, where $|x|=|r|$, has a hard-core predicate for the uniform distribution on r and any distribution on x for which f is strongly one-way. This hard-core predicate is the inner product modulo 2 of r and x (viewed as vectors over Z_2).

Finally, we recall the following notable property of pseudorandom generators: in order to have a generator which expands strings to any polynomial length, it suffices to construct a generator which expands strings of length k into strings of length $k+1$. This generator can be iteratively applied for polynomially many times without harming the pseudorandomness of its output [GrM]. We now prove Lemma 2.2.1.

Proof of Lemma 2.2.1: Note that $F^{(i)}(h,x)=(h,F_0^{(i)}(h,x))$. Thus, the condition in the Lemma implies that $F_0(h,x)$ is hard to invert for $|x|+1$ iterations even when h is given to the inverter. We construct the following generator, G, which expands its input by one bit. Let s be the seed for G, so that $s=(r,h,x)$, where $|x|=n, |r|=n$. Then, we define

$$G(s)=G(r,h,x)=(r,h,b_0,\cdots,b_n)$$

where for $i=0,\ldots,n$, b_i is the inner product modulo 2 of r and $F_0^{(i)}(h,x)$. (We denote $F_0^{(0)}(h,x)=x$).

We claim that this generator is pseudorandom. This is proved by noting that the output string is unpredictable. This is true for the r and h part as they were chosen as truly random strings. For the other bits this is guaranteed by Goldreich-Levin result and the fact that F_0 is hard to invert for $n+1$ iterations (even when h is given to the inverter). □

2.2.3. Main Ideas

We prove Theorem 2.1.1 by transforming any regular and (strongly) one-way function f into a new strongly one-way function F for which the conditions of Lemma 2.2.1 hold.

The following are the main ideas behind this construction. Since the function f is strongly one-way, any algorithm trying to invert f can succeed with only negligible probability. Here the probability distribution on the range of f is induced by choosing a random element from the domain and applying f. However, this condition says nothing about the capability of an algorithm to invert f when the distribution on the range is substantially different. For example, there may be an algorithm which is able to invert f if we consider the distribution on the range elements induced by choosing a random element from the domain and applying f twice or more

(see example in section 2.1.2). To prevent this possibility, we "randomly" redistribute, after each application of f, the elements in the range to locations in the domain. We prove the validity of our construction by showing that the probability distribution induced on the range of f by our "random" transformations (and the application of f) is close to the distribution induced by the first application of f.

The function F we construct must be deterministic, and therefore the "random" redistribution must be deterministic (i.e. uniquely defined by the input to F). To achieve this, we use high quality hash functions. More specifically, we use hash functions which map n-bit strings to n-bit strings, such that the locations assigned to the strings by a randomly selected hash function are uniformly distributed and n-wise independent. For properties and implementations of such functions see [CW, J, CG, Lu]. We denote this set of hash functions by $H(n)$. Elements of $H(n)$ can be described by bit strings of length n^2. In the sequel $h(\in H(n))$ refers to both the hash function and to its representation.

2.2.4. The Construction of F

We view the input string to F as containing two types of information. The first part of the input is the description of hash functions that implement the "random" redistributions and the other part is interpreted as the input for the original function f.

The following is the definition of the function F:

$$F(h_0, \cdots, h_{t(n)-1}, i, x) = (h_0, \cdots, h_{t(n)-1}, i^+, h_i(f(x)))$$

where $x \in \{0,1\}^n$, $h_j \in H(n)$, $0 \le i \le t(n)-1$. The function $t(n)$ is a polynomial in n, and i^+ is defined as $(i+1) \bmod t(n)$.

The rest of this section is devoted to the proof of the following theorem.

Theorem 2.2.2: Let f be a regular and strongly one-way function and let $t(n)$ be any polynomial. Then the function F defined above is strongly one-way for $t(n)$ iterations on strings x of length n.

Theorem 2.1.1 follows from Theorem 2.2.2 and Lemma 2.2.1 by choosing $t(n) > n$.

Let $h_0, h_1, \cdots, h_{t(n)-1}$ be $t(n)$ functions from the set $H(n)$. For $r = 1, \cdots, t(n)$, let g_r be the function $g_r = f\, h_{r-1}\, f\, h_{r-2}\, f \cdots h_0\, f$ acting on strings of length n, let $G_r(n)$ be the set of all such functions g_r, let g be $g_{t(n)}$ and let $G(n)$ be the set of such functions g. From the above description of the function F it is apparent that the inversion of an iterate of F boils down to the problem of inverting f when the probability distribution on the range of f is $g_r(x)$ where $x \in_R \{0,1\}^n$. We show that, for most $g \in G(n)$, the number of preimages under g for each element in its range is close (up to a polynomial factor) to the number of preimages for the same range element under

f. This implies that the same statement is true for most $g_r \in G_r(n)$ for all $r = 1, \cdots, t(n)$. The proof of this result reduces to the analysis of the combinatorial game that we present in the next subsection.

2.2.5. The game

Consider the following game played with M balls and M cells where $t(n) \ll M \leq 2^n$. Initially each cell contains a single ball. The game has $t(n)$ iterations. In each iteration, cells are mapped randomly to cells by means of an independently and randomly selected hash function $h \in_R H(n)$. This mapping induces a transfer of balls so that the balls residing (before an iteration) in cell σ are transferred to cell $h(\sigma)$. We are interested in bounding the probability that some cells contain "too many" balls when the process is finished. We show that after $t(n)$ iterations, for $t(n)$ a polynomial, the probability that there is any cell containing more than some polynomial in n balls is negligibly small (i.e. less than any polynomial in n fraction).

We first proceed to determine a bound on the probability that a specific set of n balls is mapped after $t(n)$ iterations to a single cell.

Lemma 2.2.3: The probability that a specific set of n balls is mapped after $t(n)$ iterations to the same cell is bounded above by $p(n) = \left(\frac{n \cdot t(n)}{M} \right)^{n-1}$.

Proof: Let $B = \{b_1, b_2, \cdots, b_n\}$ be a set of n balls. Notice that each execution of the game defines for every ball b_i a path through $t(n)$ cells. In particular, fixing $t(n)$ hash functions $h_0, h_1, \cdots, h_{t(n)-1}$, a path corresponding to each b_i is determined. Clearly, if two such paths intersect at some point then they coincide beyond this point. We modify these paths in the following way. The initial portion of the path for b_i that does not intersect the path of any smaller indexed ball is left unchanged. If the path for b_i intersects the path for b_j for some $j < i$ then the remainder of the path for b_i is chosen randomly and independently of the other paths from the point of the first such intersection.

Because the functions h_i are chosen totally independently of each other and because each of them has the property of mapping cells in an n-independent manner, it follows that the modified process just described is equivalent to a process in which a totally random path is selected for each ball in B. Consider the modified paths. We say that two balls b_i and b_j *join* if and only if their corresponding paths intersect. Define *merge* to be the reflexive and transitive closure of the relation join (over B). The main observation is that if $h_0, h_1, \cdots, h_{t(n)-1}$ map the balls of B to the same cell, then $b_1, b_2, \cdots, b_n$ are all in the same equivalence class with respect to the relation merge. In other words, the probability that the balls in B end up in the same cell in the original game is bounded above by the probability that the merge relation has a single equivalence class

(containing all of B). Let us now consider the probability of the latter event.

If the merge relation has a single equivalence class then the join relation defines a connected graph with the n balls as vertices and the join relation as the set of edges. The "join graph" is connected if and only if it contains a spanning tree. Thus, an upper bound on the probability that the "join graph" is connected is obtained by the sum of the probabilities of each of the possible spanning trees which can be embedded in the graph. Each particular tree has probability at most $(t(n)/M)^{n-1}$ to be embedded in the graph ($t(n)/M$ is an upper bound on the probability of each edge to appear in the graph). Multiplying this probability by the (Cayley) number of different spanning trees (n^{n-2} cf. [E, Sec. 2.3]), the lemma follows. □

A straightforward upper bound on the probability that there is some set of n balls which are merged is the probability that n specific balls are merged multiplied by the number of possible distinct subsets of n balls. Unfortunately, this bound is worthless (as $\binom{M}{n} \cdot p(n) > 1$ (This phenomena is independent of the choice of the parameter n.). Instead we use the following technical lemma.

Lemma 2.2.4: Let S be a finite set, and let Π denote a partition of S. Assume we have a probability distribution on partitions of S. For every $A \subseteq S$, we define $\chi_A(\Pi)=1$ if A is contained in a single class of the partition Π and $\chi_A(\Pi)=0$ otherwise. Let n and n' be integers such that $n < n'$. Let $p(n)$ be an upper bound on the maximum over all $A \subseteq S$ such that $|A|=n$ of the probability that $\chi_A=1$. Let $q(n')$ be an upper bound on the probability that there exists some $B \subseteq S$ such that $|B| \geq n'$ and $\chi_B=1$. Then

$$q(n') \leq \frac{\binom{|S|}{n} \cdot p(n)}{\binom{n'}{n}}$$

Proof: For $B \subseteq S$ we define $\xi_B(\Pi)=1$ if B is exactly a single class of the partition Π and $\xi_B(\Pi)=0$ otherwise. Fix a partition Π. Observe that every $B, |B| \geq n'$, for which $\xi_B(\Pi)=1$, contributes at least $\binom{n'}{n}$ *different* subsets A of size n for which $\chi_A=1$. Thus we get that

$$\binom{n'}{n} \cdot \sum_{B \subseteq S, |B| \geq n'} \xi_B(\Pi) \leq \sum_{A \subseteq S, |A|=n} \chi_A(\Pi)$$

Dividing both sides of this inequality by $\binom{n'}{n}$, and averaging according to the probability distribution on the partitions Π, the left hand side is an upper bound for $q(n')$, while the right hand side is bounded above by $\frac{\binom{|S|}{n} \cdot p(n)}{\binom{n'}{n}}$. □

Remark 2.2.1: Lemma 2.2.4 is useful in situations when the ratio $\frac{p(n)}{p(n')}$ is smaller than $\binom{|S|-n}{n'-n}$. Assuming that $n' \ll |S|$, this happens when $p(n)$ is greater than $|S|^{-n}$. Lemma 2.2.3 is such a case, and thus the application of Lemma 2.2.4 is useful.
Combining Lemmas 2.2.3 and 2.2.4, we get

Theorem 2.2.5: Consider the game played for $t(n)$ iterations. Then, the probability that there is $4t(n) \cdot n^2 + n$ balls which end up in the same cell is bounded above by 2^{-n}.

Proof: Let S be the set of M balls in the above game. Each game defines a partition of the balls according to their position after $t(n)$ iterations. The probability distribution on these partitions is induced by the uniform choice of the mappings h. Theorem 2.2.5 follows by using Lemma 2.2.4 with $n' = 4t(n) \cdot n^2 + n$, and the bound $p(n)$ of Lemma 2.2.3. □

2.2.6. Proof of Theorem 2.2.2

We now apply Theorem 2.2.5 to the analysis of the function F. As before, let $G(n)$ be the set of functions of the form $g = f\, h_{t(n)-1}\, f \cdots h_0\, f$. The functions $h = h_j$ are hash functions used to map the range of f to the domain of f. We let $h_0, \cdots, h_{t(n)-1}$ be randomly chosen uniformly and independently from $H(n)$, and this induces a probability distribution on $G(n)$. Denote the range of f (on strings of length n) by $R(n) = \{z_1, z_2, \ldots, z_M\}$. Let each z_i represent a cell. Consider the function h as mapping cells to cells. We say that h maps the cell z_i to the cell z_j if $h(z_i) \in f^{-1}(z_j)$, or in other words $f(h(z_i)) = z_j$. By the regularity of the function f, we have that the size of $f^{-1}(z_i)$ (which we have denoted by $m(n)$) is equal for all $z_i \in R(n)$, and therefore the mapping induced on the cells is uniform. It is now apparent that $g \in_R G(n)$ behaves exactly as the random mappings in the game described in Section 2.2.5, and thus Theorem 2.2.5 can be applied. We get

Lemma 2.2.6: There is a constant c_0, such that for any constant $c > 0$ and sufficiently large n

$$Prob\left(\exists\, z \text{ with } |g^{-1}(z)| \geq n^{c_0} \cdot m(n)\right) \leq \frac{1}{n^c},$$

where $g \in_R G(n)$.

Let us denote by $G'(n)$ the set of functions $g \in G(n)$ such that for all z in the range of f, $|g^{-1}(z)| < n^{c_0} \cdot m(n)$. By the above lemma, $G'(n)$ contains almost all of $G(n)$. It is clear that if $g \in G'(n)$ then for all z in the range of f and for all $r = 1, \cdots, t(n)$ the function g_r defined by the first r iterations of g satisfies $|g_r^{-1}(z)| < n^{c_0} \cdot m(n)$.

Lemma 2.2.7: For any probabilistic polynomial time algorithm A, for any positive constant c and sufficiently large n and for all $r = 1, \cdots, t(n)$,

$$Prob(A(g_r, z) \in f^{-1}(z)) < n^{-c}$$

where $g_r \in_R G_r(n)$ and $z = g_r(x)$, $x \in_R \{0,1\}^n$.

Proof: We prove the claim for $r = t(n)$ and the claim for $r = 1, \cdots, t(n)$ follows in an analogous way. Assume to the contrary that there is a probabilistic polynomial time algorithm A and a constant c_A such that $Prob(A(g, z) \in f^{-1}(z)) > n^{-c_A}$, where $g \in_R G(n)$ and $z = g(x)$, $x \in_R \{0,1\}^n$.

By using A, we can demonstrate an algorithm A' that inverts f, contradicting the one-wayness of f. The input to A' is $z = f(x)$ where $x \in_R \{0,1\}^n$. A' chooses $g \in_R G(n)$ and outputs $A(g, z)$. We show that A' inverts f with non-negligible probability. By assumption there is a non-negligible subset $G''(n)$ of $G'(n)$ such that, for each $g \in G''(n)$, A succeeds with significant probability to compute a $y \in f^{-1}(z)$ where $z = g(x)$ and $x \in_R \{0,1\}^n$. Since $g \in G'(n)$, for all z in the range of f the probability induced by g on z differs by at most a polynomial factor in n from the probability induced by f. Thus, for $g \in G''(n)$, A succeeds with significant probability to compute a $y \in f^{-1}(z)$ where $z = f(x)$ and $x \in_R \{0,1\}^n$. This is exactly the distribution of inputs to A', and thus A' succeeds to invert f with non-negligible probability, contradicting the strong one-wayness of f. □

The meaning of Lemma 2.2.7 is that the function f is hard to invert on the distribution induced by the functions $g_r, r = 1, \ldots, t(n)$, thus proving the strong one-wayness of the function F for $t(n)$ iterations. Theorem 2.2.2 follows.

2.2.7. Extensions

In the above exposition we assumed for simplicity that the function f is length preserving, i.e. $x \in \{0,1\}^n$ implies that the length of $f(x)$ is n. This condition is not essential to our proof and can be dispensed with in the following way. If f is not length preserving then it can be modified to have the following property: For every n, there is an n' such that $x \in \{0,1\}^n$ implies that the length of $f(x)$ is n'. This modification can be carried out using a padding technique that preserves the regularity of f. We can then modify our description of F to use hash functions mapping n'-bit strings to n-bit strings. Alternatively, we can transform the above f into a length preserving and regular function $\hat{f}$ by defining $\hat{f}(xy) = f(x)$, where $|x| = n, |y| = n' - n$.

For the applications in Section 2.3, and possibly for other cases, the following extension (referred to as *semi-regular*) is useful. Let $\{f_x\}_{x \in \{0,1\}^*}$ be a family of regular functions, then our construction can be still applied to the function f defined as $f(x, y) = (x, f_x(y))$. The idea is to use the construction for the application of the function f_x, while keeping x unchanged.

Another extension is a relaxation of the regularity condition. A useful notion in this context is the histogram of a function.

Definition 2.2.2: The *histogram* of the function $f:\{0,1\}^* \to \{0,1\}^*$ is a function $hist_f: \mathbf{N}\times\mathbf{N} \to \mathbf{N}$ such that $hist_f(n,k)$ is the cardinality of the set

$$\{x \in \{0,1\}^n : \left\lfloor \log_2 |f^{-1}(f(x))| \right\rfloor = k\}$$

Regular functions have trivial histograms. Let f be a regular function such that for all $x \in \{0,1\}^n$, $|f^{-1}(f(x))| = m(n)$. The histogram satisfies $hist_f(n,k) = 2^n$ for $k = \lfloor \log_2(m(n)) \rfloor$ and $hist_f(n,k) = 0$ otherwise. Weakly regular functions have slightly less dramatic histograms.

Definition 2.2.3: The function f is *weakly regular* if there is a polynomial $p(\cdot)$ and a function $b(\cdot)$ such that the histogram of f satisfies (for all n)

i) $$hist_f(n, b(n)) \geq \frac{2^n}{p(n)}$$

ii) $$\sum_{k=b(n)+1}^{n} hist_f(n,k) < \frac{2^n}{(n \cdot p(n))^2}$$

Clearly, this definition extends the original definition of regularity. Using our techniques one can show that the existence of weakly regular strongly one-way functions implies the existence of pseudorandom generators.

Observe that if the $b(n)$-th level of the histogram contains all of the 2^n strings of length n then we can apply a similar analysis as done for the regular case. The only difference is that we have to analyze the game of subsection 2.2.5 not for cells of equal size, but for cells that differ in their size by a multiplicative factor of at most two. Similar arguments hold when considering the case where the $b(n)$-th level of the histogram contains at least $1/p(n)$ of the strings and the rest of strings lie below this level (i.e. $hist_f(n,k) = 0$, for $k > b(n)$). Note that the "small" balls of low levels cannot cause the cells of the $b(n)$-th level to grow significantly. On the other hand, for balls bellow level $b(n)$ nothing is guaranteed. Thus, we get that in this case the function F we construct is weakly one-way on its iterates. More precisely, it is hard to invert on its iterates for at least a $1/p(n)$ fraction of the input strings. In order to use this function for generating pseudorandom bits, we have to transform it into a strongly one-way function. This is achieved following Yao's construction [Y] by applying F in parallel on many copies. For the present case the number of copies could be any function of n which grows faster than $c \cdot p(n) \cdot \log n$, for any constant c. This increases the number of iterations for which F has to remain one-way by a factor equal to the number of copies used in the above transformation. That is, the number $t(n)$ of necessary iterates increases from the original requirement of $n+1$ (see section 2.2.2) to a quantity which is greater than $c \cdot p(n) \cdot n \cdot \log n$, for any constant c. Choosing this way the function $t(n)$ in the definition of F in section 2.2.4, we get F which is one-way for the right number of iterations.

Finally, consider the case in which there exist strings above the $b(n)$-th level. When considering the game of subsection 2.2.5 we want to show that, also in this case, most of the cells of the $b(n)$-th level do not grow considerably. This is guaranteed by condition (ii) in Definition 2.2.3. Consider the worst case possibility in which in every iteration the total weight of the "big" balls (those above level $b(n)$) is transferred to cells of the $b(n)$-th level. After $t(n)$ iterations this causes a concentration of "big" balls in the $b(n)$-th level having a total weight of at most $t(n) \cdot \frac{2^n}{(n \cdot p(n))^2}$. Choosing $t(n) = ½\, p(n)\, n^2$ this weight will be at most $\frac{2^n}{2\, p(n)}$. But then one half of the weight in the $b(n)$-th level remains concentrated in balls that were not effected by the "big" balls. In other words we get that the function F so constructed is one-way for $t(n)$ iterations on $\frac{1}{2\, p(n)}$ of the input strings. Applying Yao's construction , as explained above, we get a function F which satisfies the criterion of Lemma 2.2.1 and then suitable for the construction of pseudorandom generators.

Further Remarks:

1) The denominator in condition (ii) of Definition 2.2.3 can be substituted by any function growing faster than $c \cdot p^2(n) \cdot n$, for any constant c. This follows from the above analysis and the fact that the construction of a hard-core predicate in [GL] allows extracting $\log n$ secure bits with each application of the one-way function.

2) The entire analysis holds when defining histograms with polynomial base (instead of base 2). Namely, $hist_f(n, k)$ is the cardinality of the set

$$\{x \in \{0,1\}^n : \left\lfloor \log_{Q(n)} |f^{-1}(f(x))| \right\rfloor = k\}$$

where $Q(n)$ is a polynomial.

2.3. APPLICATIONS : Pseudorandom Generators Based on Particular Intractability Assumptions

In this section we apply our results in order to construct pseudorandom generators (*PRG*s) based on the assumption that one of the following computational problems is "hard on a non-negligible fraction of the instances".

2.3.1. PRG Based on the Intractability of the General Factoring Problem

It is known that pseudorandom generators can be constructed assuming the intractability of factoring integers of a special form [Y]. More specifically, in [Y] it is assumed that any polynomial time algorithm fails to factor a non-negligible fraction of integers that are the product of primes congruent to 3 modulo 4. With respect to such an integer N, squaring modulo N defines a permutation over the set of quadratic residues mod N, and therefore the intractability of factoring (such N's) yields the existence of a one-way permutation [R]. It was not known how to construct a one-way permutation or a pseudorandom generator assuming that factoring a non-negligible fraction of *all* the integers is intractable. In such a case modular squaring is a one-way function, but this function does not necessarily induce a permutation. Fortunately, modular squaring is a semi-regular function (see subsection 2.2.7), so we can apply our results.

Assumption IGF (*Intractability of the General Factoring Problem*): There exists a constant $c > 0$ such that for any probabilistic polynomial time algorithm A, and sufficiently large k

$$Prob\left(A(N) \text{ does not split } N \right) > k^{-c},$$

where $N \in_R \{0,1\}^k$.

Corollary 2.3.1: The IGF assumption implies the existence of pseudorandom generators.

Proof: Define the following function $f(N, x) = (N, x^2 \bmod N)$. Clearly, this function is semi-regular. The one-wayness of the function follows from IGF (using Rabin's argument [R]). Using an extension of Theorem 2.2.2 (see subsection 2.2.7) the corollary follows. □

Subsequently, J. (Cohen) Benaloh has found a way to construct a one-way permutation based on the IGF assumption. This yields an alternative proof of Corollary 2.3.1.

2.3.2. PRG Based on the Intractability of Decoding Random Linear Codes

One of the most outstanding open problems in coding theory is that of decoding random linear codes. Of particular interest are random linear codes with constant information rate which can correct a constant fraction of errors. An (n, k, d)-*linear code* is an k-by- n binary matrix in which the bit-by-bit XOR of any subset of the rows has at least d ones. The Gilbert-Varshamov bound for linear codes guarantees the existence of such a code provided that $k/n < 1 - H_2(d/n)$, where H_2 is the binary entropy function [McS, ch. 1, p. 34]. The same argument can be used to show (for every $\varepsilon > 0$) that if $k/n < 1 - H_2((1+\varepsilon) \cdot d/n)$, then almost all k-by- n binary matrices constitute (n, k, d)-linear codes.

We suggest the following function $f:\{0,1\}^* \to \{0,1\}^*$. Let C be an k-by- n binary matrix, $x \in \{0,1\}^k$, and $e \in E_t^n \subseteq \{0,1\}^n$ be a binary string with at most $t = \lfloor (d-1)/2 \rfloor$ ones, where d satisfies the condition of the Gilbert-Varshamov bound (see above). Clearly E_t^n can be uniformly sampled by an algorithm S running in time polynomial in n (i.e. $S:\{0,1\}^{poly(n)} \to E_t^n$). Let $r \in \{0,1\}^{poly(n)}$ be a string such that $S(r) \in E_t^n$. Then,

$$f(C, x, r) = (C, C(x) + S(r)),$$

where $C(x)$ is the codeword of x (i.e. $C(x)$ is the vector resulting by the matrix product xC). One can easily verify that f just defined is semi-regular (i.e. $f_C(x,r) = C(x) + S(r)$ is regular for all but a negligible fraction of the C's). The vector $xC + e$ ($e = S(r)$) represents a codeword perturbed by the error vector e.

Assumption IDLC (*Intractability of Decoding Random Linear Codes*): There exists a constant $c > 0$ such that for any probabilistic polynomial time algorithm A, and sufficiently large k

$$Prob(\ A(C, C(x) + e) \neq x\ )\ >\ k^{-c},$$

where C is a randomly selected k-by- n matrix, $x \in_R \{0,1\}^k$ and $e \in_R E_t^n$.

Now, either assumption IDLC is false which would be an earth-shaking result in coding theory or pseudorandom generators do exist.

Corollary 2.3.2: The IDLC assumption implies the existence of pseudorandom generators.

Proof: The one-wayness of the function f follows from IDLC. Using an extension of Theorem 2.2.2 (see subsection 2.2.7) the corollary follows. □

2.3.3. PRG Based on the Average Difficulty of Combinatorial Problems

Some combinatorial problems which are believed to be hard on the average can be used to construct a regular one-way function and hence be a basis for a pseudorandom generator. Consider, for example, the *Subset-Sum Problem*.

Input: Modulo M, $|M| = n$, and $n+1$ integers $a_0, a_1, \cdots, a_n$ of length n-bit each.

Question: Is there a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i \equiv a_0 \pmod{M}$

Conjecture: The above problem is hard on the average, when the a_i's and M are chosen uniformly in $[2^{n-1}, 2^n - 1]$.

Under the above conjecture, the function

$$f_S(a_1, a_2, \cdots, a_n, M, I) = (a_1, a_2, \cdots, a_n, M, (\sum_{i \in I} a_i \bmod M))$$

is both weakly-regular and one-way.

Chapter 3:

Sparse Pseudorandom Distributions

3.1. INTRODUCTION

Most of the previous work on pseudorandomness, as well as the results presented in chapter 2 of this thesis, have focused on the construction of (efficient) pseudorandom generators. The natural requirement that these generators work in polynomial time enforces this investigation to be based on some intractability assumptions.

In this chapter we study the notion of pseudorandomness when decoupled from the notion of efficient generation. This investigation is carried out using no unproven assumptions. The first question we address is the existence of non-trivial pseudorandom distributions. That is, pseudorandom distributions that are neither the uniform distribution nor statistically close to it (see Definition 3.2.5 bellow). Yao [Y] presents a particular example of such a distribution. Further properties of such distributions are developed here.

We prove the existence of *sparse* pseudorandom distributions. A distribution is called sparse if it is concentrated on a negligible part of the set of all strings of the same length. For example, given a positive constant $\delta<1$ we construct a probability distribution concentrated on $2^{\delta k}$ of the strings of length k which cannot be distinguished from the uniform distribution on the set of all k-bit strings (and hence is pseudorandom).

We show that sparse pseudorandom distributions can be uniformly generated by probabilistic algorithms (that run in non-polynomial time). On the other hand, we prove the existence of effectively generable pseudorandom distributions which cannot even be approximated by probabilistic polynomial-time algorithms. Moreover, we show the existence of *evasive* pseudorandom distributions which are not only sparse but also have the property that no polynomial-time algorithm may find an element in their support, except for a negligible probability.

An application of these results to the field of zero-knowledge interactive proofs is presented in chapter 5.

3.2. PRELIMINARIES

In chapter 2 we have defined the concept of a pseudorandom generator (definition 2.1.1). Here we define the general concept of a pseudorandom distribution. This definition is stated in asymptotical terms, so we shall not discuss single distributions but rather collections of

probability distributions called probability ensembles.

Definition 3.2.1: A *probability ensemble* Π is a collection of probability distributions $\{\pi_k\}_{k \in K}$, such that K is an infinite set of indices (nonnegative integers) and for every $k \in K$, π_k is a probability distribution on the set of (binary) strings of length k.

In particular, an ensemble $\{\pi_k\}_{k \in K}$ in which π_k is a uniform distribution on $\{0,1\}^k$ is called a *uniform ensemble*.

Next, we give a formal definition of a pseudorandom ensemble. This is done in terms of polynomial indistinguishability between ensembles.

Definition 3.2.2: Let $\Pi = \{\pi_k\}$ and $\Pi' = \{\pi'_k\}$ be two probability ensembles. Let T be a probabilistic polynomial time algorithm outputting 0 or 1 (T is called a *statistical test*). Denote by $p_T(k)$ the probability that T outputs 1 when fed with an input selected according to the distribution π_k. Similarly, $p'_T(k)$ is defined with respect to π'_k. The test T *distinguishes* between Π and Π' if and only if there exists a constant $c > 0$ and infinitely many k's such that $|p_T(k) - p'_T(k)| > k^{-c}$. The ensembles Π and Π' are called *polynomially indistinguishable* if there exists no polynomial-time statistical test that distinguish between them.

Definition 3.2.3: A probabilistic ensemble is called *pseudorandom* if it is polynomially indistinguishable from a uniform ensemble.

Remark 3.2.1: Some authors define pseudorandomness by requiring that pseudorandom ensembles be indistinguishable from uniform distributions even by *non-uniform* (polynomial) tests. We stress that the results (and proofs) in this chapter also hold for these stronger definitions.

We are interested in the question of whether non-trivial pseudorandom ensembles can be effectively sampled by means of probabilistic algorithms. The following definition capture the notion of 'samplability'.

Definition 3.2.4: A *sampling algorithm* is a probabilistic algorithm A that on input a string of the form 1^n, outputs a string of length n. The *probabilistic ensemble* $\Pi^A = \{\pi_n^A\}_n$ *induced by a sampling algorithm* A is defined by $\pi_n^A(y) = Prob(A(1^n) = y)$, where the probability is taken over the coin tosses of algorithm A. A *samplable* ensemble is a probabilistic ensemble induced by a sampling algorithm. If the sampling algorithm uses, on input 1^n, less than n random bits then we call the ensemble *strongly-samplable*.

Note that using the above terminology one can view pseudorandom generators as efficient strong-sampling algorithms (the seed is viewed as the random coins for the sampling algorithm).

We consider as trivial, pseudorandom ensembles that are "close" to a uniform ensemble. The meaning of "close" is formalized in the next definition.

Definition 3.2.5: Two probabilistic ensembles Π and Π' are *statistically close* if for any positive c and any sufficiently large n, $\sum_{x \in \{0,1\}^n} |\pi_n(x) - \pi_n'(x)| < n^{-c}$.

A special case of non-trivial pseudorandom ensembles are those ensembles we call "sparse".

Definition 3.2.6: A probabilistic ensemble is called *sparse* if (for sufficiently large n's) the support of π_n is a set of *negligible* size relative to the set $\{0,1\}^n$ (i.e for every $c>0$ and sufficiently large n, $|\{x \in \{0,1\}^n : \pi_n(x) > 0\}| < n^{-c}\, 2^n$).

Clearly, a sparse pseudorandom ensemble cannot be statistically close to a uniform ensemble.

Our proof of the existence of sparse pseudorandom distributions applies counting arguments. A central ingredient is the following inequality from Probability Theory due to W. Hoeffding [H].

Hoeffding Inequality: Suppose a urn contains u balls of which w are white and $u - w$ are black. Consider a random sample of s balls from the urn (without replacing any balls in the urn at any stage). Hoeffding inequality states that the proportion of white balls in the sample is close, with high probability, to its expected value, i.e. to the proportion of white balls in the urn. More precisely, let x be a random variable assuming the number of white balls in a random sample of size s. Then, for any ε, $0 \le \varepsilon \le 1$

$$Prob\left(\left| \frac{x}{s} - \frac{w}{u} \right| \ge \varepsilon \right) < 2\, e^{-2 s \varepsilon^2} \qquad (3.2.1)$$

This bound is oftenly used for the case of binomial distributions (i.e when drawn balls are replaced in the urn). The inequality for that case is due to H. Chernoff [C]. More general inequalities appear in Hoeffding's paper [H], as well as a proof that these bounds apply also for the case of samples without replacement.

3.3. THE EXISTENCE OF SPARSE PSEUDORANDOM ENSEMBLES

The main result in this section is the following Theorem.

Theorem 3.3.1: There exist strongly-samplable sparse pseudorandom ensembles.

In order to prove this theorem we present an ensemble of sparse distributions which are pseudorandom even against non-uniform distinguishers. These distributions assign equal probability to the elements in their support. We use the following definitions.

Definition 3.3.1: Let C be a (probabilistic) circuit with k inputs and a single output. We say that a set $S \subseteq \{0,1\}^k$ is $\varepsilon(k)$ – *distinguished* by the circuit C if

$$| p_C(S) - p_C(\{0,1\}^k)| \leq \varepsilon(k)$$

where $p_C(S)$ (resp. $p_C(\{0,1\}^k)$) denotes the probability that C outputs 1 when given elements of S (resp. $\{0,1\}^k$), chosen with uniform probability.

Definition 3.3.2: A set $S \subseteq \{0,1\}^k$ is called $(\tau(k), \varepsilon(k))$-*pseudorandom* if it is *not* $\varepsilon(k))$-distinguished by any circuit of size $\tau(k)$.

Note that a collection of uniform distributions on a sequence of sets $S_1, S_2, \ldots$ where each S_k is a $(\tau(k), \varepsilon(k))$-pseudorandom set, constitutes a pseudorandom ensemble, provided that both functions $\tau(k)$ and $\varepsilon^{-1}(k)$ are super-polynomial (i.e. grow faster than any polynomial). Our goal is to prove the existence of such a collection in which the ratio $|S_k|/2^k$ is negligibly small.

Remark 3.3.1: In the following we consider only deterministic circuits (tests). The ability to toss coins does not add power to non-uniform tests. Using a standard averaging argument one can show that whatever a probabilistic non-uniform distinguisher C can do, may be achieved by a deterministic circuit in which the "best coins" of C are incorporated.

The next Lemma measures the number of sets which are $\varepsilon(k)$-distinguished by a given circuit. Notice that this result does not depend on the circuit size.

Lemma 3.3.2: For any k-input Boolean circuit C, the probability that a random set $S \subseteq \{0,1\}^k$ of size N is $\varepsilon(k)$-distinguished by C is at most $2\, e^{-2N\varepsilon^2(k)}$.

Proof: Let $L_C(k)$ be the set $\{x \in \{0,1\}^k : C(x)=1\}$. Thus, $p_C(\{0,1\}^k) = \frac{|L_C(k)|}{2^k}$ and $p_C(S) = \frac{|S \cap L_C(k)|}{|S|}$.

Consider the set of strings of length k as a urn containing 2^k balls. Let those balls in $L_C(k)$ be painted white and the others black. The proportion of white balls in the urn is clearly $p_C(\{0,1\}^k)$, and the proportion of white balls in a sample S of N balls from the urn is $p_C(S)$. (We consider here a sample *without* replacement, i.e. sampled balls are not replaced in the urn).

Lemma 3.3.2 follows by using Hoeffding inequality (3.2.1)

$$Prob\left(|p_C(S) - p_C(\{0,1\}^k)| \geq \varepsilon(k)\right) < 2\, e^{-2N\varepsilon^2(k)}.$$

where the probability is taken over all the subsets $S \subseteq \{0,1\}^k$ of size N, with uniform probability. □

Corollary 3.3.3: For any positive integers k and N, and functions $\tau(\cdot)$ and $\varepsilon(\cdot)$, the proportion of subsets of $\{0,1\}^k$ of size N which are $(\tau(k), \varepsilon(k))$-pseudorandom is at least $1 - 2^{\tau^2(k) - 2N\varepsilon^2(k)}$.

Proof: The number of Boolean circuits of size $\tau(k)$ is at most $2^{\tau^2(k)}$. Therefore, using Lemma 3.3.2 we get that the proportion of sets $S \subseteq \{0,1\}^k$ of size N which are $\varepsilon(k)$-distinguished by any k-input Boolean circuit of size $\tau(k)$ is at most

$$2^{\tau^2(k)} \cdot 2\, e^{-2N\varepsilon^2(k)} \;<\; 2^{\tau^2(k)-2N\varepsilon^2(k)} \;.$$

□

The following Corollary states the existence of pseudorandom ensembles composed of uniform distributions with very sparse support.

Corollary 3.3.4: Let $k(n)$ be any subexponential function of n (i.e. $k(n)=2^{o(n)}$). [1] There exist super-polynomial functions $\tau(\cdot)$ and $\varepsilon^{-1}(\cdot)$, and a sequence of sets $S_1, S_2, \ldots$ such that S_n is a $(\tau(k(n)), \varepsilon(k(n)))$-pseudorandom subset of $\{0,1\}^{k(n)}$ and $|S_n| = 2^n$.

Proof: Using Corollary 3.3.3 we get that a $(\tau(k(n)), \varepsilon(k(n)))$-pseudorandom set $S \subseteq \{0,1\}^{k(n)}$ of size 2^n exists provided that

$$2^n \varepsilon^2(k(n)) > \tau^2(k(n)) \qquad (3.3.1)$$

It is easy to see that for any subexponential function $k(n)$ we can find super-polynomial functions $\varepsilon^{-1}(\cdot)$ and $\tau(\cdot)$ such that inequality (3.3.1) holds for each value of n. □

The following Lemma states that the sparse pseudorandom ensembles presented above are strongly-samplable. This proves Theorem 3.3.1.

Lemma 3.3.5: Let $k(n)$ be any subexponential function of n. There exist (non-polynomial) generators which expand random strings of length n into pseudorandom strings of length $k(n)$.

Proof: Let $\tau(\cdot)$ and $\varepsilon(\cdot)$ be as in Corollary 3.3.4. We construct a generator which on input a seed of length n finds the $(\tau(k(n)), \varepsilon(k(n)))$-pseudorandom set $S_n \subseteq \{0,1\}^{k(n)}$ whose existence is guaranteed by Corollary 3.3.4, and uses the n input bits in order to choose a random element from S_n. Clearly, the output of the generator is pseudorandom.

To see that the set S_n can be effectively found, note that it is effectively testable whether a given set S of size 2^n is $(\tau(k), \varepsilon(k))$-pseudorandom. This can be done by enumerating all the circuits of size $\tau(k)$ and computing for each circuit C the quantities $p_C(S)$ and $p_C(\{0,1\}^k)$. Thus, our generator will test all the possible sets $S \subseteq \{0,1\}^k$ of size 2^n until S_n is found. □

Remark 3.3.2: Inequality (3.3.1) implies a trade-off between the expansion function $k(n)$ and the size of the tests (circuits) resisted by the generated ensemble. The pseudorandom ensembles we construct may be "very" sparse, in the sense that the expansion function $k(n)$ can be chosen to be

[1] $o(n)$ denotes any function $f(n)$ such that $\lim_{n\to\infty} f(n)/n = 0$

very large (e.g. $2^{\sqrt{n}}$). On the other hand if we consider "moderate" expansion functions such as $k(n) = 2n$, we can resist rather powerful tests, e.g. circuits of size $2^{n/4}$.

Remark 3.3.3: The subexponential expansion, as allowed by our construction, is optimal since no generator exists which expands strings of length n into strings of length $k(n)=2^{O(n)}$. To see this, consider a circuit C of size $k(n)^{O(1)}$ ($=(2^n)^{O(1)}$) which incorporates the (at most) 2^n strings of length $k(n)$ output by the generator. On input a string of length $k(n)$ the circuit C looks up whether this input appears in the incorporated list of strings output by the generator. Clearly, this circuit C constitutes a (non-uniform) test (of size polynomial in $k(n)$) which distinguishes the output of this generator from the uniform distribution on $\{0,1\}^{k(n)}$.

Remark 3.3.4: The subexponential expansion implies that the supports of the resultant pseudorandom distributions are very sparse. More precisely, our construction implies the existence of generators which induce on strings of length k a support of size *slightly* super-polynomial (i.e. of size $k^{\omega(k)}$ for an arbitrary non-decreasing unbounded function $\omega(k)$). Thus, by wiring this support into a Boolean circuit, we are able to construct *non-uniform* generators of size slightly super-polynomial. (On input a seed s the circuit (generator) outputs the s-th element in this "pseudorandom" support). Let us point out that an improvement of this result, i.e. a proof of the existence of non-uniform pseudorandom generators of polynomial size, will imply that non-uniform-P $\neq$ non-uniform-NP !. This follows by considering the language $\{x \in \{0,1\}^k : x \textit{ is in the image of } G\}$, where G is a pseudorandom generator in non-uniform-P. Clearly, this language is in non-uniform-NP, but not in non-uniform-P, otherwise a decision procedure for it can be transformed into a test distinguishing the output of G from the uniform distribution on $\{0,1\}^k$.

Remark 3.3.5: The (uniform) complexity of the generators constructed in Lemma 3.3.5 is slightly super-exponential, i.e. $2^{k^{\omega(k)}}$, for unbounded $\omega(\cdot)$. (The complexity is, up to a polynomial factor, $2^{\tau^2(k)} \cdot (2^n + 2^k) \cdot \binom{2^k}{2^n}$, and 2^n is, as in Remark 3.3.4, slightly super-polynomial in k). We stress that the existence of pseudorandom generators running in exponential time, and with arbitrary polynomial expansion function, would have interesting consequences in Complexity Theory as BPP $\subseteq \cap_{\varepsilon>0}$ DTIME(2^{n^ε}) [Y, NW].

3.4. THE COMPLEXITY OF APPROXIMATING PSEUDORANDOM ENSEMBLES

In the previous section we have shown sparse pseudorandom ensembles which can be sampled by probabilistic algorithms running in super-exponential time. Whether is it possible to sample pseudorandom ensembles by polynomial-time algorithms or even exponential ones, cannot be proven today without using complexity assumptions. On the other hand, do such assumptions

guarantee that each samplable pseudorandom ensemble can be sampled by polynomial, or even exponential means? We give here a negative answer to this question, proving that for any complexity function $\phi(\cdot)$ there exists a samplable pseudorandom ensemble which cannot be sampled nor even "approximated" by algorithms in RTIME(ϕ). The notion of approximation is defined next.

Definition 3.4.1: A probabilistic ensemble Π is *approximated* by a sampling algorithm A if the ensemble Π^A induced by A is statistically close to Π.

The main result of this section is stated in the following theorem.

Theorem 3.4.1: For any complexity (constructive) function $\phi(\cdot)$, there is a strongly samplable pseudorandom ensemble that cannot be approximated by any algorithm whose running time is bounded by ϕ.

Proof: We say that two probability distributions π and π' on a set X are $\frac{1}{2}$ – *close* if

$$\sum_{x \in X} |\pi(x) - \pi'(x)| < \tfrac{1}{2} .$$

We say that a sampling algorithm M $\frac{1}{2}$-approximates a set $S \subseteq \{0,1\}^k$ if the probability distribution π_k^M induced by M on $\{0,1\}^k$ and the uniform distribution U_S on S are $\frac{1}{2}$-close.

We show that for any sampling algorithm M most subsets of $\{0,1\}^k$ of size 2^n are not $\frac{1}{2}$-approximated by M (for k sufficiently large with respect to n). This follows from the next Lemma.

Lemma 3.4.2: Let π be a probability distribution on $\{0,1\}^k$. The probability that π and U_S are $\frac{1}{2}$-close, for S randomly chosen over the subsets of $\{0,1\}^k$ of size 2^n, is less than $(1/2)^{k-n-1}$.

Proof: Notice that if two different sets S and T are $\frac{1}{2}$-close to π, then the two sets are close themselves. More precisely, we have that $\sum_{x \in \{0,1\}^k} |U_S(x) - \pi(x)| < \frac{1}{2}$ and $\sum_{x \in \{0,1\}^k} |U_T(x) - \pi(x)| < \frac{1}{2}$. Using the triangle inequality we conclude that $\sum_{x \in \{0,1\}^k} |U_S(x) - U_T(x)| < 1$. Denoting the last sum by σ and the symmetric difference of S and T by D, we have that $|D| \cdot \frac{1}{2^n} < \sigma < 1$ (this follows from the fact that U_S and U_T assign uniform probability to the 2^n elements of S and T, respectively). But this implies that $|D| < 2^n$, and then (using $|S| + |T| = |D| + 2 \cdot |S \cap T|$) we get $|S \cap T| > 2^n/2$.

Let T be a particular subset of $\{0,1\}^k$ of size 2^n which is $\frac{1}{2}$-close to π. From the above argument it follows that the collection of subsets of size 2^n which are $\frac{1}{2}$-close to π is included in the collection $\{S \subseteq \{0,1\}^k : |S| = 2^n, |S \cap T| > 2^n/2\}$. Thus, we are able to bound the probability that π is $\frac{1}{2}$-close to a random set S of size 2^n, by the probability of the following experiment. Fix a set $T \subseteq \{0,1\}^k$ of size 2^n, and take at random a set S of 2^n elements among all the strings in $\{0,1\}^k$. We are interested in the probability that $|S \cap T| > 2^n/2$. Clearly, the expectation of $|S \cap T|$ is $\frac{|S| \cdot |T|}{2^k}$.

Using Markov inequality for nonnegative random variables we have

$$Prob\left(|S\cap T|>\frac{2^n}{2}\right)\cdot\frac{2^n}{2}<\frac{|S|\cdot|T|}{2^k}$$

and then

$$Prob\,(|S\cap T|>2^n/2)<2/2^{k-n} \tag{3.4.1}$$

The lemma follows. ◊

We now extend the pseudorandom generator constructed in Lemma 3.3.5, in order to obtain a generator for a pseudorandom ensemble which is not approximated by any ϕ-time sampling algorithm. On input a string of length n, the generator proceeds as in Lemma 3.3.5. Once a $(\tau(k(n)),\varepsilon(k(n)))$-pseudorandom subset S_n is found, the generator checks whether S_n is $\frac{1}{2}$-approximated by some of the first n Turing machines, in some canonical enumeration, by running each of them as a sampling algorithm for $\phi(k(n))$ steps. Clearly, it is effectively testable whether a given machine M $\frac{1}{2}$-approximates a given set S. If the set S_n is $\frac{1}{2}$-approximated by some of these machines, it is discarded and the next $S\subseteq\{0,1\}^k$, $|S|=2^n$ is checked (first for pseudorandomness and then for approximation).

By Corollary 3.3.3 we have that for a suitable choice of the functions $\tau(\cdot)$ and $\varepsilon(\cdot)$ the probability that a set S is $(\tau(k(n)),\varepsilon(k(n)))$-pseudorandom is almost 1. On the other hand, the probability that a set S is $\frac{1}{2}$-approximated by n sampling machines is, using Lemma 3.4.2, less than $n/2^{k(n)-n-1}$. For suitable $k(\cdot)$, e.g. $k(n)\geq 2n$, this probability is negligible. Thus, we are guaranteed to find a set S_n which is $(\tau(k(n)),\varepsilon(k(n)))$-pseudorandom as well as not $\frac{1}{2}$-approximated by the first n sampling algorithms running ϕ-time. The resultant ensemble is as stated in the theorem. □

Remark 3.4.1: The result in Theorem 3.4.1 relies on the fact that the sampling algorithms we have run are uniform ones. Nevertheless, if we use Hoeffding inequality (3.2.1) to bound the left side in (3.4.1), we derive a much better bound, which implies that for any constant $\alpha<1$, there exist strongly-samplable pseudorandom ensembles that cannot be approximated by Boolean circuits of size $2^{\alpha n}$.

3.5. EVASIVE PSEUDORANDOM ENSEMBLES

In this section we prove the existence of pseudorandom ensembles which have the property that no polynomial-time sampling algorithm will output an element in their support, except for a negligible probability.

Definition 3.5.1: A probability ensemble $\Pi = \{\pi_k\}_{k \in K}$ is called *polynomial-time evasive* if for any polynomial-time sampling algorithm A, any constant c and sufficiently large k,

$$Prob\left(A(1^k) \in support(\pi_k) \right) < k^{-c}$$

($support(\pi_k)$ denotes the set $\{x \in \{0,1\}^k : \pi_k(x) > 0\}$).

Notice that evasiveness does not imply pseudorandomness. For example, any evasive ensemble remains evasive if we add to each string in the support a leading '0', while the resultant distributions are obviously not pseudorandom. On the other hand, an evasive pseudorandom ensemble is clearly sparse.

Following is the main result of this section. An interesting application of this result appears in chapter 5.

Theorem 3.5.1: There exist (strongly-samplable) polynomial-time evasive pseudorandom ensembles.

Proof: The proof outline is similar to the proof of Theorem 3.4.1. We again extend the generator of Lemma 3.3.5 by testing whether the $(\tau(k(n)), \varepsilon(k(n)))$-pseudorandom set S_n, found by that generator on input of length n, evades the first n Turing machines (run as polynomial-time sampling algorithms). We have to show that for each sampling algorithm M there is a small number of sets $S \subseteq \{0,1\}^k$ of size 2^n for which machine M outputs an element of S with significant probability. Throughout this proof we shall consider as "significant" a probability that is greater than $2^{3n}/2^k$. (This choice is motivated by an application of this Theorem in chapter 5. Any negligible portion suffices here. Thus, we are assuming $k \geq 4n$). We need the following technical Lemma.

Lemma 3.5.2: Let π be a fixed probability distribution on a set U of size K. For any $S \subseteq U$ denote $\pi(S) = \sum_{s \in S} \pi(s)$. Then

$$Prob(\pi(S) > \varepsilon) < \frac{N}{\varepsilon K}$$

where the probability is taken over all the sets $S \subseteq U$ of size N with uniform probability.

Proof: Consider a random sample of N *distinct* elements from the set U. Let $X_i, 1 \leq i \leq N$, be random variables so that X_i assumes the value $\pi(u)$ if the i-th element chosen in the sample is u. Define the random variable X to be the sum of the X_i's (i.e. $X = \sum_{i=1}^{N} X_i$).

Clearly, each X_i has expectation $1/K$ and then the expectation of X is N/K. Using Markov inequality for nonnegative random variables we get

$$Prob(\, X > \varepsilon\,) < \frac{E(X)}{\varepsilon} = \frac{N}{\varepsilon\, K}$$

proving the Lemma. ◊

Let π_k^M be the probability distribution induced by the sampling algorithm M on $\{0,1\}^k$. Consider a randomly chosen $S \subseteq \{0,1\}^k$ of size 2^n. Lemma 3.5.2 states that

$$Prob\left(\pi_k^M(S) > \frac{2^{3n}}{2^k}\right) < \frac{1}{2^{2n}} \tag{3.5.1}$$

Thus, we get that only $1/2^{2n}$ of the subsets S fail the evasivity test for a single machine. Running n such tests the portion of failing sets is at most $n/2^{2n}$. Therefore, there exists a set passing all the distinguishing and evasivity tests. (Actually, using Corollary 3.3.3, we get that most of the sets of size 2^n pass these tests). This completes the proof of the Theorem. □

Remark 3.5.1: Actually, we have proven that for any uniform time-complexity class **C**, there exist pseudorandom ensembles which evades any sampling algorithm of the class **C**. Notice that no restriction on the running time of the sampling machines is required. It is interesting to note that we cannot find ensembles evading the output of non-uniform circuits of polynomial-size, since for each set S there exists a circuit which outputs an element of S with probability 1. Thus, the results in this section imply the results of section 3.4 on unapproximability by uniform algorithms, but not the unapproximability by non-uniform circuits (see Remark 3.4.1).

Remark 3.5.2: In the previous remark we stressed the impossibility of the existence of ensembles evading the output of non-uniform machines with a polynomially long advice. Nevertheless, if we restrict the length of the advice our construction of Theorem 3.5.1 still works. Indeed, for the results in chapter 5, we need a slightly stronger result than the one stated in the above theorem. This application requires a pseudorandom ensemble that evades not only sampling algorithm receiving 1^k as the only input, but also algorithms having an additional input of length n (the parameters k and n are as defined above). The proof of Theorem 3.5.1 remains valid also in this case. This follows by observing that each such algorithm defines 2^n distributions, one for each possible input of length n. Thus, the n algorithms we run in the above proof contribute $n \cdot 2^n$ distributions. Using the above bound (3.5.1) we can guarantee the existence of sets S that evade any of these distributions.

3.6. NON-UNIFORM EVASIVE COLLECTIONS

In Remark 3.5.1 we have pointed out that evasive ensembles (in the sense of Definition 3.5.1) cannot evade non-uniform machines, since such a machine can always be supplied with an element in the ensemble support. For the application of pseudorandom and evasive distributions presented in chapter 5, we need a notion of "evasiveness" which also resists non-uniform

adversaries. In order to formalize this notion we use a collection of sets (or distributions) for each input length, rather than a single probability ensemble as in the uniform case.

Definition 3.6.1: A collection $S=\{S_1, S_2, \ldots, S_m\}$ of subsets of $\{0,1\}^k$ is called $(\tau(k), \varepsilon(k))$*-evasive* if for every (probabilistic) circuit C of size $\tau(k)$ with log m inputs and k outputs

$$Prob(C(i) \in S_i) < \varepsilon(k)$$

where the probability is taken over the random coins of C and i uniformly distributed over $\{1, \ldots, m\}$.

(The sets S_i can be viewed as supports of distributions on the set $\{0,1\}^k$).

Remark 3.6.1: In the above definition it is equivalent to consider deterministic circuits. Such a circuit may have wired in a sequence of "random coins" which maximizes the probability $Prob(C(i) \in S_i)$.

Definition 3.6.2: For $n=1,2,\ldots$ let $S^{(n)}$ be a $(\tau(n), \varepsilon(n))$-evasive collection of subsets of $\{0,1\}^{Q(n)}$, for a fixed polynomial Q. The sequence $S^{(1)}, S^{(2)}, \ldots$ is called *non-uniform polynomial-time evasive* (denoted *P/poly-evasive*) if $\tau(n)$ and $\varepsilon^{-1}(n)$ are both functions which grow faster than any polynomial.

That is, a sequence $S^{(1)}, S^{(2)}, \ldots$ is P/poly-evasive if any circuit of size polynomial in n, which gets a randomly selected index of one of the sets in $S^{(n)}$, cannot succeed to output an element in that set, except for a negligible probability.

In this section we show the existence (and samplability) of P/poly-evasive sequences. Furthermore, we prove the existence of such families composed of "pseudorandom" sets. We use the notion of a $(\tau(k), \varepsilon(k))$-pseudorandom set as defined in Definition 3.3.2. Recall that a collection of uniform distributions on the sets $S_1, S_2, \ldots$, where each S_k is a $(\tau(k), \varepsilon(k))$-pseudorandom set, constitutes a pseudorandom ensemble, provided that both functions $\tau(n)$ and $\varepsilon^{-1}(n)$ grow faster than any polynomial.

Following is the main result concerning P/poly-evasive and pseudorandom collections.

Theorem 3.6.1: There exists a P/poly-evasive collection $S^{(1)}, S^{(2)}, \ldots$ with parameters $Q(n)=4n$, $\tau(n)=\varepsilon^{-1}(n)=2^{n/4}$, such that for every n, $S^{(n)}=\{S_1^{(n)}, \ldots, S_{2^n}^{(n)}\}$, where each $S_i^{(n)}$ is a $(2^{n/4}, 2^{-n/4})$-pseudorandom set of cardinality 2^n. Furthermore, there exists a Turing machine which on input 1^n outputs the collection $S^{(n)}$.

Proof: Denote by $R^{(n)}$ the collection of sets $S \subseteq \{0,1\}^{4n}$ of cardinality 2^n which are $(2^{n/4}, 2^{-n/4})$-pseudorandom. We will show that there exists a positive probability to choose at random 2^n sets from $R^{(n)}$ which form a $(2^{n/4}, 2^{-n/4})$-evasive collection. This implies the existence of

such a collection.

Let $C^{(n)}$ denote the set of (deterministic) circuits of size $2^{n/4}$ having n inputs and $4n$ outputs. For a fixed $C \in C^{(n)}$ and a fixed $i, 1 \le i \le 2^n$ consider the probability $Prob(C(i) \in S)$, for S uniformly chosen over all subsets of $\{0,1\}^{4n}$ of size 2^n. Clearly,

$$Prob(C(i) \in S) = 1 - \frac{\binom{2^{4n}-1}{2^n}}{\binom{2^{4n}}{2^n}} = \frac{2^n}{2^{4n}} < \frac{1}{2^{2n}}$$

We shall call a set $S \subseteq \{0,1\}^{4n}$, $|S| = 2^n$, *C-bad* if there exists some $i, 1 \le i \le 2^n$ such that $C(i) \in S$. Fixing a circuit C, we have that for S uniformly chosen over all subsets of $\{0,1\}^{4n}$ of size 2^n,

$$Prob(\ S \text{ is } C-bad\ ) \le \sum_{i=1}^{2^n} Prob(C(i) \in S) < 2^n 2^{-2n} = 2^{-n}.$$

On the other hand, using Corollary 3.3.3 we get that the proportion of sets S which are $(2^{n/4}, 2^{-n/4})$-pseudorandom is at least $1 - 2^{-2^{n/4}}$. Therefore, for each circuit $C \in C^{(n)}$ the probability, hereafter denoted as ρ_C, to choose from $R^{(n)}$ a set S which is C-bad is

$$\rho_C = Prob(\ S \text{ is } C-bad \mid S \in R^{(n)}) \le \frac{2^{-n}}{1-2^{-2^{n/4}}}$$

We now proceed to compute the probability that for a fixed circuit $C \in C^{(n)}$, a collection of 2^n randomly chosen sets from $R^{(n)}$ contain a significant portion of C-bad sets. We define as "significant" a fraction $\rho_C + \delta_n$. (The quantity δ_n will be determined later). Let us introduce a random variable ρ assuming as its value the fraction of C-bad sets on a random sample of 2^n sets from $R^{(n)}$. Clearly, the expected value of ρ is ρ_C. Using Hoeffding inequality (see 3.2.1) we get

$$Prob(\ \rho \ge \rho_C + \delta_n) \le e^{-2\,2^n \delta_n^2}$$

Recall that we are interested to choose 2^n sets which are good for *all* the circuits $C \in C^{(n)}$. That is, we require that for any C the number of C-bad sets among the 2^n sets we choose is negligible. In order to bound the probability that 2^n randomly selected sets do not satisfy this condition, we multiply the above probability, computed for a single circuit, by the total number of circuits in $C^{(n)}$ which is $2^{\tau^2(n)} = 2^{2^{n/2}}$. Putting $\delta_n = \frac{2^{-n/4}}{\sqrt{2}}$ we get

$$2^{\tau^2(n)} \cdot e^{-2\,2^n \delta_n^2} = 2^{2^{n/2}} \cdot e^{-2\,2^n 2^{-\frac{n}{2}-1}} = 2^{2^{n/2}} \cdot e^{-2^{n/2}} < 1$$

We conclude that there exists a positive probability that 2^n sets $S_1, \ldots, S_{2^n}$ chosen at random from $R^{(n)}$ have the condition that for any circuit $C \in C^{(n)}$ the fraction of C-bad sets among $S_1, \ldots, S_{2^n}$ is

less than $\rho_C + \delta_n$. Therefore, such a collection of sets does exist.

It remains to show that the collection $S_1,\ldots,S_{2^n}$ satisfies the conditions stated in the Theorem. Clearly each set in the collection is $(2^{n/4}, 2^{-n/4})$-pseudorandom as it was selected from $R^{(n)}$. In order to show the evasiveness condition we bound, for any circuit $C \in C^{(n)}$, the probability $Prob(C(i) \in S_i)$, for i randomly chosen from $\{1,\ldots,m\}$. We have

$$Prob(C(i) \in S_i) = Prob(C(i) \in S_i | S_i \text{ is } C-bad\ ) \cdot Prob(S_i \text{ is } C-bad\ )$$

$$+ Prob(C(i) \in S_i | S_i \text{ is not } C-bad\ ) \cdot Prob(S_i \text{ is not } C-bad\ )$$

$$\leq 1 \cdot (\rho_C + \delta_n) \ + \ 0 \ \leq \ \frac{2^{-n}}{1-2^{-2^{n/4}}} \ + \ \frac{2^{-n/4}}{\sqrt{2}} \ < \ 2^{-n/4}$$

Thus, we have shown for every circuit C of size $2^{-n/4}$ that $Prob(C(i) \in S_i) < 2^{-n/4}$, and then $S_1,\ldots,S_{2^n}$ is a $(2^{n/4}, 2^{-n/4})$-evasive collection.

Such a collection can be generated by a Turing machine by considering all possible collections $\{S_1,\ldots,S_{2^n}\}$ and checking whether they evade all the circuits in the set $C^{(n)}$.

□

Chapter 4:

How to Predict Congruential Generators

4.1. INTRODUCTION

In this chapter we present our proof that congruential number generators are efficiently predictable.

A *number generator* is a deterministic algorithm that given a sequence of initial values, outputs an (infinite) sequence of numbers. For cryptographic applications a crucial property for the sequences generated is their *unpredictability*. That is, the next element generated should not be efficiently predictable, even given the entire past sequence. The efficiency of the predicting algorithm is measured both by the number of prediction mistakes and the time taken to compute each prediction. (A formal definition of an *efficient predictor* is given in section 4.2).

A pseudorandom number generator that has received much attention is the so called *linear congruential generator*, an algorithm that on input integers a, b, m, s_0 outputs a sequence $s_1, s_2, \cdots$ where

$$s_i \equiv a\, s_{i-1} + b \ (mod\ m)\,.$$

Knuth [K1] extensively studied some statistical properties of these generators.

Boyar [P] proved that linear congruential generators are efficiently predictable even when the coefficients and the modulus are unknown to the predictor. Later, Boyar [B] extended her own method, proving the predictability of a large family of number generators. She considered *general congruential generators* where the element s_i is computed as

$$s_i \equiv \sum_{j=1}^{k} \alpha_j\, \Phi_j(s_0, s_1, \cdots, s_{i-1}) \ (mod\ m) \qquad (4.1.1)$$

for integers m and α_j, and computable integer functions Φ_j, $j = 1, \ldots, k$. She showed that these sequences can be predicted, for some class of functions Φ_j, by a predictor knowing these functions and able to compute them, but not given the coefficients α_j or the modulus m. Boyar's method requires that the functions Φ_j have the *unique extrapolation property*. The functions $\Phi_1, \Phi_2, \cdots, \Phi_k$ have the *unique extrapolation property with length r*, if for every pair of generators working with the above set of functions, the same modulus m and the same initial values, if both generators coincide in the first r values generated, then they output the same infinite sequence. Note that these generators need not be identical (i.e. they may have different coefficients).

The number of mistakes made by Boyar's predictors depends on the extrapolation length. Therefore, her method yields efficient predictors provided that the functions Φ_j have a *small* extrapolation length. The linear congruential generator is an example of a generator having the extrapolation property (with length 2). Boyar proved this property also for two extensions of the linear congruential generator. Namely, the generators in which the element s_i satisfies the recurrence

$$s_i \equiv \alpha_1 s_{i-k} + \cdots + \alpha_k s_{i-1} \ (mod\ m)$$

and those for which

$$s_i \equiv \alpha_1 s_{i-1}^2 + \alpha_2 s_{i-1} + \alpha_3 \ (mod\ m)$$

The first case with length $k+1$, the second with length 3. She also conjectured the predictability of generators having a polynomial recurrence:

$$s_i \equiv p(s_{i-1}) (mod\ m)$$

for an unknown polynomial p of fixed (and known) degree.

A natural generalization of the above examples is a generator having a *multivariate polynomial recurrence,* that is a generator outputting a sequence $s_0, s_1, \ldots$ where

$$s_i \equiv P(s_{i-n}, \ldots, s_{i-1}) (mod\ m)$$

for a polynomial P in n variables. Note that for polynomials P of fixed degree and fixed n, the recurrence is a special case of the general congruential generators. Lagarias and Reeds [LR] showed that multivariate polynomial recurrences have the unique extrapolation property. Furthermore, for the case of a one-variable polynomial of degree d, they proved this property with length $d+1$, thus settling Boyar's conjecture concerning the efficient predictability of such generators. However, for the general case they did not give a bound on the length for which these recurrences are extrapolatable (neither a way to compute this length). Thus, unfortunately, Boyar's method does not seem to yield an efficient predicting algorithm for general multivariate polynomial recurrences (since it is not guaranteed to make a *small* number of mistakes but only a *finite* number, depending on the length of the extrapolation).

In this thesis we show how to predict any general congruential generator, i.e. any generator of the form (4.1.1). The only restriction on the functions Φ_j is that they are computable in polynomial time when working over the integers. This condition is necessary to guarantee the efficiency of our method. (The same is required in Boyar's method). Thus, we remove the necessity of the unique extrapolation property, and extend the predictability results to a very large class of number generators. In particular, we show that multivariate polynomial recurrence generators *are efficiently predictable*.

Our predicting technique is based on ideas from Boyar's method, but our approach to the prediction problem is somewhat different. Boyar's method tries to simulate the generator by "discovering" its secrets: the modulus m and the coefficients α_j that the generator works with.

Instead, our algorithm uses only the knowledge that these coefficients exist, but does not try to find them. Some algebraic techniques introduced by Boyar when computing over the integers, are extended by us to work also when computing over the ring of integers modulo m.

Our prediction results concern number generators outputting all the bits of the generated numbers, and does not apply to generators that output only parts of the numbers generated. Recent works treat the problem of predicting linear congruential generators which output only parts of the numbers generated [FHKLS, K2, S].

4.2. DEFINITIONS AND NOTATION

Definition 4.2.1: A *number generator* is an algorithm that given n_0 integer numbers, called the *initial values* and denoted $s_{-n_0},\cdots,s_{-1}$, outputs an infinite sequence of integers $s_0, s_1,\ldots$ where each element s_i is computed deterministicly from the previous elements, including the initial values.

For example, a generator of the form $s_i \equiv \alpha_1 s_{i-k}+\cdots+\alpha_k s_{i-1}\ (mod\ m)$ requires a set of k initial values to begin computing the first elements $s_0, s_1,\cdots$ of the sequence. Thus, for this example $n_0 = k$.

Definition 4.2.2: A *(general) congruential generator* is a number generator for which the i-th element of the sequence is a $\{0,\ldots,m-1\}$-valued number computed by the congruence

$$s_i \equiv \sum_{j=1}^{k} \alpha_j \Phi_j(s_{-n_0},\cdots,s_{-1},s_0,\cdots,s_{i-1})\ (mod\ m)$$

where α_j and m are arbitrary integers and $\Phi_j, 1\le j\le k$, is a computable integer function. For a given set of k functions $\Phi=\{\Phi_1,\Phi_2,\ldots,\Phi_k\}$ a congruential generator working with these functions (and arbitrary coefficients and modulus) will be called a Φ-*generator*.

Example: Consider a number generator which outputs a sequence defined by a multivariate polynomial recurrence, i.e. $s_i \equiv P(s_{i-n},\ldots,s_{i-1})(mod\ m)$, where P is a polynomial in n variables and fixed degree d. Such a generator is a Φ-generator in which each function Φ_j represents a monomial in P and α_j are the corresponding coefficients. In this case we have $k=\binom{n+d}{d}$, and the functions (monomials) Φ_j are applied to the last n elements in the sequence.

Note that in the above general definition, the functions Φ_j work on sequences of elements, so the number of arguments for these functions may be variable. Some matrix notation will be more convenient.

Notation: $s(i)$ will denote the *vector* of elements (including the initial values) until the element s_i, i.e.

$$s(i) = (s_{-n_0}, \cdots, s_{-1}, s_0, \cdots, s_i) \quad i = -1, 0, 1, 2 \cdots$$

Thus, $\Phi_j(s_{-n_0}, \cdots, s_{-1}, s_0, \cdots, s_{i-1})$ will be written as $\Phi_j(s(i-1))$.
Let α denote the vector $(\alpha_1, \alpha_2, \cdots, \alpha_k)$ and $B_i, i \geq 0$, denote the column vector

$$B_i = \begin{bmatrix} \Phi_1(s(i-1)) \\ \Phi_2(s(i-1)) \\ \vdots \\ \Phi_k(s(i-1)) \end{bmatrix}$$

Then we can rewrite the Φ-generator's recurrence as

$$s_i \equiv \alpha \cdot \mathrm{B_i} \pmod{\mathrm{m}} \tag{4.2.1}$$

Here, and in the sequel, $\cdot$ denotes matrix multiplication.
Finally, $B(i)$ will denote the matrix

$$B(i) = \begin{bmatrix} B_0 \; B_1 \cdots B_i \end{bmatrix}$$

For complexity considerations we refer to the size of the prediction problem as given by the size of the modulus m and the number k of coefficients the generator actually works with. (Note that the coefficients as well as the elements output by the generator have size at most $\log m$). We consider as *efficient*, congruential generators for which the functions $\Phi_j, 1 \leq j \leq k$, are computable in time polynomial in $\log m$ and k. Also the efficiency of a predictor will be measured in terms of these parameters, which can be seen as measuring the amount of information hidden from the predictor.

We shall be concerned with the complexity of the functions Φ_j when acting on the vectors $s(i)$, but computed over the integers (and not reduced modulo m). This will be referred to as the *non-reduced complexity* of the functions Φ_j. The performance of our predicting algorithm will depend on this complexity.

Definition 4.2.3: Φ-generators having non-reduced time-complexity polynomial in $\log m$ and k are called *non-reduced polynomial-time* Φ*-generators*.

Next we define the basic concept, throughout this chapter, of a *predictor*:

Definition 4.2.4: A *predictor* for a Φ-generator is an algorithm that interacts with the Φ-generator in the following way. The predictor gets as input the initial values that the generator is working

with. For $i=0,1,2,\ldots$ the predictor outputs its prediction for the element s_i and the generator responds with the true value of s_i.

An *efficient predictor for a family of congruential generators* is an algorithm which given a set of k functions $\Phi=\{\Phi_1,\Phi_2,\ldots,\Phi_k\}$ corresponding to a Φ-generator in the family, behaves as a predictor for this Φ-generator, and there exist polynomials P and Q for which

1) the computation time of every prediction is bounded by $P(k,\log m)$

2) the number of prediction mistakes is bounded by $Q(k,\log m)$

In the above definition we may consider the functions Φ_j as given to the algorithm by means of "black-boxs" or oracles to these functions. In this case the output of such an oracle is the integer value of the function before it is reduced according to the secret modulus.

Observe that when computing its prediction for s_i the predictor has seen the entire segment of the sequence before s_i, and the initial values. The only secret information kept by the generator is the coefficients and the modulus. If the predictor is not given the initial values then our method cannot be applied to *arbitrary* Φ-generators. However, in typical cases (including the multivariate polynomial recurrence) generators have recurrences depending only on the last n_0 elements, for some constant n_0. In this case the predictor may consider the first n_0 elements generated as initial values, and begin predicting after the generator outputs them.

4.3. THE PREDICTING ALGORITHM

The predictor tries to infer the element s_i from knowledge of all the previous elements of the sequence, including the initial values. It does not know the modulus m the generator is working with, so it uses different estimates for this m. Its first estimate is $\hat{m}=\infty$, i.e. the predictor begins by computing over the integers. After some portion of the sequence is revealed, and taking advantage of possible prediction mistakes, a new (finite) estimate $\hat{m}_0$ for m is computed. Later on, new values for $\hat{m}$ are computed in such a way that each $\hat{m}$ is a (non-trivial) divisor of the former estimate, and all are multiples of the actual m. Eventually $\hat{m}$ may reach the true value of m. (For degenerate cases, like a generator producing a constant sequence, it may happen that m will never be reached but this will not effect the prediction capabilities of the algorithm).

We shall divide the predicting algorithm into two *stages* . The first stage is when working over the integers, i.e. $\hat{m}=\infty$. The second one is after the first finite estimate $\hat{m}_0$ was computed. The distinction between these two stages is not essential, but some technical reasons make it convenient. In fact, the algorithm is very similar for both stages.

The idea behind the algorithm is to find linear dependencies among the columns of the matrix $B(i)$ and to use these dependencies in making the prediction of the next element s_i. More specificly, we try to find a representation of B_i as a linear combination (modulo the current $\hat{m}$) of the previous B_j's (that are known to the predictor at this time). If such a combination exists, we apply it to the previous elements in the sequence (i.e. previous s_j's) to obtain our *prediction* for s_i. If not correct, we made a mistake but gain information that allows us to refine the modulus $\hat{m}$. A combination as above will not exist if B_i is independent of the previous columns. We show that under a *suitable definition of independence*, the number of possible *independent* B_i's cannot be *too large*. Therefore only a *small* number of mistakes is possible, allowing us to prove the efficiency of the predictor.

The number of mistakes made by the predictor, until it is able to refine the current $\hat{m}$, will be bounded by a polynomial in the size of this $\hat{m}$. Also the total number of distinct moduli $\hat{m}$ computed during the algorithm is bounded by the size of the first (finite) $\hat{m}_0$. Thus, the total number of possible mistakes is polynomial in this size, which in turn is determined by the length of the output of the non-reduced functions Φ_j. This is the reason for which the non-reduced complexity of these functions is required to be polynomial in the size of the true m and k. In this case the total number of mistakes made by the predictor will also be polynomial in these parameters. The same is true for the computation time of every prediction.

The algorithm presented here is closely related to Boyar's [B]. Our first stage is exactly the same as the first stage there. That is, the two algorithms begin by computing a multiple of the modulus m. Once this is accomplished, Boyar's strategy is to find a set of coefficients $\{\alpha'_j\}_{j=1}^{k}$ and a sequence of moduli $\hat{m}$ which are refined during the algorithm until no more mistakes are made. For proving the correctness and efficiency of her predictor, it is required that the generator satisfies the *unique extrapolation property* (mentioned in the Introduction). In our work, we do not try to find the coefficients. Instead, we extend the ideas of the first stage, and apply them also in the second stage. In this way the need for an extrapolation property is avoided, allowing the extensions of the predictability results.

4.3.1 First Stage

Let us describe how the predictor computes its prediction for s_i. At this point the predictor knows the whole sequence before s_i, i.e. $s(i-1)$, and so far it has failed to compute a finite multiple of the modulus m, so it is still working over the integers. In fact, the predictor is able at this point to compute all the vectors $B_0, B_1, \cdots, B_i$, since they depend only on $s(i-1)$. Moreover, our predictor keeps at this point, a submatrix of $B(i-1)$, denoted by $\overline{B(i-1)}$, of linearly independent (over the rationals) columns. (For every i, when predicting the element s_i, the predictor checks if the column B_i is independent of the previous ones. If this is the case then B_i is added to

$\overline{B(i-1)}$ to form $\overline{B(i)}$). Finally, let us denote by $\overline{s(i-1)}$ the corresponding *subvector* of $s(i-1)$, having the entries indexed with the same indices appearing in $\overline{B(i-1)}$.

Prediction of s_i in the first stage:

The predictor begins by computing the (column) vector B_i. Then, it solves, **over the rationals**, the system of equations

$$\overline{B(i-1)} \cdot \mathrm{x} = \mathrm{B_i}$$

If no solution exists, B_i is independent of the columns in $\overline{B(i-1)}$ so it sets

$$\overline{B(i)} = \left[\overline{B(i-1)} \;\; B_i \right]$$

and it fails to predict s_i.

If a solution exists, let c denote the solution (vector) computed by the predictor. The prediction for s_i, denoted $\hat{s}_i$, will be

$$\hat{s}_i = \overline{s(i-1)} \cdot \mathrm{c}$$

The predictor, once having received the true value for s_i, checks whether this prediction is correct or not (observe that the prediction $\hat{s}_i$ as computed above may not even be an integer). If correct, it has succeeded and goes on predicting s_{i+1}. If not, i.e. $\hat{s}_i \neq s_i$, the predictor has made a mistake, but now it is able to compute $\hat{m}_0 \neq \infty$, the first multiple of the modulus m, as follows. Let l be the number of columns in matrix $\overline{B(i-1)}$ and let the solution c be

$$c = \begin{bmatrix} c_1/d_1 \\ c_2/d_2 \\ \vdots \\ c_l/d_l \end{bmatrix}$$

Now, let d denote the least common multiple of the dominators in these fractions, i.e. $d = lcm(d_1, \cdots, d_l)$. The value of $\hat{m}_0$ is computed as follows

$$\hat{m}_0 = \left| d\hat{s}_i - ds_i \right| .$$

Observe that $\hat{m}_0$ is an integer, even if $\hat{s}_i$ is not. Moreover this integer is a multiple of the true modulus m the generator is working with (see Lemma 4.3.1 below).

Once $\hat{m}_0$ is computed, the predictor can begin working modulo this $\hat{m}_0$. So the first stage of the algorithm is terminated and it goes on into the second one.

The main facts concerning the performance of the predicting algorithm during the first stage are summarized in the next Lemma.

Lemma 4.3.1:

a) The number $\hat{m}_0$ computed at the end of the first stage is a nonzero multiple of the modulus m.

b) The number of mistakes made by the predictor in the first stage is at most $k+1$.

c) For non-reduced polynomial time Φ-generators, the prediction time for each s_i during the first stage is polynomial in log m and k.

d) For non-reduced polynomial time Φ-generators, the size of $\hat{m}_0$ is polynomial in log m and k. More precisely, let M be an upper bound on the output of each of the functions Φ_j, $j=1,\ldots,k$, working on $\{0,\ldots,m-1\}$-valued integers. Then, $\hat{m}_0 \le (k+1)\, k^{k/2}\, m\, M^k$.

Proof:

a) From the definition of the generator we have the congruence $s_j \equiv \alpha \cdot \mathrm{B_j} \pmod{m}$ for all $j \ge 0$, therefore

$$\overline{s(i-1)} \equiv \alpha \cdot \overline{\mathrm{B(i-1)}} \pmod{m} \qquad (4.3.1)$$

Thus,

$d\hat{s}_i = d\ \overline{s(i-1)} \cdot \mathrm{c}$ (by definition of $\hat{s}_i$)

$\equiv d\alpha \cdot \overline{\mathrm{B(i-1)}} \cdot \mathrm{c} \pmod{m}$ (by (4.3.1))

$= d\alpha \cdot \mathrm{B_i}$ (c is a solution to $\overline{B(i-1)} \cdot \mathrm{x} = \mathrm{B_i}$)

$\equiv d\, s_i \ (mod\ m)$ (By definition of s_i (4.2.1))

So we have shown that $d\hat{s}_i \equiv ds_i\ (mod\ m)$. Observe that it cannot be the case that $d\hat{s}_i = ds_i$, because this implies $\hat{s}_i = s_i$, contradicting the incorrectness of the prediction. Thus, we have proved that $\hat{m}_0 = |\, d\hat{s}_i - ds_i\,|$ is indeed a nonzero multiple of m.

b) The possible mistakes in the first stage are when a rational solution to the system of equations $\overline{B(i-1)} \cdot \mathrm{x} = \mathrm{B_i}$ does not exist, or when such a solution exists but our prediction is incorrect. The last case will happen only once because after that occurs the predictor goes into the second stage. The first case cannot occur "too much". Observe that the matrices $B(j)$ have k rows, thus the maximal number of independent columns (over the rationals) is at most k. So the maximal number of mistakes made by the predictor in the first stage is $k+1$.

c) The computation time for the prediction of s_i is essentially given by the time spent computing B_i and solving the above equations. The functions Φ_j are computable in time polynomial in log m and k, so the computation of the vector B_i is also polynomial in log m and k. The complexity of solving the system of equations, over the rationals, is polynomial in k and in the size of the entries of $\overline{B(i-1)}$ and B_i (see [Ed], [Sch, Ch. 3]). These entries are determined by the output of the (non-reduced) functions Φ_j, and therefore their size is bounded by a polynomial in log m and k. Thus, the total complexity of the prediction step is polynomial in log m and k, as required.

d) As pointed out in the proof of claim c), a solution to the system of equations in the algorithm, can be found in time bounded polynomially in log m and k. In particular this guarantees that the *size* of the solution will be polynomial in log m and k. (By size we mean the size of the denominators and numerators in the entries of the solution vector.) Clearly, by the definition of $\hat{m}_0$, the polynomiality of the size of the solution c implies that the size of $\hat{m}_0$ is itself polynomial in log m and k.

The explicit bound on $\hat{m}_0$ can be derived as follows. Using Cramer's rule we get that the solution c to the system $\overline{B(i-1)} \cdot \mathrm{x} = \mathrm{B_i}$, can be represented as $c=(c_1/d,\ldots,c_l/d)$ where each c_j and d are determinants of l by l submatrices in the above system of equations. Let D be the maximal possible value of a determinant of such a matrix. We have that $d\,\hat{s}_i = d\,\overline{s(i-1)}\,c \le l\,m\,D$ (here m is a bound on $\overline{s(i-1)}$ entries) and $d\,s_i \le m\,D$, then $\hat{m}_0 = |\,d\hat{s}_i - ds_i\,| \le (l+1)\,m\,D$. In order to bound D we use Haddamard's inequality which states that each n by n matrix $A=(a_{ij})$ satisfies $det(A) \le \prod_{i=1}^{n} (\sum_{j=1}^{n} a_{ij}^2)^{1/2}$. In our case the matrices are of order l by l, and the entries to the system are bounded by M (the bound on Φ_j output). Thus, $D \le \prod_{i=1}^{l} (\sum_{j=1}^{l} M^2)^{1/2} = (l\,M^2)^{l/2}$, and we get

$$\hat{m}_0 \le (l+1)\,m\,D \le (l+1)\,m\,(l\,M^2)^{l/2} \le (k+1)\,k^{k/2}\,m\,M^k$$

The last inequality follows since $l \le k$. □

4.3.2 Second Stage

After having computed $\hat{m}_0$, the first multiple of m, we proceed to predict the next elements of the sequence, but now working modulo a finite $\hat{m}$. The prediction step is very similar to the one described for the first stage. The differences are those that arise from the fact that the computations are modulo an integer. In particular the equations to be solved will not be over a field (in the first stage it was over the rationals), but rather over the ring of residues modulo $\hat{m}$. Let us denote the ring of residues modulo n by Z_n. In the following definition we extend the concept of linear dependence to these rings.

Definition 4.3.1: Let $v_1, v_2, \ldots, v_l$ be a sequence of l vectors with k entries from Z_n. We say that this sequence is *weakly linearly dependent mod n* if $v_1 = 0$ or there exists an index $i, 2 \le i \le l$, and elements $c_1, c_2, \ldots, c_{i-1} \in Z_n$, such that $v_i \equiv c_1 v_1 + c_2 v_2 + \cdots + c_{i-1} v_{i-1} \ (mod\ n)$. Otherwise, we say that the sequence is *weakly linearly independent*.

Note that the order here is important. Unlike the case in the traditional definition over a field, in the above definition it is *not* equivalent to say that *some* vector in the set can be written as a linear combination of the *others*. Another important difference is that it is not true in general, that $k+1$ vectors of k components over Z_n must contain a dependent vector. Fortunately, a slightly weaker statement does hold.

Theorem 4.3.2: Let $v_1, v_2, \ldots, v_l$ be a sequence of k-dimensional vectors over Z_n. If the sequence is weakly linearly independent mod n, then $l \le k \log_q n$, where q is the smallest prime dividing n.

Proof: Let $v_1, v_2, \ldots, v_l$ be a sequence of l vectors from Z_n^k, and suppose this sequence is weakly linearly independent mod n. Consider the set

$$V = \{ \sum_{i=1}^{l} c_i v_i \ (mod\ n) : c_i \in \{0, 1, \cdots, q-1\} \}$$

We shall show that this set contains q^l different vectors. Equivalently, we show that no two (different) combinations in V yield the same vector.

Claim: For every $c_i, c_i' \in \{0, 1, \cdots, q-1\}, 1 \le i \le l$, if $\sum_{i=1}^{l} c_i v_i \equiv \sum_{i=1}^{l} c_i' v_i \ (mod\ m)$ then $c_i = c_i'$ for $i = 1, 2, \ldots, l$.

Suppose this is not true. Then we have $\sum_{i=1}^{l} (c_i - c_i') v_i \equiv 0 \ (mod\ n)$. Denote $c_i - c_i'$ by d_i. Let t be the maximal index for which $d_t \ne 0$. This number d_t satisfies $-q < d_t < q$, so it has an inverse modulo n (recall that q is the least prime divisor of n), denoted d_t^{-1}. It follows that $v_t \equiv \sum_{i=1}^{t-1} - d_t^{-1} d_i v_i (mod\ n)$ contradicting the independence of v_t, and thus proving the claim.

Hence, $|V| = q^l$ and therefore

$$q^l = |V| \le |Z_n^k| = n^k$$

which implies $l \le k \log_q n$, proving the Theorem. □

With the above definition of independence in mind, we can define the matrix $\overline{B(i)}$ as a submatrix of $B(i)$, in which the (sequence of) columns are weakly linearly independent *mod* $\hat{m}$.

Note that $\hat{m}$ will have distinct values during the algorithm, so when writing $\overline{B(i)}$ we shall refer to its value modulo the current $\hat{m}$.

Prediction of s_i in the second stage:

Let us describe the prediction step for s_i when working modulo $\hat{m}$. In fact, all we need is to point out the differences with the process described for the first stage.

As before, we begin by computing the vector B_i (now reduced modulo $\hat{m}$), and solving the system of equations

$$\overline{B(i-1)} \cdot \mathrm{x} \equiv \mathrm{B_i} \pmod{\hat{m}}$$

We stress that this time we are looking for a solution over $Z_{\hat{m}}$. In case a solution does not exist, we fail to predict, exactly as in the previous case. As before, the vector $B_i(mod\ \hat{m})$ is added to $\overline{B(i-1)}$ to form the matrix $\overline{B(i)}$. If a solution does exist, we output our prediction, computed as before, but the result is reduced mod $\hat{m}$. Namely, we set $\hat{s}_i = \overline{s(i-1)} \cdot \mathrm{c} \pmod{\hat{m}}$, where c is a solution to the above system of modular equations. If the prediction is correct, we proceed to predict the next element s_{i+1}. If not, we take advantage of this error to update $\hat{m}$. This is done by computing

$$\acute{m} = gcd(\hat{m}, \hat{s}_i - s_i)$$

This $\acute{m}$ will be the new $\hat{m}$ we shall work with in the coming predictions.

To see that the prediction algorithm as described here, is indeed an *efficient predictor*, we have to prove the following facts summarized in Lemma 4.3.3. (Lemma 4.3.3 is analogous to Lemma 4.3.1 for the second stage).

Lemma 4.3.3: The following claims hold for the above predictor when predicting a non-reduced polynomial time Φ-generator.

a) The number $\acute{m}$ computed above is a nontrivial divisor of $\hat{m}$ and a multiple of the modulus m.

b) Let $\hat{m}_0$ be the modulus computed at the end of the first stage. The total number of mistakes made by the predictor during the second stage is bounded by $(k+1)\log \hat{m}_0$, and then polynomial in $\log m$ and k.

c) The prediction time for each s_i during the second stage is polynomial in $\log m$ and k.

Proof:

a) Recall that $\acute{m} = gcd(\hat{m}, \hat{s}_i - s_i)$, so it is a divisor of $\hat{m}$. It is a nontrivial divisor because $\hat{s}_i$ and s_i are reduced mod $\hat{m}$ and m respectively, and then their difference is strictly less than $\hat{m}$. It cannot be zero because $\hat{s}_i \neq s_i$, as follows from the incorrectness of the prediction. The proof that $\acute{m}$ is a

multiple of m is similar to that of claim a) of Lemma 4.3.1. It is sufficient to show that $\hat{s}_i - s_i$ is a multiple of m, since $\hat{m}$ is itself a multiple of m. We show this by proving $\hat{s}_i \equiv s_i \,(mod\ m)$:

$$\hat{s}_i \equiv \overline{s(i-1)} \cdot \mathrm{c} \,(\mathrm{mod}\ \hat{m}) \qquad \text{(by definition of } \hat{s}_i)$$

$$\equiv \alpha \cdot \overline{\mathrm{B(i-1)}} \cdot \mathrm{c}\ (\mathrm{mod\ m}) \qquad \text{(by (4.3.1))}$$

$$\equiv \alpha \cdot \mathrm{B_i}\,(\mathrm{mod}\ \hat{m}) \qquad (c \text{ is a solution to } \overline{B(i-1)} \cdot \mathrm{x} \equiv \mathrm{B_i}\,(\mathrm{mod}\ \hat{m}))$$

$$\equiv s_i\ (mod\ m) \qquad \text{(By definition of } s_i \text{ (4.2.1))}$$

As m divides $\hat{m}$, claim a) follows.

b) The possible mistakes during the second stage are of two types. Mistakes of the first type happen when a solution to the above congruential equations does not exist. This implies the independence modulo the current $\hat{m}$ of the corresponding B_i. In fact, this B_i is also independent $mod\ \hat{m}_0$. This follows from the property that every $\hat{m}$ is a divisor of $\hat{m}_0$. By Theorem 4.3.2, we have that the number of weakly linearly independent vectors $mod\ \hat{m}_0$ is at most $k \log \hat{m}_0$. Therefore the number of mistakes by lack of a solution is bounded by this quantity too. The second type of mistake is when a solution exists but the computed prediction is incorrect. Such a mistake can occur only once per $\hat{m}$. After it occurs, a new $\hat{m}$ is computed. Thus, the total number of such mistakes is as the number of different $\hat{m}$'s computed during the algorithm. These $\hat{m}$'s form a decreasing sequence of positive integers in which every element is a divisor of the previous one. The first (i.e. largest) element is $\hat{m}_0$ and then the length of this sequence is at most $\log \hat{m}_0$. Consequently, the total number of mistakes during the second stage is at most $(k+1) \log \hat{m}_0$, and by Lemma 4.3.1 claim d) this number is polynomial in $\log m$ and k.

c) By our assumption of the polynomiality of the functions Φ_j when working on the vectors $s(i)$, it is clear that the computation of each B_i $(mod\ \hat{m})$, takes time that is polynomial in $\log m$ and k. We only need to show that a solution to $\overline{B(i-1)} \cdot \mathrm{x} \equiv \mathrm{B_i}\ (\mathrm{mod}\ \hat{m})$ can be computed in time polynomial in $\log m$ and k. A simple method for the solution of a system of linear congruences like the above, is described in [BS] (and [B]). This method is based on the computation of the *Smith Normal Form* of the coefficients matrix in the system. This special matrix and the related transformation matrices, can be computed in polynomial time, using an algorithm of [KB]. Thus, finding a solution to the above system (or deciding that none exists) can be accomplished in time polynomial in $\log m$ and k. Therefore the whole prediction step is polynomial in these parameters. □

Combining Lemmas 4.3.1 and 4.3.3 we get

Theorem 4.3.4: The predicting algorithm described above is an efficient predictor for non-reduced polynomial-time Φ-generators. The number of prediction mistakes is at most $(k+1)(log\ \hat{m}_0 + 1) = O(\ k^2 \log(k\ m\ M)\ )$, where $\hat{m}_0$ is the first finite modulus computed by the algorithm, and M is an upper bound on the output of each of the functions Φ_j, $j = 1, \ldots, k$, working over integers in the set $\{0, \ldots, m-1\}$.

As a special case we get

Corollary 4.3.5: Every multivariate polynomial recurrence generator is efficiently predictable. The number of prediction mistakes for a polynomial recurrence in n variables and degree d is bounded by $O(\ k^2 \log(k\ m^d)\ )$, where $k = \binom{n+d}{d}$.

Proof: A multivariate polynomial recurrence is a special case of a Φ-generator with $M < m^d$, as each monomial is of degree at most d and it is computed on integers less than m. Therefore, by Lemma 4.3.1 d) we get $\hat{m}_0 < (k+1)\, k^{k/2}\, m^{dk+1}$. The number k of coefficients is as the number of possible monomials in such a polynomial recurrence which is $\binom{n+d}{d}$. The bound on the number of mistakes follows by substituting these parameters in the general bound of Theorem 4.3.4. □

Remark 4.3.1: Notice that the number k of coefficients equals the number of possible monomials in the polynomial. For general polynomials in n variables and of degree d, this number is $\binom{n+d}{d}$. Nevertheless, if we consider special recurrences in which not every monomial is possible, e.g. $s_i \equiv \alpha_1\, s_{i-n}^2 + \cdots + \alpha_n\, s_{i-1}^2\ (mod\ m)$, then the number k may be much smaller, and then a better bound on the number of mistakes for such cases is derived.

4.4. VECTOR-VALUED RECURRENCES

The most interesting subclass of Φ-generators is the class of multivariate polynomial recurrence generators mentioned in previous sections. Lagarias and Reeds [LR] studied a more general case of polynomial recurrences in which a sequence of n-dimensional vectors over Z_m is generated, rather than a sequence of Z_m elements as in our case. These vector-valued polynomial recurrences have the form

$$\bar{s}_i \equiv (\ P_1(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n})(mod\ m), \ldots, P_n(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n})(mod\ m)\ )$$

where each $P_l, 1 \le l \le n$, is a polynomial in n variables and of maximal degree d. Clearly, these recurrences extend the single-valued case, since for any multivariate polynomial P which generates a sequence $\{s_i\}_{i=0}^{\infty}$ of Z_m elements, one can consider the sequence of vectors $\bar{s}_i = (s_i, s_{i-1}, \ldots, s_{i-n+1})$ where $\bar{s}_i = (P(s_{i-1}, \ldots, s_{i-n})(mod\ m), s_{i-1}, \ldots, s_{i-n+1})$.

The vector-valued polynomial recurrences can be generalized in terms of Φ-generators as follows. Consider n congruential generators $\Phi^{(1)}, \ldots, \Phi^{(n)}$, where $\Phi^{(l)} = \{\Phi_j^{(l)}\}_{j=1}^{k}$, and for each $j, l, \Phi_j^{(l)}$ is a function in n variables. For any set $\{\alpha_j^{(l)} : 1 \le j \le k, 1 \le l \le n\}$ of coefficients and modulus m, we define a vector-valued generator which outputs a sequence of vectors $\bar{s}_0, \bar{s}_1, \ldots$, where each $\bar{s}_i = (\bar{s}_{i,1}, \ldots, \bar{s}_{i,n}) \in Z_m^n$ is generated by the recurrence

$$\bar{s}_i \equiv \left(\sum_{j=1}^{k} \alpha_j^{(1)} \Phi_j^{(1)}(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n})(mod\ m), \ldots, \sum_{j=1}^{k} \alpha_j^{(n)} \Phi_j^{(n)}(\bar{s}_{i-1,1}, \ldots, \bar{s}_{i-1,n})(mod\ m) \right) \qquad (4.4.1)$$

It is easy to see that vector-valued recurrences of the form (4.4.1) can be predicted in a similar way to the single-valued recurrences studied in the previous section. One can apply the prediction method of Section 4.3 to each of the "sub-generators" $\Phi_j^{(l)}, l = 1, \ldots, n$. Notice that $\bar{s}_i$ is computed by applying the functions $\Phi_j^{(l)}$ to the vector $\bar{s}_{i-1}$, and that this $\bar{s}_{i-1}$ is *known* to the predictor at the time of computing its prediction for $\bar{s}_i$. Thus, each of the sequences $\{s_{i,l}\}_{i=0}^{\infty}, l = 1, \ldots, n$ are efficiently predictable and so is the whole vector sequence. The number of possible prediction errors is as the sum of possible errors in each of the sub-generators $\Phi^{(l)}$. That is, at most n times the bound of Theorem 4.3.4.

One can take advantage of the fact that the different sub-generators work with the same modulus m in order to accelerate the convergence to the true value of m. At the end of each prediction step, we have n (not necessarily different) estimates $\hat{m}^{(1)}, \ldots, \hat{m}^{(n)}$ computed by the predictors for $\Phi^{(1)}, \ldots, \Phi^{(n)}$, respectively. In the next prediction we put all the predictors to work with the same estimate $\hat{m}$ computed as $\hat{m} = gcd(\hat{m}^{(1)}, \ldots, \hat{m}^{(n)})$. This works since each of the $\hat{m}^{(l)}$ is guaranteed to be a multiple of m (claim (a) in Lemmas 4.3.1 and 4.3.3). In this way we get that the total number of mistakes is bounded by $(nk+1)(\log \hat{m}_0 + 1)$. Notice that the dimension of the whole system of equations corresponding to the n $\Phi^{(l)}$-generators is nk (as is the total number of coefficients hidden from the predictor). On the other hand, the bound on $\hat{m}_0$ from Lemma 4.3.1 is still valid. It does not depend on the number of sub-generators since we predict each $\Phi^{(l)}$-generator (i.e. solve the corresponding system of equations) separately. Thus, we can restate Theorem 4.3.4 for the vector-valued case.

Theorem 4.4.1: Vector-valued recurrences of the form (4.4.1) are efficiently predictable provided that each $\Phi^{(l)}$-generator, $l = 1, \ldots, n$, has polynomial-time non-reduced complexity. The number of mistakes made by the above predictor is $O(\, n\, k^2 \log(k\, m\, M)\,)$, where M is an upper bound on the

output of each of the functions $\Phi_j^{(l)}$, $j=1,\ldots,k$, $l=1,\ldots,n$, working over integers in the set $\{0,\ldots,m-1\}$. In particular, for vector-valued polynomial recurrences in n variables and degree at most d the number of mistakes is $O(\, n\,k^2 \log(k\,m^d)\,)$, where $k = \binom{n+d}{d}$.

Remark 4.4.1: For simplicity we have restricted ourselves to the case (4.4.1) in which the sub-generators $\Phi^{(l)}$ work on the last vector $\bar{s}_{i-1}$. Clearly, our results hold for the more general case in which each of these sub-generators may depend on the whole vector sequence $\bar{s}_{-n_0},\ldots,\bar{s}_{i-1}$ output so far. In this case the number n of sub-generators does not depend on the number of arguments the sub-generators work on, and the number of arguments does not effect the number of mistakes.

Chapter 5:

The Composition of Zero-Knowledge Interactive Proofs

5.1. INTRODUCTION

In this chapter we apply the results on evasive pseudorandom distributions, presented in chapter 3, to the investigation of zero-knowledge interactive proof systems. We address the question of whether the (sequential and/or parallel) composition of interactive proofs preserves the zero-knowledge property.

The notions of interactive proofs and zero-knowledge were introduced by Goldwasser, Micali and Rackoff [GMR1]. Here, we give an informal outline of these notions. For formal and complete definitions, as well as the basic results concerning these concepts, the reader is referred to [GMR1, GMW1, GMR2].

An *interactive proof* for a language L is a two-party protocol in which a computationally powerful *Prover* proves to a probabilistic polynomial-time *Verifier* whether their common input x belongs to the language L. The computation is carried out by exchanging messages between the two parties. The acceptance or rejection of the input x (as belonging to L) is decided by the verifier depending on the whole conversation. If the assertion is true, i.e. $x \in L$, then the verifier will accept with very high probability. (This is referred to as the *completeness* condition). If the assertion is false then the probability to convince the verifier to accept is negligibly small, no matter how the prover behaves during the execution of the protocol. (This is the *soundness* condition).

An interactive proof is called *zero-knowledge* if on input $x \in L$ no polynomial-time verifier (even one that arbitrarily deviates from the predetermined program) gains information from the execution of the protocol, except the knowledge that x belongs to L. This means that any polynomial-time computation based on the conversation with the prover can be simulated, without interacting with the real prover, by a probabilistic polynomial-time machine ("the simulator") that gets x as its only input. More precisely, let $< P, V^* > (x)$ denote the probability distribution generated by the interactive machine (verifier) V^* which interacts with the prover P on input $x \in L$. We say that an interactive proof is *zero-knowledge* if for all probabilistic polynomial-time machines V^*, there exists a probabilistic polynomial-time algorithm M_{V^*} (called a *simulator*) that on input $x \in L$ produces a probability distribution $M_{V^*}(x)$ that is polynomially indistinguishable (see definition 3.2.2) from the distribution $< P, V^* > (x)$. (This notion of zero-knowledge is also

called *computational zero-knowledge*. The results in this chapter concern only this notion [1]).

The above description of the notion of zero-knowledge corresponds to the original definition of zero-knowledge in [GMR1]. This definition requires that the simulator is able to simulate the conversations between the prover and any probabilistic polynomial-time verifier on the common input. Later, stronger formulations of zero-knowledge were introduced in which the simulation requirement is extended to deal with stronger verifiers [F, GMR2, O, TW]. Namely, verifiers with non-uniform properties, e.g. probabilistic polynomial-time verifiers which get an additional *auxiliary-input* tape. One central reason for these extensions is that in these stronger models one can prove that repeated executions of zero-knowledge protocols preserve the zero-knowledge condition. That is, the *sequential composition* of zero-knowledge protocols results in a new protocol which is still zero-knowledge (see [O]). This preservation property is crucial for the the utilization of zero-knowledge interactive proofs in cryptographic applications and in particular to the construction of cryptographic protocols for playing any computable game [Y2,GMW2].

Whether the original ("uniform") formulation of zero-knowledge is closed under sequential composition was an open problem. It was conjectured that this is not the case, what implies the necessity of the stronger models for guaranteeing the preservation property. In section 5.2 we prove this conjecture.

Another way to compose interactive proofs is by *parallel composition*, i.e. by concurrent execution of the corresponding protocols. This kind of composition is used in order to decrease the error probability of an interactive proof *without* increasing the number of messages exchanged. Parallelism is also used in multy-party protocols in which parties wish to prove (the same and/or different) statements to various parties concurrently. We prove that parallel composition of interactive proofs does not necessarily preserve zero-knowledge, not even under the strong models of zero-knowledge. We present two protocols, both being zero-knowledge in a strong sense yet their parallel composition is not zero-knowledge (not even in the weak sense of [GMR1] formulation).

Further results concerning the parallel composition of zero-knowledge interactive proofs are presented in chapter 6.

5.2. SEQUENTIAL COMPOSITION OF ZERO-KNOWLEDGE PROTOCOLS

Here we prove that the original definition of zero-knowledge (with uniform verifiers) introduced in [GMR1] *is not closed* under sequential composition.

[1] Other definitions were proposed in which it is required that the distribution generated by the simulator is *identical* to the distribution of conversations between the verifier and the prover (*perfect* zero-knowledge), or at least statistically close (*statistical* zero-knowledge). See [GMR2] for further details.

We begin by giving a formal definition of "sequential composition" of interactive proof systems.

Definition 5.2.1: Let $< P_1, V_1 >$, ... , $< P_k, V_k >$ be interactive proof systems for languages L_1, L_2, ..., L_k, respectively. A *sequential composition* of the k protocols, $< P, V >$, is defined as follows. The common input, x, to $< P, V >$ will be a string of the form $x_1\%x_2\%\ldots\%x_k\%$, where ' %' is a delimiter. The execution of $< P, V >$ consists of k stages. At stage i, P and V activate P_i and V_i, respectively, as subroutines on x_i. The verifier V accepts if all V_i's have accepted.

Intuitively, the reason that a zero-knowledge protocol could not be closed under sequential composition is that the definition of zero-knowledge requires that the information transmitted in the execution of the protocol is "useless" for any polynomial-time computation; it does not rule out the possibility that a cheating verifier could take advantage of this "knowledge" in a subsequent interaction with the (non-polynomial) prover for obtaining valuable information. This intuition (presented in [F]) is the basis of our example of a protocol which is zero-knowledge in a single execution but reveals significant information when composed twice in a sequence. This protocol uses a polynomial-time evasive ensemble as defined in section 3.5. We essentially use the result of Theorem 3.5.1. For the application here we need the following technical formulation of the claim proved in that theorem.

Theorem 5.2.1: There exists an infinite sequence of sets $S_1, S_2, \cdots$, such that for each n, S_n is a subset of $\{0,1\}^{4n}$ of size 2^n, and the collection of uniform distributions on each of these sets constitutes an evasive and pseudorandom ensemble.

As stressed in Remark 3.5.2 the above ensemble is evasive even against algorithms which get an additional input of length n (in our application the algorithms are probabilistic polynomial-time verifiers and the additional input is the input x to the protocol). Since the proof of existence (and samplability) of such an ensemble does not rely on any unproven assumption, so does the proof of the next theorem.

Theorem 5.2.2: Computational Zero-Knowledge ([GMR1] formulation) is not closed under sequential composition.

Proof: Let $S_1, S_2, \ldots$ be an "evasive and pseudorandom" sequence as described in Theorem 5.2.1. Also, let K be a hard Boolean function, in the sense that the language $L_K = \{x: K(x)=1\}$ is not in BPP (we use this function as a "knowledge" function).

We present the following interactive-proof protocol $< P, V >$ for the language $L = \{0,1\}^*$. (Obviously, this language has a trivial zero-knowledge proof in which the verifier accepts every input, without carrying out any interaction. We intentionally modify this trivial protocol in order to demonstrate a zero-knowledge protocol which fails sequential composition).

Let x be the common input for P and V, and let n denote the length of x. The verifier V begins by sending to the prover a random string s of length $4n$. The prover P checks whether $s \in S_n$ (the n-th set in the above sequence). If this is the case (i.e. $s \in S_n$) then P sends to V the value of $K(x)$. Otherwise (i.e. $s \notin S_n$), P sends to V a string s_0 randomly selected from S_n. In any case the verifier accepts the input x (as belonging to L).

We stress that the same sequence of sets is used in all the executions of the protocol. Thus, the set S_n does not depend on the specific input to the protocol, but only on its length. Therefore, the string s_0, obtained by the verifier in the first execution of the protocol, enables him to deviate from the protocol during a second execution in order to obtain the value of $K(x')$, for any x' of length n (and in particular for $x' = x$). Indeed, consider a second execution of the protocol, this time on input x'. A "cheating" verifier which sends the string $s = s_0$ instead of choosing it at random, will get the value of $K(x')$ from the prover. Observe that this cheating verifier obtain information that cannot be computed by itself. There is no way to simulate in probabilistic polynomial-time the interaction in which the prover sends the value of $K(x')$ (otherwise the language L_K is in BPP).

Thus, it is clear that the protocol is not zero-knowledge when composed twice. On the other hand, the protocol is zero-knowledge (when executed the first time). To show this, we present for any verifier V^*, a polynomial-time simulator M_{V^*} that can simulate the conversations between V^* and the prover P. There is only one message sent by the prover during the protocol. It sends the value of $K(x)$, in case that the string s sent by the verifier belongs to the set S_n, and a randomly selected element of S_n, otherwise. By the evasivity condition of the set S_n, there is only a negligible probability that the first case holds. Indeed, no probabilistic polynomial-time machine (in our case, the verifier) can find such a string $s \in S_n$, except with insignificant probability (no matter what the input x to the protocol is). Thus, the simulator can succeed by always simulating the second possibility, i.e. the sending of a random element s_0 from S_n. This step is simulated by randomly choosing s_0 from $\{0,1\}^{4n}$ rather than from S_n. The indistinguishability of this choice from the original one follows from the indistinguishability between the uniform distribution on S_n and the uniform distribution on $\{0,1\}^{4n}$. □

Remark 5.2.1: The argument presented in the proof generalizes to any language L having a zero-knowledge interactive proof. Simply, modify the zero-knowledge proof for L as in the above proof of Theorem 5.2.2.

Remark 5.2.2: Another example of a zero-knowledge protocol which is not closed under sequential composition was independently found by D. Simon [Si]. His construction assumes the existence of secure encryption systems.

5.3. PARALLEL COMPOSITION OF ZERO-KNOWLEDGE PROTOCOLS

In this section we address the question of whether zero-knowledge interactive proofs are robust under parallel composition.

First we present a definition of "parallel composition" of interactive proof systems.

Definition 5.3.1: Let $< P_1, V_1 >$, ... , $< P_k, V_k >$ be interactive proof systems for languages L_1, L_2, ..., L_k, respectively. Without loss of generality, assume that all protocols are m-step protocols. A *parallel composition* of the k protocols, $< P, V >$, is defined as follows. $< P, V >$ will also be an m-step protocol. The common input, x, to $< P, V >$ will be a string of the form $x_1 \% x_2 \% \ldots \% x_k \%$, where ' %' is a delimiter. The i-th message in $< P, V >$ will consist of the concatenation of the i-th messages of $< P_1, V_1 >$, ... , $< P_k, V_k >$ respectively. The verifier V accepts if all V_i's have accepted.

Clearly, we cannot expect the [GMR1] definition of zero-knowledge to be closed under parallel composition: it is easy to see that a zero-knowledge protocol which is not closed under sequential composition can be transformed into another zero-knowledge protocol which fails parallel composition. Thus, our result of the previous section implies that zero-knowledge with uniform verifiers is not closed under parallel execution.

In light of the fact that *auxiliary-input* zero-knowledge is robust under sequential composition [O], it is an interesting open question whether this formulation of zero-knowledge is also robust under parallel composition. The main result of this section is that this is *not* the case. We prove the existence of protocols which are zero-knowledge even against non-uniform verifiers (e.g. auxiliary-input zero-knowledge), but which do not remain zero-knowledge when executed twice in parallel. As in the case of sequential composition our result concerns only computational zero-knowledge.

The ideas used for the design of a protocol which fails parallel composition are similar to those used for the sequential case. There, we have used a pseudorandom and evasive ensemble to construct the intended protocol. We use this method also here. The main difference is that now we need an evasive collection which resists also non-uniform verifiers. Thus, we use the stronger notion of non-uniform evasiveness introduced in section 3.6.

Theorem 5.3.1: Computational Zero-Knowledge (even with non-uniform verifiers) is not closed under parallel composition.

Proof: We present a pair of protocols $< P_1, V_1 >$ and $< P_2, V_2 >$ which are zero-knowledge when executed independently, but whose parallel composition is provable not zero-knowledge.

We use some dummy steps in the protocols in order to achieve synchronization between them. Of course one can modify the protocol substituting these extra steps by significant ones.

The version we give here prefers simplicity rather than "naturality". Both protocols consist of five steps and are described next. An alternative description is presented in Figure 5.3.1. The notation $s \in_R S$ means the element s is chosen at random (i.e. with uniform probability) from the set S.

P_1	V_1	step	P_2	V_2
$i \in_R \{1,\cdots,2^n\} \dashrightarrow$		1	dummy step	
	dummy step	2		$\dashleftarrow j \in_R \{1,\cdots,2^n\}$
dummy step		3	$r \in_R S_j^{(n)} \dashrightarrow$	
	$\dashleftarrow s \in_R \{0,1\}^{4n}$	4		dummy step
if $s \in S_i^{(n)}$: $K(x) \dashrightarrow$		5	dummy step	

Figure 5.3.1: protocols $< P_1, V_1 >$ and $< P_2, V_2 >$ with input x.

The first protocol is denoted $< P_1, V_1 >$. Let x be the input to the protocol and let n denote its length. The protocol uses (for all its executions) a P/poly-evasive sequence $S^{(1)}, S^{(2)}, \ldots$ with the properties described in Theorem 3.6.1. It also involves an (arbitrary) hard Boolean function K as in the proof of Theorem 5.2.2. The prover P_1 begins by sending to V_1 an index $i \in_R \{1,\ldots,2^n\}$. After two dummy steps the verifier V_1 sends to P_1 a string $s \in_R \{0,1\}^{4n}$. The prover P_1 checks whether $s \in S_i^{(n)}$. If this is the case then it sends to V_1 the value of $K(x)$. This concludes the protocol.

The second protocol $< P_2, V_2 >$ uses the *same* P/poly-evasive sequence $S^{(1)}, S^{(2)}, \ldots$ as protocol $< P_1, V_1 >$ does. The first step of the protocol is a dummy one. In the second step the verifier V_2 sends to P_2 an index $j \in_R \{1,\ldots,2^n\}$. The prover P_2 responds with a string $r \in_R S_j^{(n)}$. After two more dummy steps the protocol stops.

We show that the above protocols are indeed zero-knowledge (even for non-uniform verifiers). For the first protocol, there are two steps of the prover to be simulated. In the first step P_1 sends an index $i \in_R \{1,\ldots,2^n\}$. The simulator does the same. In the second step, the prover sends the value of $K(x)$ *only* if the verifier succeeds to present him a string which belongs to the set $S_i^{(n)}$. By the evasivity condition of the sequence $S^{(1)}, S^{(2)}, \ldots$, this will happen with negligible probability and therefore the simulator can always simulate this step as for the case where the verifier sends a string $s \notin S_i^{(n)}$. Observe that the circuits in the definition of P/poly-evasive sequences only get as input the index of the set to be hit. Nevertheless, in our case the circuits also get an additional input x. This cannot help them finding an element in $S_i^{(n)}$. Otherwise,

circuits which have such a string incorporated will contradict the evasiveness condition.

In the second protocol, $< P_2, V_2 >$, the only significant step of the prover P_2 is when it sends an element $r \in_R S_j^{(n)}$ in response to the index j sent by the verifier. In this case the simulator will send a string $r' \in_R \{0,1\}^{4n}$. Using the pseudorandomness property of the set $S_j^{(n)}$ we get that the simulator's choice is polynomially indistinguishable from the prover's one.

Finally we show that the parallel composition of the above protocols into a single protocol $< P, V >$ is not zero-knowledge. Let V^* be a "cheating" verifier which behaves as follows. Instead of sending a randomly selected index j (corresponding to the second step of the subprotocol $< P_2, V_2 >$) it sends the index i received from P as part of P_1's first step. Thus, $j = i$, and the prover P will respond with a string $r \in S_i^{(n)}$. In the next step this string r will be sent by V to P as the "random" string s that V_1 should send to P_1. The prover P will verify that $r \in S_i^{(n)}$ and then will send the information $K(x)$. By the hardness of the function K this step cannot be simulated by a probabilistic polynomial-time machine. Therefore, the composed protocol $< P, V >$ is not zero-knowledge. □

Remark 5.3.1: The two protocols $< P_1, V_1 >$ and $< P_2, V_2 >$ can be merged into a single zero-knowledge protocol which is not robust under parallel composition. For example, let the prover in the merged protocol choose at random an index $i \in \{1, 2\}$, send it to V, and then both parties execute the protocol $< P_i, V_i >$. This protocol, when executed twice in parallel, has probability one-half to become a parallel execution of $< P_1, V_1 >$ and $< P_2, V_2 >$. Therefore, it is not zero-knowledge.

Chapter 6:

On the Round Complexity of Zero-Knowledge Interactive Proofs

6.1. INTRODUCTION

In chapter 5 we have shown that the definition of zero-knowledge does not guarantee the preservation of the zero-knowledge property when composing protocols in parallel. This was shown even for the strong formulations of zero-knowledge which allow non-uniform "cheating" verifiers. This result leaves open the question of whether particular protocols have the composition property or not. That is, whenever we are interested to compose concrete protocols we must investigate whether the resultant composed protocol remains zero-knowledge or not.

Such a question arises in many cases. In particular, since the first works on zero-knowledge [GMR1, GMW1] it was repeatedly asked whether the parallel versions of the protocols for Quadratic Residuosity, Graph Isomorphism and for any language in NP, presented in these papers, are zero-knowledge. The main motivation for this question is that these "parallel versions" use only a constant number of rounds (in this case, 3 rounds), while the "sequential versions" (proved zero-knowledge in the above works) use an unbounded number of rounds.

In this chapter we report a general result concerning the round complexity of zero-knowledge interactive proofs which, in particular, resolves the question of parallelization of the mentioned protocols. This general result states that *only BPP languages have 3-round interactive proofs which are black-box simulation zero-knowledge.*

Since the parallel versions of the above examples are 3-round interactive proofs it follows that these interactive proofs cannot be proven zero-knowledge using black-box simulation zero-knowledge, unless the corresponding languages are in BPP. We say that an interactive proof *is black-box simulation zero-knowledge* if there exists a universal simulator which using any (even non-uniform) verifier V^* as a black box, produces a probability distribution which is polynomially indistinguishable from the distribution of conversations of (the same) V^* with the prover. This definition of zero-knowledge is more restrictive than the original one which allows to have a specific simulator for each verifier V^*. Nevertheless, all known zero-knowledge protocols (with non-uniform verifiers) are also black-box simulation zero-knowledge. This fact cannot come as a surprise since it is hard to conceive a way to take advantage of the full power of the more liberal definition.

It is not plausible that our result could be extended to 4-round interactive proofs since such proofs are known for languages believed to be outside BPP. The protocols for Quadratic Non-Residuosity [GMR1] and Graph Non-Isomorphism [GMW1] are such examples. In addition, zero-knowledge interactive proofs of 5 rounds are known for Quadratic Residuosity and Graph Isomorphism [BMO], and assuming the existence of claw-free permutations there exist 5-round zero-knowledge interactive proofs for any language in NP [GKa]. Moreover, our results extend to zero-knowledge *arguments* [1], for which Feige and Shamir [FS] presented (assuming the existence of one-way functions) a 4-round protocol for any language in NP. Our result implies that the round complexity of this protocol is optimal (as long as BPP $\neq$ NP).

When restricting ourselves to Arthur-Merlin interactive proofs, we can extend the above result to any constant number of rounds. We show that *only BPP languages have constant-round Arthur-Merlin proofs which are also black-box simulation zero-knowledge.*

Arthur-Merlin interactive proofs, introduced by Babai [Ba], are interactive proofs in which all the messages sent by the verifier are the outcome of his coin tosses. In other words, the verifier "keeps no secrets from the prover". This result is a good reason to believe that the only feasible way of constructing constant-round zero-knowledge interactive proofs is to let the verifier use "secret coins". (Indeed, the above mentioned constant-round zero-knowledge proofs use secret coins). Thus, "secret coins" help in the (black-box simulation) zero-knowledge setting. This should be contrasted with the result of Goldwasser and Sipser [GS] which states that Arthur-Merlin interactive proofs are *equivalent* to general interactive proofs (as far as language recognition is concerned). They show that any language having a general interactive proof of k rounds, has also an Arthur-Merlin proof of k rounds. Using our result we see that in the zero-knowledge setting such a preservation of rounds is not plausible (e.g., Graph Non-Isomorphism).

Our result is tight in the sense that, the languages considered above (e.g. Graph Non-Isomorphism, NP) have unbounded (i.e. $\omega(n)$-round, for every unbounded function $\omega: N \rightarrow N$) Arthur-Merlin proof systems which are black-box simulation zero-knowledge. In particular, we get that bounded round Arthur-Merlin proofs which are black-box zero-knowledge exist only for BPP, while unbounded round proofs of the same type exist for all NP (if one-way functions exist). That is, while the *finite hierarchy* of languages having black-box zero-knowledge Arthur-Merlin proofs collapses to BPP (= AM(0)), the corresponding *infinite hierarchy* contains all NP. This implies (assuming the existence of one-way functions) a separation between the two hierarchies.

---

[1] Interactive *arguments* differ from an interactive proof system in that the soundness condition of the system is formulated with respect to *probabilistic polynomial-time* provers, possibly with auxiliary input (see [BCC]).

6.2. SECRET COINS HELP ZERO-KNOWLEDGE

In this section we present our result concerning zero-knowledge proofs systems in which the interaction is of Arthur-Merlin type. In such systems the (honest) verifier chooses its messages at random, while the only real computation it carries out is the evaluation of a polynomial-time predicate at the end of the interaction, in order to decide the acceptance or rejection of the input to the protocol.

We show that only languages in BPP have constant-round Arthur-Merlin interactive proofs which are *black-box simulation zero-knowledge*. A zero-knowledge interactive proof $< P, V >$ is called *black-box simulation* zero-knowledge if it is proved zero-knowledge by presenting a universal simulator, which using any verifier V^* as a black-box, succeeds in simulating the $< P, V^* >$ interaction [O]. In this definition of zero-knowledge we allow the verifier V^* to be non-uniform.

The main Theorem of this section is

Theorem 6.2.1: A language L has a constant-round Arthur-Merlin interactive proof which is black-box simulation zero-knowledge if and only if $L \in$ BPP.

We present a proof for a special case of this Theorem. Namely, for the case of a three-round Arthur-Merlin protocol. The general case is proved using careful extensions of the ideas presented here. The three-round case can also be extended for general interactive proof systems. That is, we also have the following Theorem.

Theorem 6.2.2: A language L has a three-round interactive proof which is black-box simulation zero-knowledge if and only if $L \in$ BPP.

6.2.1 The case AM(3)

Consider an Arthur-Merlin protocol $< P, V >$ for a language L, consisting of three rounds. We use the following notation. Denote by x the input for the protocol, and by n the length of this input. The first message in the interaction is sent by the prover. We denote it by α. The second round is for V which sends a string β. The third (and last) message is from P and we denote it by γ. The predicate computed by the verifier V in order to accept or reject the input x is denoted by ρ_V, and we consider it, for convenience, as a deterministic function $\rho_V(x, \alpha, \beta, \gamma)$. (Our results hold also for the case in which the predicate ρ_V also depends on an additional random string r). We will also assume, w.l.o.g., the existence of a polynomial $l(n)$ such that $|\alpha| = |\beta| = l(n)$.

Let this three-round Arthur-Merlin protocol $< P, V >$ be black-box simulation zero-knowledge. Denote by M the guaranteed black-box simulator which given access to the black-box V^* can simulate $< P, V^* >$. The process of simulation consists of several "tries" or calls to the interacting verifier V^* ("the black-box"). In each such call the simulator M feeds the arguments for V^*.

These arguments are the input y (which may be different from the "true" input x), the random coins for V^*, denoted r, and a string α representing the message sent by the prover P. Finally, after completing its tries the simulator outputs a conversation $(y, r, \alpha, \beta, \gamma)$. Notice that the simulator runs polynomial-time and therefore there exists a polynomial $t(n)$ which bounds the number of calls tried before outputting a conversation.

We shall make some simplifying assumptions on the behavior of the simulator M, which will not restrict the generality. In particular, we assume that some cases, which may arise with only *negligible* [2] probability, do not happen at all. This cannot significantly affect the success probability of the simulator. We assume that the conversations output by M have always the form $(x, r, \alpha, \beta, \gamma)$, i.e. $y = x$, and that the string β equals the message output by V^* when given the inputs x, r and α. Note that these conditions always hold for the real conversations generated by the prover P and the verifier V^*. Therefore, the simulator must almost always do the same. (Otherwise, a distinguisher which has access to V^*, would distinguish between the simulator's output and the original conversations). We also assume that the simulator M explicitly tries, in one of its calls to V^*, the parameters x, r and α appearing in the output conversation.

We observe that the behavior of the simulator M, interacting with a verifier V^*, is *completely determined* by the input x, the random tape R_M used by M and the strings output by V^* in response to the arguments fed by the simulator during its tries. Based on this observation we define the following process in which the simulator M itself is used as a subroutine.

Fix an input x of length n, a string R_M and $t = t(n)$ strings $\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(t)}$. Activate M on input x with its random tape containing R_M. For each y, r and α presented by M, respond in the following way. The responses will depend only on the strings α (and not in y and r). If α was previously presented by M, respond with the same β as before. If α is the i-th *differentstring presented by M then respond with* $\beta^{(i)}$. *We denote the i-th different* α *by* $\alpha^{(i)}$. *Clearly,* $\alpha^{(i)}$ *is uniquely determined by* x, R_M *and the* $i-1$ *strings* $\beta^{(1)}, \ldots, \beta^{(i-1)}$. *That is, there exists a deterministic function* α_M *such that* $\alpha^{(i)} = \alpha_M(x, R_M, \beta^{(1)}, \ldots, \beta^{(i-1)})$. *We denote by* $conv_M(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)}) = (x, r, \alpha, \beta, \gamma)$, *the conversation output by the simulator M when activated with these parameters. By our convention on the simulator M, there exists* $i, 1 \le i \le t$ *such that* $\alpha = \alpha^{(i)}$ *and* $\beta = \beta^{(i)}$.

Definition: We say that a vector $(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$ is *M-good* if $conv_M(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$ is an accepting conversation for the (honest) verifier V. Namely, if $\rho_V(x, \alpha, \beta, \gamma) = 1$, where $conv_M(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)}) = (x, r, \alpha, \beta, \gamma)$. We say that $(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$ is *i-good* if it is *M*-good and $\alpha = \alpha^{(i)}$ and $\beta = \beta^{(i)}$.

[2] We use the term "negligible" for denoting functions which are (asymptoticly) smaller than $1/Q$, for any polynomial Q.

The main property of M-good strings is stated in the following Lemma. We get as a corollary the proof of Theorem 6.2.1 for the case AM(3).

Lemma 6.2.3: Let $< P, V >$ be a 3-rounds Arthur-Merlin protocol for a language L. Suppose $< P, V >$ is black-box simulation zero-knowledge, and let M be a black-box simulator as above. Then, $x \in L$ if and only if all but a negligible portion of the vectors $(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$ are M-good.

Before proving this key lemma, we use it to prove Theorem 6.2.1 for the case of three-round Arthur-Merlin interactive proof.

Proof of Theorem 6.2.1 (for case AM(3)): By Lemma 6.2.3 we get that the following is a BPP algorithm for the language L.

* select at random a vector $(R_M, \beta^{(1)}, \ldots, \beta^{(t)})$.

* accept the input x if and only if $(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$ is M-good. □

Proof of Lemma 6.2.3:

IF direction: Let R_0 be a string for which there exist a non-negligible number of vectors $(\beta^{(1)}, \ldots, \beta^{(t)})$ such that $(x, R_0, \beta^{(1)}, \ldots, \beta^{(t)})$ are M-good. There exists an index $i_0, 1 \le i_0 \le t$, for which a non-negligible fraction of the above $(x, R_0, \beta^{(1)}, \ldots, \beta^{(t)})$ are i_0-good. Thus, there exists a non-negligible number of prefixes $(\beta^{(1)}, \ldots, \beta^{(i_0-1)})$, each with a non-negligible number of i_0-good continuations $(\beta^{(i_0)}, \ldots, \beta^{(t)})$ (i.e., such that $(x, R_0, \beta^{(1)}, \ldots, \beta^{(i_0-1)}, \beta^{(i_0)}, \ldots, \beta^{(t)})$ are i_0-good). Let $(\beta^{(1)}, \ldots, \beta^{(i_0-1)})$ be such a prefix, and let $\alpha^{(i_0)} = \alpha_M(x, R_0, \beta^{(1)}, \ldots, \beta^{(i_0-1)})$.

For each i_0-good continuation $(\beta^{(i_0)}, \ldots, \beta^{(t)})$ machine M outputs a conversation $(x, r, \alpha^{(i_0)}, \beta^{(i_0)}, \gamma)$ for which $\rho_V(x, \alpha^{(i_0)}, \beta^{(i_0)}, \gamma) = 1$. In particular, there exists a non-negligible number of $\beta^{(i_0)}$ for which this happens.

In other words, we have shown the existence of a string α $(= \alpha^{(i_0)})$ for which the set $B(x, \alpha) = \{\beta : \exists \gamma, \rho_V(x, \alpha, \beta, \gamma) = 1\}$ is of non-negligible size among all possible strings β. By the soundness property of the AM protocol for L, we get $x \in L$. (For $x \notin L$, the prover may convince V that $x \in L$ with only negligible probability. Since the honest V selects its responses β at random, then we have that for $x \notin L$ and for all α, the set $B(x, \alpha)$ is of negligible size).

ONLY IF direction: We show that for $x \in L$ most (i.e. all but a negligible portion) of the vectors $(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$ are M-good. We do it by considering the behavior of the simulator M when simulating the conversations of the prover P with a particular family of verifiers which we introduce shortly.

Let $x \in L$ and let n denote its length. Consider a family of hash functions $H(n)$ which map $l(n)$-bit strings to $l(n)$-bit strings, such that the locations assigned to the strings by a randomly

selected hash function are uniformly distributed and $t(n)$-wise independent. (Recall that $l(n)$ is the length of messages α and β in the Arthur-Merlin protocol $< P, V >$ for L, while $t(n)$ is the bound on the number of M's tries). For properties and implementation of such functions see [J]. We denote this set of hash functions by $H(n)$.

For each hash function $h \in H(n)$ we associate a (deterministic) verifier V_h^*, which responds to the prover's message α with the string $\beta = h(\alpha)$. Consider the simulation of $< P, V_h^* >$ conversations by the simulator M. Fixing a random tape R_M for M and a function $h \in H(n)$, the whole simulation is determined. In particular, this defines a sequence of α's tried by the simulator, and the corresponding responses β of V_h^*. Denote by $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(s)}$, the *different* values of α in these tries. In case that $s < t$, we complete this sequence to $\alpha^{(1)}, \ldots, \alpha^{(s)}, \alpha^{(s+1)}, \ldots, \alpha^{(t)}$, by adding $t - s$ strings α in some canonical way, such that the resultant $\alpha^{(1)}, \ldots, \alpha^{(t)}$ are all different. Let $\beta^{(i)} = h(\alpha^{(i)}), 1 \le i \le t$, and define $v(x, R_M, h) = (x, R_M, \beta^{(1)}, \ldots, \beta^{(t)})$. The only-if direction of the Lemma follows from the following two Claims.

Claim 1: For any $x \in L$ and for all but a negligible portion of the pairs (R_M, h) the vector $v(x, R_M, h)$ is M-good.

Proof: By the completeness property of the protocol $< P, V >$, most of the conversations between P and V on input $x \in L$ are accepting. That is, for most coin sequences R_P of the prover P, and most choices β of V, the resultant conversation $(x, \alpha(x, R_P), \beta, \gamma(x, R_P, \beta))$ is accepting.

Consider now the interaction between the prover P and the verifiers V_h^* on $x \in L$. By the uniformity property of the family $H(n)$ we get that for every α, all β's are equi-probable as the result of $h(\alpha)$. This, together with the above remark on the conversations between P and V, implies that for most strings R_P, and for most hash functions h, the interaction of P with V_h^* leads to an accepting conversation.

Since the simulator M succeeds in simulating $< P, V_h^* >$ conversations for all functions $h \in H(n)$, we get that for most h's the simulator M outputs with very high probability an accepting conversation. The Claim follows. ◊

Claim 2: For all strings x and R_M, the vector $v(x, R_M, h)$ is uniformly distributed over the set $\{(x, R_M, \beta^{(1)}, \ldots, \beta^{(t)}) : \beta^{(i)} \in \{0,1\}^{l(n)}\}$

Proof: Observe that

$$v(x, R_M, h) = (x, R_M, \beta^{(1)}, \ldots, \beta^{(t)}))$$

if and only if for every $i, 1 \le i \le t$,

$$h(\alpha_M(x, R_M, \beta^{(1)}, \ldots, \beta^{(i-1)})) = \beta^{(i)} .$$

On the other hand, by the uniformity and $t(n)$-independence property of the family $H(n)$, we have that for any t *different* elements $a_1,\ldots,a_t$ in the domain of the functions $h \in H(n)$, the sequence $h(a_1),\ldots,h(a_t)$ is uniformly distributed over all possible sequences $b_1,\ldots,b_t$ for b_i in the range of the functions $H(n)$.
Putting $a_i = \alpha_M(x, R_M, \beta^{(1)},\ldots,\beta^{(i-1)})$, and $b_i = \beta^{(i)}$, and using the above observation the claim follows. ◊

Claim 2 states that for any R_M, the value of $v(x, R_M, h)$ is uniformly distributed over all possible vectors $(x, R_M, \beta^{(1)},\ldots,\beta^{(t)})$. On the other hand, by Claim 1, most $v(x, R_M, h)$ are M-good, and then we get that most $(x, R_M, \beta^{(1)},\ldots,\beta^{(t)})$ are M-good.

The Lemma follows. □

References

[ACGS] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As the Whole", *SIAM Jour. on Computing*, Vol. 17, 1988, pp. 194-209.

[Ba] Babai, L., "Trading Group Theory for Randomness", *Proc. 17th STOC*, 1985, pp. 421-429.

[BMO] Bellare, M., Micali S. and Ostrovsky R., "Perfect Zero-Knowledge in Constant Rounds", to appear in *Proc. 22nd STOC*, 1990.

[BBS] L. Blum, M. Blum and M. Shub, *A Simple Secure Unpredictable Pseudo-Random Number Generator*, *SIAM Jour. on Computing*, Vol. 15, 1986, pp. 364-383.

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

[B] Boyar, J. "Inferring Sequences Produced by Pseudo-Random Number Generators", *Jour. of ACM*, Vol. 36, No. 1, 1989, pp.129-141.

[BCC] Brassard, G., D. Chaum, and C. Crépau, "Minimum Disclosure Proofs of knowledge", *JCSS*, Vol. 37, No. 2, 1988, pp. 156-189.

[BS] Butson, A.T., and Stewart, B.M., "Systems of Linear Congruences", *Canad. J. Math.*, Vol. 7, 1955, pp. 358-368.

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

[C] Chernoff, H., "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations", *Annals of Mathematical Statistics*, Vol. 23, 1952, pp. 493-507.

[CG] Chor, B., and O. Goldreich, "On the Power of Two-Point Based Sampling", *Jour. of Complexity*, Vol. 5, 1989, pp. 96-106.

[CGG] Chor, B., O. Goldreich, and S. Goldwasser, "The Bit Security of Modular Squaring Given Partial Factorization of the Modulos", *Advances in Cryptology - Crypto 85 Proceedings*, ed. H.C. Williams, Lecture Notes in Computer Science, 218, Springer Verlag, 1985, pp. 448- 457.

[DH] W. Diffie, and M. E. Hellman, "New Directions in Cryptography", *IEEE transactions on Info. Theory*, IT-22 (Nov. 1976), pp. 644-654

[Ed] Edmonds, J., "Systems of Distinct Representatives and Linear Algebra", *Journal of Research of the National Bureau of Standards (B)*, Vol. 71B, 1967, pp. 241-245.

[E] S. Even, *Graph Algorithms*, Computer Science Press, 1979.

[F] Feige, U., "Interactive Proofs", M.Sc. Thesis, Weizmann Institute, 1987.

[FS] Feige, U., and A. Shamir, "Zero knowledge proofs of knowledge in two rounds", Proceedings of *Crypto89*, 1989.

[FHKLS] Frieze, A.M., Hastad, J., Kannan, R., Lagarias, J.C., and Shamir, A. Reconstructing Truncated Integer Variables Satisfying Linear Congruences *SIAM J. Comput.*, Vol. 17, 1988, pp. 262-280.

[G] Goldreich, O., "A Note on Computational Distinguishability", TR-602, Computer Science Dept., Technion, Haifa, January 1990. To appear in *IPL*.

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

[GKa] Goldreich, O., and Kahan, A., "Using Claw-Free Permutations to Construct Constant-Round Zero-Knowledge Proofs for NP", in preparation, 1989.

[GK1] Goldreich, O., and Krawczyk, H., "Sparse Pseudorandom Distributions", Proceedings of *Crypto89*, 1989. Submitted to *Random Structures and Algorithms*.

[GK2] Goldreich, O. and Krawczyk, H., "On the Composition of Zero-Knowledge Proof Systems", TR-570, Computer Science Dept., Technion, Haifa, June 1989. To be presented at *ICALP90*.

[GKL] Goldreich, O., H. Krawczyk and M. Luby, "On the Existence of Pseudorandom Generators", *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 1988, pp 12-24. Submitted to *SIAM J. on Comput.*.

[GL] Goldreich, O., and L.A. Levin, "A Hard-Core Predicate for any One-Way Function", *21st STOC*, 1989, pp. 25-32.

[GrM] Goldreich, O., and S. Micali, "The Weakest Pseudorandom Bit Generator Implies the Strongest One", unpublished manuscript, 1984.

[GMW1] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *Proc. 27th FOCS*, 1986, pp. 174-187. Submitted to *Jour. of ACM*.

[GMW2] Goldreich, O., S. Micali, and A. Wigderson, "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority", *Proc. 19th STOC*, 1987,

pp. 218-229.

[GO] Goldreich, O., and Oren, Y., "Definitions and Properties of Zero-Knowledge Proofs", in preparation.

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

[GMR1] Goldwasser, S., S. Micali, and C. Rackoff, "Knowledge Complexity of Interactive Proofs", *Proc. 17th STOC*, 1985, pp. 291-304.

[GMR2] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Jour. on Computing*, Vol. 18, 1989, pp. 186-208.

[GS] Goldwasser, S., and M. Sipser, "Private Coins vs. Public Coins in Interactive Proof Systems", *Proc. 18th STOC*, 1986, pp. 59-68.

[H] Hoeffding W., "Probability Inequalities for Sums of Bounded Random Variables", *Journal of the American Statistical Association*, Vol. 58, 1963, pp. 13-30.

[Ha] Hastad J., "Pseudo-Random Generators under Uniform Assumptions", to appear in *Proc. 22nd STOC*, 1990.

[ILL] Impagliazzo, R., L.A., Levin and M.G. Luby, "Pseudo-random Generation from One-Way Functions", *21st STOC*, 1989, pp. 12-24.

[J] A. Joffe, "On a Set of Almost Deterministic k-Independent Random Variables", *the Annals of Probability*, 1974, Vol. 2, No. 1, pp. 161-162.

[KB] Kannan, R., and Bachem, A., "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix", *SIAM J. Comput.*, Vol. 8, 1979, pp. 499-507.

[K1] Knuth, D.E., *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass., 1969.

[K2] Knuth, D.E., "Deciphering a Linear Congruential Encryption", *IEEE Trans. Info. Th.* IT-31, 1985, pp. 49-52.

[K] Krawczyk, H., "How to Predict Congruential Generators", to appear in *Journal of Algorithms*. Presented at *Crypto89*.

[LR] Lagarias, J.C., and Reeds, J., "Unique Extrapolation of Polynomial Recurrences", *SIAM J. Comput.*, Vol. 17, 1988, pp. 342-362.

[L] L.A. Levin, "One-Way Function and Pseudorandom Generators", *Combinatorica*, Vol. 7, No. 4, 1987, pp. 357-363.

[L2] L.A. Levin, "Homogeneous Measures and Polynomial Time Invariants", *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 1988, pp 36-41.

[Lu] M. Luby, "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *SIAM J. Comput.*, Vol. 15, No. 4, Nov. 1986, pp. 1036-1054.

[LuR] M. Luby and C. Rackoff, "How to Construct Pseudorandom Permutations From Pseudorandom Functions", *SIAM Jour. on Computing*, Vol. 17, 1988, pp. 373-386.

[McS] McWilliams, F.J., and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North-Holland Publishing Company, 1977.

[N] M. Naor, "Bit Commitment Using Pseudorandomness", *Crypto89* proceedings, 1989.

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

[NW] Nissan, N. and Wigderson, A., "Hardness vs. Randomness", *Proc. of the 29th IEEE Symp. on Foundation of Computer Science*, 1988, pp. 2-11.

[O] Oren, Y., "On the Cunning Power of Cheating Verifiers: Some Observations About Zero-Knowledge Proofs", *Proc. of the 28th IEEE Symp. on Foundation of Computer Science*, 1987, pp. 462-471.

[P] Plumstead (Boyar), J.B., "Inferring a Sequence Generated by a Linear Congruence", *Proc. of the 23rd IEEE Symp. on Foundations of Computer Science*, 1982, pp. 153-159.

[R] M.O. Rabin, "Digitalized Signatures and Public Key Functions as Intractable as Factoring", MIT/LCS/TR-212, 1979.

[RSA] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM*, Vol. 21, Feb. 1978, pp 120-126

[Sch] Schrijver, A., "Theory of Linear and Integer Programming", Willey, Chichester, 1986.

[Sh] A. Shamir, "On the Generation of Cryptographically Strong Pseudorandom Sequences", *ACM Transaction on Computer Systems*, Vol. 1, No. 1, February 1983, pp. 38-44.

[Si] Simon, D., "Issues in the Definition of Zero Knowledge", M.Sc. Thesis, University of Toronto, 1989.

[S] Stern, J., "Secret Linear Congruential Generators Are Not Cryptographically Secure", *Proc. of the 28rd IEEE Symp. on Foundations of Computer Science*, 1987.

[TW] Tompa, M., and H. Woll, "Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information", *Proc. of the 28th IEEE Symp. on Foundation of Computer Science*, 1987, pp. 472-482.

[Y] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

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