Foundations of Cryptography – Teaching Notes

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science, Israel.
Email: oded@wisdom.weizmann.ac.il

Fall 2001

II

Preface

These teaching notes were initially intended for myself, as an aid for preparing the lectures in the course *Foundations of Cryptography* given at Weizmann Institute at the Fall of 2001. It then occurred to me that these notes may also serve as suggestions to others as to how such a course can be taught, and the notes were slightly adapted accordingly.

The course was based on the first two volumes of my work *Foundations of Cryptography*. The first volume, entitled *Foundations of Cryptography – Basic Tools* [14], has appeared with Cambridge University Press in June 2001. The second volume, entitled *Foundations of Cryptography – Basic Applications*, will hopefully be completed in a couple of years. Preliminary versions of the chapters of the second volume are available from the web-page

`http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`

You may want to check this web-site for various updates and notices concerning the entire work.

These teaching notes refer to the above work. Specifically, they merely summarize and annotate the parts of the work that was taught in class. The detailed material is to be found in the book itself, and the boxes flushed to the right (as the example below) refer to the sections in the book in which the relevant material is to be found.

$$\boxed{\textbf{Example}}$$

State and usage of these notes: Unfortunately, I've stopped writing these notes at the middle of the semester. Thus, the current notes only cover the first 8 lectures (out of the planned 14 lectures). I hope to complete the missing notes next time I teach this course (which may be next year).

Unlike the abovementioned work, these teaching notes were not carefully proofread. I apologize for the various errors that they may contain, and hope that nevertheless these notes will be useful.

Needless to say, the teaching pace should and does depend on the specific class. In fact, I often proceeded in pace different than the one I have originally planned, omitting or adding material on-the-fly.

From the preface of Volume 1

It is possible to build a cabin with no foundations,
but not a lasting building.

Eng. Isidor Goldreich (1906–1995)

Cryptography is concerned with the construction of schemes that withstand any abuse: Such schemes are constructed so to maintain a desired functionality, even under malicious attempts aimed at making them deviate from their prescribed functionality.

The design of cryptographic schemes is a very difficult task. One cannot rely on intuitions regarding the typical state of the environment in which the system operates. For sure, the *adversary* attacking the system will try to manipulate the environment into untypical states. Nor can one be content with counter-measures designed to withstand specific attacks, since the adversary (which acts after the design of the system is completed) will try to attack the schemes in ways that are typically different from the ones the designer had envisioned. The validity of the above assertions seems self-evident, still some people hope that in practice ignoring these tautologies will not result in actual damage. Experience shows that these hopes rarely come true; cryptographic schemes based on make-believe are broken, typically sooner than later.

In view of the above, we believe that it makes little sense to make assumptions regarding the specific *strategy* that the adversary may use. The only assumptions that can be justified refer to the computational *abilities* of the adversary. Furthermore, it is our opinion that the design of cryptographic systems has to be based on *firm foundations*; whereas ad-hoc approaches and heuristics are a very dangerous way to go. A heuristic may make sense when the designer has a very good idea about the environment in which a scheme is to operate, yet a cryptographic scheme has to operate in a maliciously selected environment which typically transcends the designer's view.

This book is aimed at presenting firm foundations for cryptography. The foundations of cryptography are the paradigms, approaches and techniques used to conceptualize, define and provide solutions to natural "security concerns". We will present some of these paradigms, approaches and techniques as well as some of the fundamental results obtained using them. Our emphasis is on the clarification of fundamental concepts and on demonstrating the feasibility of solving several central cryptographic problems.

Solving a cryptographic problem (or addressing a security concern) is a two-stage process consisting of a *definitional stage* and a *constructive stage*. First, in the definitional stage, the functionality underlying the natural concern is to be identified, and an adequate cryptographic problem has to be defined. Trying to list all undesired situations is infeasible and prone to error. Instead, one should define the functionality in terms of operation in an imaginary ideal model, and require a candidate solution to emulate this operation in the real, clearly defined, model (which specifies the adversary's abilities). Once the definitional stage is completed, one proceeds to construct a system that satisfies the definition. Such a construction may use some simpler tools, and its security is proven relying on the features of these tools. (In practice, of course, such a scheme may need to satisfy also some specific efficiency requirements.)

This book focuses on several archetypical cryptographic problems (e.g., encryption and signature schemes) and on several central tools (e.g., computational difficulty, pseudorandomness, and zero-knowledge proofs). For each of these problems (resp., tools), we start by presenting the natural concern underlying it (resp., its intuitive objective), then define the problem (resp., tool), and finally demonstrate that the problem may be solved (resp., the tool can be constructed). In the latter step, our focus is on demonstrating the feasibility of solving the problem, not on providing a practical solution. As a secondary concern, we typically discuss the level of practicality (or impracticality) of the given (or known) solution.

Computational Difficulty

The specific constructs mentioned above (as well as most constructs in this area) can exist only if some sort of computational hardness exists. Specifically, all these problems and tools require (either explicitly or implicitly) the ability to generate instances of hard problems. Such ability is captured in the definition of one-way functions (see further discussion in Section 2.1). Thus, one-way functions is the very minimum needed for doing most sorts of cryptography. As we shall see, they actually suffice for doing much of cryptography (and the rest can be done by augmentations and extensions of the assumption that one-way functions exist).

Our current state of understanding of efficient computation does not allow us to prove that one-way functions exist. In particular, the existence of one-way functions implies that $\mathcal{NP}$ is not contained in $\mathcal{BPP} \supseteq \mathcal{P}$ (not even "on the average"), which would resolve the most famous open problem of computer science. Thus, we have no choice (at this stage of history) but to assume that one-way functions exist. As justification to this assumption we may only offer the combined believes of hundreds (or thousands) of researchers. Furthermore, these believes concern a simply stated assumption, and their validity is supported by several widely believed conjectures which are central to some fields (e.g., the conjecture that factoring integers is hard is central to computational number theory).

As we need assumptions anyhow, why not just assume what we want (i.e., the existence of a solution to some natural cryptographic problem)? Well, first we need to know what we want: as stated above, we must first clarify what exactly do we want; that is, go through the typically complex definitional stage. But once this stage is completed, can we just assume that the definition derived can be met? Not really: the mere fact that a definition was derived does NOT mean that it can be met, and one can easily define objects that cannot exist (without this fact being obvious in the definition). The way to demonstrate that a definition is viable (and so the intuitive security concern can be satisfied at all) is to construct a solution based on a *better understood* assumption (i.e., one that is more common and widely believed). For example, looking at the definition of zero-knowledge proofs, it is not a-priori clear that such proofs exist at all (in a non-trivial sense). The non-triviality of the notion was first demonstrated by presenting a zero-knowledge proof system for statements, regarding Quadratic Residuosity, which are believed to be hard to verify (without extra information). Furthermore, in contrary to prior beliefs, it was later shown in that the existence of one-way functions implies that any NP-statement can be proven in zero-knowledge. Thus, facts that were not known at all to hold (and even believed to be false), were shown to hold by reduction to widely believed assumptions (without which most of modern cryptography collapses anyhow). To summarize, not all assumptions are equal, and so reducing a complex, new and doubtful assumption to a widely-believed simple (or even merely simpler) assumption is of great value. Furthermore, reducing the solution of a new task to the assumed security of a well-known primitive typically means providing a construction that, using the known primitive, solves the new task. This means that we do not only know (or assume) that the new task is solvable but rather have a solution based on a primitive that, being well-known, typically has several candidate implementations.

Structure and Prerequisites

Our aim is to present the basic concepts, techniques and results in cryptography. As stated above, our emphasis is on the clarification of fundamental concepts and the relationship among them. This is done in a way independent of the particularities of some popular number theoretic examples. These particular examples played a central role in the development of the field and still offer the most practical implementations of all cryptographic primitives, but this does not mean that the presentation has to be linked to them. On the contrary, we believe that concepts are best clarified when presented at an abstract level, decoupled from specific implementations. Thus, the most relevant background for this book is provided by basic knowledge of algorithms (including randomized ones), computability and elementary probability theory. Background on (computational) number theory, which is required for specific implementations of certain constructs, is not really required here (yet, a short appendix presenting the most relevant facts is included in this volume so to support the few examples of implementations presented here).

Volume 1:	Introduction and Basic Tools
	Chapter 1: Introduction
	Chapter 2: Computational Difficulty (One-Way Functions)
	Chapter 3: Pseudorandom Generators
	Chapter 4: Zero-Knowledge Proofs
Volume 2:	Basic Applications
	Chapter 5: Encryption Schemes
	Chapter 6: Signature Schemes
	Chapter 7: General Cryptographic Protocols
Volume 3:	Beyond the Basics
	...

Figure 0.1: Organization of this book

Organization of the book. The book is organized in three parts (see Figure 0.1): *Basic Tools*, *Basic Applications*, and *Beyond the Basics*. The current (first) volume contains an introductory chapter as well as the first part (Basic Tools). This part contains chapters on computational difficulty (one-way functions), pseudorandomness and zero-knowledge proofs. These basic tools will be used for the Basic Applications of the second part, which consist of Encryption, Signatures, and General Cryptographic Protocols.

The partition of the book into three parts is a logical one. Furthermore, it offers the advantage of publishing the first part without waiting for the completion of the other parts. Similarly, we hope to complete the second part within a couple years, and publish it without waiting for the third part.

Teaching. The material presented in this book is, on one hand, way beyond what one may want to cover in a course, and on the other hand falls very short of what one may want to know about Cryptography in general. To assist these conflicting needs we make a distinction between *basic* and *advanced* material, and provide suggestions for further reading (in the last section of each chapter). In particular, sections, subsections, and subsubsections marked by an asterisk (*) are intended for advanced reading.

Each lecture consists of one hour. Lectures 1–15 are covered by the current volume. Lectures 16–28 will be covered by the second volume.

Lecture 1: Introduction, Background, etc.
(depending on class)
Lecture 2–5: *Computational Difficulty* (*One-Way Functions*)
Main: Definition (Sec. 2.2), Hard-Core Predicates (Sec. 2.5)
Optional: Weak implies Strong (Sec. 2.3), and Sec. 2.4.2–2.4.4
Lecture 6–10: *Pseudorandom Generators*
Main: Definitional issues and a construction (Sec. 3.2–3.4)
Optional: Pseudorandom Functions (Sec. 3.6)
Lecture 11–15: *Zero-Knowledge Proofs*
Main: Some definitions and a construction (Sec. 4.2.1, 4.3.1, 4.4.1–4.4.3)
Optional: Sec. 4.2.2, 4.3.2, 4.3.3–4.3.4, 4.4.4
Lecture 16–20: *Encryption Schemes*
Definitions and a construction (consult Apdx. B.1.1–B.1.2)
(See also fragments of a draft for the encryption chapter [12].)
Lecture 21–24: *Signature Schemes*
Definition and a construction (consult Apdx. B.2)
(See also fragments of a draft for the signatures chapter [13].)
Lecture 25–28: *General Cryptographic Protocols*
The definitional approach and a general construction (sketches).
(Consult Apdx. B.3, see also our exposition [11].)

Figure 0.2: Plan for one-semester course on Foundations of Cryptography

Volumes 1 and 2 of this book are supposed to provide all material for a course on Foundations of Cryptography. For a one-semester course, the teacher will definitely need to skip all advanced material (marked by an asterisk) and maybe even some basic material: see suggestion in Figure 0.2. This should allow, depending on the class, to cover the basic material at a reasonable level (i.e., cover all material marked as "main" and some of the "optional"). Volumes 1 and 2 can also serve as textbook for a two-semester course. Either way, the current volume only covers the first half of the material mentioned above. The second half will be covered in Volume 2. In the meanwhile, we suggest to use other sources for the second half. A brief summary of Volume 2 and recommendations for alternative sources are given in Appendix B. (In addition, fragments and/or preliminary drafts for the three missing chapters are available from [12], [13] and [11], respectively.)

Giving a course solely based on the material that appears in the current volume is indeed possible, but such a course cannot be considered a stand-alone course in Cryptography (for the reason that the current volume does not consider at all the basic tasks of encryption and signatures).

Practice. The aim of this book is to provide sound theoretical foundations for cryptography. As argued above, such foundations are necessary for *sound* practice of cryptography. Indeed, practice requires more than theoretical foundations, whereas the current book makes no attempt to provide anything beyond the latter. However, given sound foundations, one can learn and evaluate various practical suggestions that appear elsewhere (e.g., in [26]). On the other hand, lack of sound foundations results in inability to critically evaluate practical suggestions, which in turn leads to unsound decisions. Nothing could be more harmful to the design of schemes that need to withstand adversarial attacks than misconceptions about such attacks.

Relation to another book by the author

A frequently asked question refers to the relation of the current book to our text *Modern Cryptography, Probabilistic Proofs and Pseudorandomness* [10]. The latter text consists of three brief introductions to the related topics in the title. Specifically, Chapter 1 of [10] provides a brief (i.e., 30-page) summary of the current book. The other two chapters of [10] provide a wider perspective on two topics mentioned in the current book (i.e., Probabilistic Proofs and Pseudorandomness). Further comments on the latter aspect are provided in the relevant chapters of the current book.

Lecture Summaries

Lecture 1: Getting Started. This is an introductory lecture: a bit of motivation (what is the course about), a bit of background (on computation), and a taste of the real material (one-way functions). Specifically:

1. Introduce the topics of the course.
2. Introduce the basic computational notions underlying the course.
3. Discuss computational difficulty as captured by one-way functions.
4. Present the first result and its proof (stressing the proof technique).

Lecture 2: Working with Intractability Assumptions. The bulk of this lecture is devoted to proving that if there exist weakly one-way functions then there exists strongly one-way functions. The construction itself is simple, but again the focus should be on the proof technique. In addition, we introduce the notion of a collection of one-way functions, and the notion of a hard-core predicate of (a one-way function).

Lecture 3: Hard-Core Predicates and Pseudorandom Generators. This lecture consists of two independent parts. In the first (and main) part we present a proof of the existence of a generic hard-core predicate. In the second part we provide a short introduction to the notion of pseudorandom generators, which will be the focus of subsequent lectures.

Lecture 4: Computational Indistinguishability. The focus of this lecture is on the concept of computational indistinguishability, which is a key concept in the area, and in particular underlies the notion of pseudorandomness. Specifically, we present the basic definition, discuss its relation to statistical indistinguishability, consider its preservation under repeated sampling, and present the hybrid technique (which is often used towards proving computational indistinguishability). We then recall the definition of pseudorandom generator and prove that a construction with any *given* stretching factor yields constructions for any *desired* stretching factor.

Lecture 5: Constructing Pseudorandom Generators and Functions. In the first part of this lecture we show how to construct pseudorandom generators using any 1-1 one-way function (i.e., one-way permutation). In the second part, we define pseudorandom functions and show how they can be constructed using any pseudorandom generator.

Lecture 6: Zero-Knowledge Interactive Proofs. We first motivate the notion of zero-knowledge. Next, we define, illustrate and discuss the natural notion of interactive proof systems (which is the adequate framework for the introduction of zero-knowledge proofs). Next we use this framework to define and illustrate the notion of zero-knowledge. The illustrative examples used are the Graph Non-Isomorphism and Graph Isomorphism protocols, respectively.

Lecture 7: Constructing Zero-Knowledge Proofs. We recall the basic conditions underlying the definitional framework of zero-knowledge proofs, and present the actual definition that is typically used. We claim (and sketch a proof) that this definition is preserved under sequential composition. We define and show how to construct commitment schemes, and using the latter we show how to construct zero-knowledge proof systems for every language in $\mathcal{NP}$.

Lecture 8: Defining Security of Encryption Schemes. After presenting the basic syntax, we define two equivalent notions of security: semantic security and the technical definition of indistinguishability of encryptions. We prove the equivalence of the two definitions, and consider their generalization to the encryption of several plaintexts (under the same key). We discuss the inherent role of probabilistic encryption algorithms (for satisfying the definitions).

Contents

Lecture 1

Getting Started

Summary

This is an introductory lecture: a bit of motivation (what is the course about), a bit of background (on computation), and a taste of the real material (one-way functions). Specifically:

1. Introduce the topics of the course.
2. Introduce the basic computational notions underlying the course.
3. Discuss computational difficulty as captured by one-way functions.
4. Present the first result and its proof (stressing the proof technique).

1.1 Introduction

Just give a vague feeling of what this course is about. Make sure the students are comfortable with randomized algorithms and with the notions of efficient and infeasible.

1.1.1 The topics

Sec. 1.1

The most basic topics of cryptography are:

1. Encryption: Providing secret communication.
2. Signatures: Providing authentic communication

Comment: here we mean *communication* in a broad sense. Nowadays we associate communication with electronic communication of computers (over the Internet) or humans (over phone lines); that is, communication is associated with real-time interaction. However, a century ago, communication was associated with the written letter or book (which are far from occurring in real-time). But non-immediate communication exists also nowadays (e.g., stored data), and by referring to communication we refer to such non-immediate communication as well.

We will also discuss general *cryptographic protocols*. These are protocols that are abuse-resilient; that is, they maintain their designated functionality also under attacks of malicious adversaries. In fact, maintaining a desired functionality under malicious attacks may be used as the definition of the entire area of cryptography.

But before embarking on the study of the above topics, we will present tools and attitudes.

1.1.2 Basic computational notions

Sec. 1.3

Secrets are inherent to cryptography! What is a secret? It is something selected by one party and unknown to others. If the others would know all determining factors effecting the selection then they'd know the secret as well. Thus, the selection must include random factors. That is, randomization is essential to cryptography.

Randomized algorithms (a natural extention of the notion of an algorithm) should be contrasted with the ficticious notion of non-deterministic machines (introduced as a technical tool). This key distinction should have been made in a course on computability and/or complexity; however, often things that should be done are not done (or are done wrongly which is even worse). Make sure the students understand it right!

Next, we confront the following:

- Efficient means computable in (probabilistic) polynomial-time. That is, there *exists* a polynomial bounding the running-time (such that an algorithm satisfying the time bound succeeds).
- Infeasible means not computable in (probabilistic) polynomial-time. That is, *for any* polynomial bounding the running-time (an algorithm satisfying the time bound fails).

In the context of randomized algorithms, success and failure are probabilistic events. Furthermore, we often consider input distributions (i.e., inputs drawn according to some probability distributions). Corresponding notions of rareness of success and failure are:

- Noticeable will indicate probability that is bounded below by a recipocal of *some* (positive) polynomial. That is, there *exists* a positive polynomial such that its recipocal *lower bounds* the said probability.
- Negligible will indicate probability that is bounded above by the recipocal of *any* polynomial. That is, *for any* (positive) polynomial its recipocal *upper bounds* the said probability.

Comment: There is nothing "holy" is using polynomials as the distinction between efficient (resp., noticeable) and infeasible (resp., negligible). Any other nicely-behaved classes of time-bounds will do: polynomials are just the simplest and most natural choice.

1.2 One-Way Functions

Computational difficulty (in the sense we need it) is captured by the notion of one-way functions. When proving Proposition 1.1 (or whenever preseting the first proof of such flavor), stress the difficulty of arguing about computational difficulty (as contrasted to the related triviality of the combinatorics).

1.2.1 Motivation

Sec. 2.1

Modern Cryptography is based on computational difficulty: efficiency of proper behavior versus infeasibility of causing harm by improper behavior.

Typically, the honest user holds a secret enabling it to efficiently take actions that are infeasible to the adversary (who does not know the secret). Furthermore, the secret is typically verifiable via some public information. Thus, finding the secret is an NP-problem, and we assume that it is not solveable in probabilistic polynomial-time. Thus, at the very least we assume that $\mathcal{P} \neq \mathcal{NP}$. But worst-case hardness (as provided by $\mathcal{P} \neq \mathcal{NP}$) is not enough: we want the secret to be hard to find almost always, not merely in the worst-case. Thus, we actually need problems in $\mathcal{NP}$ that are hard on the average. Furthermore, these hard-on-the-average problems should be easy to generate with their corresponding solutions (i.e., the secrets). This leads to the definition of one-way functions.

1.2.2 Definitions

Sec. 2.2

The basic definition is of (length preserving) functions that are efficiently computable, but are infeasible to invert in the average-case sense. That is, every probabilistic polynomial-time algorithm will fail to invert the function on a random image, except for with negligible probability (taken on the invertor's coins as well as on the random image). By a random image we mean one generated by applying the function to a uniformly chosen preimage (of any fixed length).

A weaker definition requires every (probabilistic polynomial-time) to fail only with noticeable probability (rather than succeed with at most negligible probability). The probability space is again taken over the invertor's coins as well as on the random image. We call the former type of functions strongly one-way (or just one-way), and call the latter weakly one-way.

In light of the motivating discussion, the students may wonder (or you can challenge them with the question of) *what are weakly one-way functions good for?* Jumping ahead, we mention that weak one-way functions can be transformed into strong ones (cf. Theorem 1.2). On the other hand, it is sometimes easier to construct candidate weak one-way functions that strong ones.

An example of a candidate weak one-way function: *integer mutiplication* (i.e., breaking the input into the binary representation of two equal-length integers, output the binary representation of their product). This seems hard to invert at least in case the preimage encodes two primes. (We could have presented a strong one-way function based on the corresponding intractability assumption by using a randomized algorithms for generating random primes; but this is more complicated – in fact we give no details.)

Sec. 2.3

Proposition 1.1 *Assuming the existence of one-way functions, there exists a weakly one-way function that is not strongly one-way.*

Given any one-way function, f, construct $f'(\sigma x) \stackrel{\rm def}{=} \sigma f(x)$ if $\sigma = 0^{\log_2 |x|}$ and $f'(\sigma x) \stackrel{\rm def}{=} \sigma x$ otherwise. (The length of σ should not be more than logarithmic in the length of x, and we let it be logarithmic rather than 1 in order to "maximize the dramatic effect".)

Proving the proposition: Clearly, f' is not strongly one-way. The less obvious thing is to prove that f' is weakly one-way. Just feeling that this is right does not suffice. We need to show that any algorithm that contradicts the weak one-way property of f' can be converted into an algorithm that contradicts the hypothesis that f is (strongly) one-way. Specifially, prove that (on inputs of length n) the function f' is hard to invert with success probability greater than $1 - \frac{1}{2n}$. Given an f'-inverter A', construct an f-inverted, denoted A, that on input $y = f(x)$ invokes A' on the input $0^{\log_2 |y|}y$, and outputs the $|y|$-bit long suffix of the answer. In the analysis, consider first the success probability of A' on inputs of the form $0^{\log_2 |y|}y$ (using the fact that such inputs occur with probability $1/|y|$ as images of f').

Preview: Although there may exists weak one-way functions that are not strongly one-way, we can transform *any* weak one-way function into a strong one-way function. That is:

Theorem 1.2 *If there exist weakly one-way functions, then there exists strongly one-way functions.*

The proof will be given in the next lecture. (Challenge the students to try to guess the construction.)

Historical Notes

The notion of one-way functions was introduced by Diffie and Hellman in their seminal work *New Directions in Cryptography* [6], which set the direction of the field for many years. The conjecture that factoring integers is intracatable was put forward by Rivest, Shamir and Adleman in their influential design of the RSA scheme [28]. Jumping ahead, I'd like to mention a third seminal paper that has actually set the tone for the entire approach followed in this course: I refer to the paper *Probabilistic Encryption* [22] of Goldwasser and Micali.

Lecture 2

Working with Intractability Assumptions

Summary

The bulk of this lecture is devoted to proving that if there exist weakly one-way functions then there exists strongly one-way functions. The construction itself is simple, but again the focus should be on the proof technique.

In addition, we introduce the notion of a collection of one-way functions, and the notion of a hard-core predicate of (a one-way function).

The title of the lecture emphasizes the non-triviality of deriving intractability results (such as the existence of strong one-way functions) based on intracatbility assumptions.

2.1 Weak one-way functions imply strong ones

Sec. 2.3

Recall the theorem stated at the end of last lecture.

Theorem 2.1 *If there exist weakly one-way functions, then there exists strongly one-way functions.*

Suppose that f is hard to invert (in probabilistic polynomial-time) on at least a $\rho = 1/\text{poly}(n)$ fraction of its probability space. Define F by partitioning the ($t \cdot n$-bit long) input string into $t \stackrel{\text{def}}{=} n/\rho$ equal-length parts and applying f to each of them. We claim that F is a strongly one-way function.

The proof DOES NOT amount to saying that *the probability of t successes in t trials, each succeeding with probability at most* $1-\rho$, *is* $(1-\rho)^t = \exp(-n)$. Such an "argument" assumes that the F-invertor operates independently on each of the t parts of the image. One may say that such an assumption is "reasonable", but what can be called reasonable when it comes to adversaries? Anyhow, we know of no way to justify this "reasonable" assumption, and we do not want any additional (ill-formed) assumption but rather a proof. The real proof proceeds by showing how any F-inverter that is too good can be transformed into an f-inverter that is too good.

Suppose towards the contradiction that F is easy to invert on at least an $\epsilon = 1/\text{poly}(n)$ fraction of the probability space. We will show that this yields that f is easy to invert on $1-\rho$ fraction of the probability space, which contradicts the hypothesis.

Specifically, we fix an inverting algorithm A for F, and proceed as follows. For $i = 1, ..., t$, call an f-preimage x i-good (or good in direction i) if the probability that A inverts F on the image $F(r_1, ..., r_{i-1}, x, r_{i+1}, ..., r_t)$ is at least $\epsilon/2t$, where the probability is taken over all possible choices of $r_1, ..., r_{i-1}, r_{i+1}, ..., r_t \in \{0,1\}^n$ and over A's coin tosses. (Note that x (and i) is fixed and the r_i's are random.)

Claim 2.2 *There exists an i such that at least $1-(\rho/2)$ fraction of the f-preimages are i-good.*

The proof is an easy counting argument. Assuming on the contrary that in all directions there are less than a $1-(\rho/2)$ fraction of good preimages, we upperbound the success probability of A by noting that non-good preimages contribute at most a probability mass of $\epsilon/2t$ in each direction, whereas the intersection of the good regions is exponentially vanishing. That's the combinatorial part.

We turn to the algorithmic part. The point is that, given $y = f(x)$, we may try to invert f on y as follows. For every $i = 1, ..., t$, using the F-invertor A, we try nt/ϵ times to invert F on $f(r_1), ..., f(r_{i-1}), y, f(r_{i+1}), ..., f(r_t)$, where $r_1, ..., r_{i-1}, r_{i+1}, ..., r_t$ are selected at random by us. If x happens to be i-good (for any i), then each of the corresponding trials has a probability of at least $\epsilon/2t$ to succeed. Trying nt/ϵ times, the probability that we fail in all trials is at most $(1-\epsilon/2t)^{nt/\epsilon} = \exp(-n)$. Combining this with Claim 2.2 (and letting i be as guaranteed by the claim), we conclude that for a random x, our f-inverying algorithm succeeds with probability at least

$$\begin{aligned} & \mathsf{Pr}[x \text{ is } i\text{-good}] \cdot (1 - \mathsf{Pr}[\text{we fail to invert } f \text{ on } f(x) \,|\, x \text{ is } i\text{-good}]) \\ \geq \;\; & \left(1 - \frac{\rho}{2}\right) \cdot (1 - \exp(-n)) \;\; > \;\; 1 - \rho \end{aligned}$$

and contradiction follows.

Digest: The crux of the entire proof is neither (the proof of) Claim 2.2 nor the *analysis* of the f-inverter (presented above). It is rather the f-inverter itself. Put in other words, the crux of the proof is in the way an F-inverter (which succeeds only with probability ϵ) can be used to derive an f-inverter (which succeeds with probability $1-\rho$). It should not be surprising that we have to invoke the former (i.e., F-invertor) many (i.e., nt/ϵ) times, because we are *amplifying* the success probability.

More thoughts: Emphasize the non-triviality of deriving intractability results (such as the existence of strong one-way functions) based on intracatbility assumptions (such as the existence of weak one-way functions). Indeed proving such implications is easier than proving the corresponding intractability results without making assumptions (currently way beyond our reach), but still the fact that we use assumptions means that we don't really understand the context (or else we would not have needed to assume anything). Thus, we should be "super-careful" in deriving implications. The only implications allowed are those that we can formally prove (and not merely conjecture or consider reasonable). Typically, implications of the above type are proven by counter-positive. Given an algorithm that violates the conclusion (that's nice: we are given something alas this thing is "generic"), we derive an algorithm that violates the hypothesis (and thus derive a contradiction). We construct the latter algorithm while using the former one as a sub-routine (because this algorithm is "generic"), and since the context is probabilitic it is important to invoke the sub-routine on input distributions about which we can make meaningful claims (regarding the sub-routine behaviour).

2.2 Collections of one-way functions

Sec. 2.4

The more cumbersome formulation is motivated in two ways. Firstly, it is more suitable for discussion of most popular candidates of one-way functions. Secondly, it allows a natural introduction of the notion of trapdoor (one-way functions).

The basic formulation refers to collections (of (finite) functions) $\{f_\alpha : D_\alpha \to R_\alpha\}_{\alpha\in I}$, where $I \subseteq \{0,1\}^*$. To be able to use such a collection we need efficient (uniform (for the collection)) algorithms for selecting indices in I (i.e., given 1^n, generate an element of $I \cap \{0,1\}^n$), for selecting domain elements (i.e., given $\alpha \in I$, generate an element of D_α), and for evaluating the function (i.e., given $\alpha \in I$ and $x \in D_\alpha$, output $f_\alpha(x)$). The one-way property has to be stated with respect to the distribution induced by the index and domain sampling algorithms.

Example: the RSA collection. We view it as a collection of one-way functions, not as an encryption and/or signature scheme. The reasons for this perspective will become clear when we define encryption and signature schemes.

Additional properties enjoyed by the RSA collection include the functions being permutations (over the corresponding domains), the sampling algorithms generating (almost) uniform samples from the corresponding sets, and – most importantly – having trapdoors that allow efficient inversion.

Trapdoor collections. The index-sampling algorithm is required to output a pair, where the first element is an index (as before) and the second is a corresponding trapdoor. In addition, an efficient algorithm is presented for inverting the function when given also the trapdoor. Inverting remains infeasible when not given the trapdoor but rather given the index of the function (which does allow efficient evaluation of the function).

2.3 Hard-core predicates (of one-way functions)

Sec. 2.5

We turn back to the less cumbersome formulation (in which there is a single infinite function rather than an infinite collection of finite functions).

We have seen (or show now) that a one-way function may "leak" many bits of its preimage (e.g., given a one-way function f consider the function $f'(x,y) = (x, f(y))$, where $|x| = |y|$). Going to the other extreme, we say that a simple (i.e., efficiently computable) predicate is a *hardcore of a function* if given a random image it is infeasible to guess (with non-negligible advantage) the predicate's value on "the" corresponding preimage. The interesting case is when the function is 1-1, otherwise hardcore may exists in a trivial (i.e., information-theoretic) manner (e.g., $f'(\sigma x) = (0, f(x))$ has the hardcore predicate $b(\sigma x) = \sigma$, regardless if f is hard to invert or not). In case the function is 1-1 (and efficiently computable), if it has a hardcore then it must be one-way (i.e., otherwise the "hardcore" is computed by inverting the function and computing the "hardcore" on the resulting preimage). In constrast, any (strongly) one-way function can be slightly modified to have a hardcore predicate. Specifically:

Theorem 2.3 *Suppose that f is a one-way function. Let $f'(x,r) = (f(x), r)$, and $b(x,r)$ be the inner-product mod 2 of x and r. Then b is a hardcore predicate of f'.*

The theorem will be proven in the next lecture. Note that f' preserves many properties of f (e.g., being 1-1, evaluation time, etc).

Historical Notes

Theorem 2.1 is due to Yao [30], and its proof has first appeared in [7]. Theorem 2.3 is due to Goldreich and Levin [16], and improves over a more complicated construction due to Yao. The latter construction is analyzed using the so-called *Yao's XOR-Lemma*, which is of independent interest. A proof of Yao's XOR-Lemma has first appeared in Levin's paper [25], but the interested reader is referred to other sources (e.g., a good place to start is [19]).

Lecture 3

Hard-Core Predicates and Pseudorandom Generators

Summary

This lecture consists of two independent parts. In the first (and main) part we present a proof of the existence of a generic hard-core predicate. In the second part we provide a short introduction to the notion of pseudorandom generators, which will be the focus of subsequent lectures.

The two parts are not unrelated: in a lecture (or two) we will see how to use hard-core predicates to construct pseudorandom generators.

3.1 Existence of a Generic Hard-core predicate

Sec. 2.5

Recall the definitions of (strong) one-way functions and hard-core predicate as well as the following result (stated in previous lecture and to be proven now).

Theorem 3.1 (Theorem 2.3, restated): *Suppose that f is a one-way function. Let $f'(x,r) = (f(x),r)$, and $b(x,r)$ be the inner-product mod 2 of x and r. Then b is a hardcore predicate of f'.*

Let A be any probabilistic polynomial-time algorithm and $\epsilon(n) \stackrel{\text{def}}{=} \mathsf{Pr}[A(f'(X_n,R_n)) = b(X_n,R_n)] - \frac{1}{2}$, where X_n and R_n are independently and uniformly distributed over $\{0,1\}^n$. That is, ϵ represents A's advantage over a random guess.

Define $s(x) \stackrel{\text{def}}{=} \mathsf{Pr}[A(f(x),R_n) = b(x,R_n)]$, so $\mathsf{E}[s(X_n)] = \frac{1}{2} + \epsilon(n)$. Define $S_n \stackrel{\text{def}}{=} \{x : s(x) > (1/2) + (\epsilon(n)/2)\}$ to be the set of x's for which A has a relatively significant advantage, and prove that $|S_n| > (\epsilon(n)/2) \cdot 2^n$. Focus on an arbitrary fixed x in S_n, and let $\epsilon = \epsilon(n)$.

The easy (alas unrealistic) case: Suppose that $s(x) > (3/4) + (\epsilon/2)$ (rather than $s(x) > (1/2) + (\epsilon/2)$). Then,

$$\mathsf{Pr}_r \left[A(f(x),r) \oplus A(f(x), r \oplus 0^{i-1}10^{n-i}) = b(x,r) \oplus b(x, r \oplus 0^{i-1}10^{n-i})\right] \;\geq\; 1 - 2 \cdot \left(\frac{1}{4} - \frac{\epsilon}{2}\right) \;=\; \frac{1}{2} + \epsilon$$

On the other hand, for $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ otherwise,

$$b(x,r) \oplus b(x, r \oplus 0^{i-1}10^{n-i}) \;\equiv\; \sum_{j=1}^{n} x_j r_j + \sum_{j=1}^{n} x_j (r_j + \delta_{i,j}) \;\equiv\; x_i \pmod 2$$

Thus, a sample of $\Theta(n/\epsilon^2)$ (pairwise) independent random r's will give us x_i with probability at least $1-(1/2n)$. Thus, with probability at least $1/2$, we correctly recover all bits of x (and so invert f on $f(x)$).

The real case: We only have $s(x) > (1/2)+(\epsilon/2)$. The *doubling of error* in the above procedure, makes it inadequate for the case $s(x) \leq 3/4$. To avoid the *error doubling* phenumena, let us dream. Suppose that somebody gave us the values of $b(x,r)$'s for sufficiently (i.e., $\Theta(n/\epsilon^2)$) many random r's. Then, it would suffice to query A on the corresponding $(f(x), r \oplus 0^{i-1}10^{n-i})$'s so to obtain guesses for $b(x, r \oplus 0^{i-1}10^{n-i})$. Thus, with probability $s(x) > (1/2) + (\epsilon/2)$, each such random r yields the value of x_i (by $A(f(x), r \oplus 0^{i-1}10^{n-i}) \oplus b(x,r)$), and ruling by majority we are again correct with probability at least $1-(1/2n)$. But how do we get the values of $b(x,r)$'s for $m \stackrel{\text{def}}{=} \Theta(n/\epsilon^2)$ random r's. The answer is that we are going to guess them! Certainly, if the r's were totally independent then we would be correct with probability 2^{-m} (which is way too low). Instead, we use (a specific construction of) m pairwise-independent random r's such that the probability that we correctly guess all $b(x,r)$'s is $1/\text{poly}(m)$ (which we can afford). Note that the majority rule works also for pairwise-independent random r's.

We construct m pairwise independent r's, based on $\ell = \lceil \log_2(m+1) \rceil$ independent random strings in $\{0,1\}^\ell$. Specifically, we uniformly and independently select $s^1, ..., s^\ell \in \{0,1\}^n$, and set $r^I = \oplus_{i \in I} s^i$ for every non-empty $I \subseteq [\ell]$. Thus, we obtain m pairwise independent r^I's.[1] We obtain a guess for all $b(x, r^I)$'s, by merely guessing all $b(x, s^i)$'s at random, and using

$$b(x, r^I) \;=\; b(x, \oplus_{i \in I} s^i) \;=\; \sum_{j=1}^{n} x_j \sum_{i \in I} s_j^i \;=\; \sum_{i \in I} \sum_{j=1}^{n} x_j s_j^i \;\equiv\; \sum_{i \in I} b(x, s^i) \pmod 2$$

Thus, if our initial ℓ guesses are correct (which happens with probability $2^{-\ell} \approx 1/m$) then the values obtained for all $b(x, r^I)$'s are correct.

3.2 Introduction to Pseudorandom Generators

$\boxed{\textbf{Sec. 3.1}}$

Loosely speaking, a pseudorandom generator is an *efficient* program (or algorithm) that *stretches* short random strings into long *pseudorandom* sequences. Thus, a pseudorandom generator does not "generate randomness" (from scratch) but rather expands small amounts of true randomness (present in the seed) into bigger objects that are not truly random but do appear as if they are. Emphasize the three fundamental aspects in the notion of a pseudorandom generator:

---

[1] To see that the r^I's are pairwise independent consider any $I \neq J$ and any $\alpha, \beta \in \{0,1\}^n$. We need to prove that $\mathsf{Pr}[r^I = \alpha \ \& \ r^J = \beta] = (1/2^n)^2$. Suppose (w.l.o.g) that $k \in I \setminus J$, and write

$$\mathsf{Pr}[r^I = \alpha \ \& \ r^J = \beta] \;=\; \mathsf{Pr}[r^J = \beta] \cdot \mathsf{Pr}[r^I = \alpha \mid r^J = \beta]$$

Clearly, $\mathsf{Pr}[r^J = \beta] = \mathsf{Pr}[\oplus_{j \in J} s^j] = 2^{-n}$. By fixing all s^j's except for s^k that is kept random, we have $\mathsf{Pr}[r^I = \alpha \mid r^J = \beta] = \mathsf{Pr}[s^k = \alpha \oplus (\oplus_{i \in I \setminus \{k\}} s^i)] = 2^{-n}$.

1. Efficiency of the generator.
2. Stretching: Short seeds are stretched into longer output sequences. Specifically, n-bit long seeds are stretched into $\ell(n)$-bit long outputs, where $\ell(n) > n$ (and typically $\ell(n) \gg n$).
3. Pseudorandomness: The longer ($\ell(n)$-bit long) outputs (produced based on uniformly distributed n-bit seeds) are *computationally indistinguishable* from uniformly distributed sequences of the same length (i.e., of length $\ell(n)$).

Indeed, the notion of computationally indistinguishable is the heart of the definition and will be discussed extensively in the next lecture.

Pseudorandom generators allow to generate, communicate and store $\ell(n)$-bit long random sequences at the cost of actually generating, communicating and storing only n (random) bits.

Historical Notes

Theorem 3.1 is due to Goldreich and Levin [16], but their original proof (see generalization in [21]) is different that the proof presented above. The proof presented above, which follows ideas that originate in [1], was discovered later (and independently) by Levin and Rackoff.

Although pseudorandom generators have been used and referred to (implicitly and explicitly) from the early days of computer science, a rigorous definition of the notion was first put forward by Blum and Micali [5].

Lecture 4

Computational Indistinguishability

Summary

The focus of this lecture is on the concept of computational indistinguishability, which is a key concept in the area, and in particular underlies the notion of pseudorandomness. Specifically, we present the basic definition, discuss its relation to statistical indistinguishability, consider its preservation under repeated sampling, and present the hybrid technique (which is often used towards proving computational indistinguishability). We then recall the definition of pseudorandom generator and prove that a construction with any *given* stretching factor yields constructions for any *desired* stretching factor.

4.1 The basic notion and some basic properties

Sec. 3.2

4.1.1 The actual definition

The intuition: An efficient procedure (or observer) is given either a sample from one distribution or a sample from the other. Can he/she distinguish the two cases?

The formalism: We cannot talk about a finite distribution (because an efficient machine can incorporate it), but rather of *distribution ensembles*; that is, infinite sequences $\{Z_n\}_{n\in\mathsf{N}}$, where each Z_n is finite random variable (or distribution). Typically, Z_n ranges over $\{0,1\}^n$ (or over $\{0,1\}^{p(n)}$, for some polynomial p that is fixed for the entire ensemble).

For an observer A, consider the sequence of $p_n \stackrel{\rm def}{=} \mathsf{Pr}[A(X_n) = 1]$ versus the sequence of $q_n \stackrel{\rm def}{=} \mathsf{Pr}[A(Y_n) = 1]$. (Spell out what these probabilities mean; that is, write $\mathsf{Pr}[A(X_n) = 1] = \sum_x \mathsf{Pr}[X_n = x] \cdot \mathsf{Pr}[A(x) = 1]$, students often have problems with this.) If $|p_n - q_n|$ is negligible (as a function of n), we say that A does not distinguish between $\{X_n\}_{n\in\mathsf{N}}$ and $\{Y_n\}_{n\in\mathsf{N}}$.

We say that $X = \{X_n\}_{n\in\mathsf{N}}$ and $Y = \{Y_n\}_{n\in\mathsf{N}}$ are computationally indistinguishable if no probabilistic polynomial-time distinguishes them.

Computational indistinguishability is a relaxation of statistical indistinguishability defined as having negligible variation distance. Exercise: Show that statistical indistinguishability implies computational indistinguishability. (Comment: the relaxation is strict if and only if one-way functions exist. More about this, at a later stage.)

Comment: Distinguishing fixed objects (rather than distributions) is a special case.

4.1.2 Repeated samples

Suppose that $X = \{X_n\}_{n \in \mathsf{N}}$ and $Y = \{Y_n\}_{n \in \mathsf{N}}$ are computationally indistinguishable. Does this means that an efficient observer cannot distinguish two independently selected samples of X from two independently selected samples of Y?

In general, the answer is NO, but for the special case we care about in which both ensembles are "efficiently sampleable" the answer is YES. (An ensemble $Z = \{Z_n\}_{n \in \mathsf{N}}$ is efficiently sampleable if there exists an a probabilistic polynomial-time S such that $S(1^n)$ is distributed identically to Z_n.)

The negative part should teach us a lesson: not to rush to conclusions regarding complex notions.

The positive part is proven next. The proof utilizes a central (new) technique, to be called a "hybrid argument". Let us restate our claim (for X and Y as above): For every polynomial t, consider the t-product of $Z = \{Z_n\}_{n \in \mathsf{N}}$

$$\{Z_n^{(1)}, Z_n^{(2)}, ..., Z_n^{(t(n))}\}_{n \in \mathsf{N}}$$

where the $Z_n^{(i)}$'s are independent copies of Z_n. We claim that if X and Y are computationally indistinguishable then so are their t-products (for every polynomial t). Suppose that A distinguishes the t-products; that is, there exists a polynomial p and infinitely many n's such that

$$\left|\Pr[A(X_n^{(1)}, X_n^{(2)}, ..., X_n^{(t(n))}) = 1] - \Pr[A(Y_n^{(1)}, Y_n^{(2)}, ..., Y_n^{(t(n))}) = 1]\right| > \frac{1}{p(n)}$$

(Verify that this is the negation, and that we may drop the absolute value.) Consider such a generic n. Then, there exists an $i \in \{0, ..., t(n) - 1\}$ such that

$$\begin{aligned} &\Pr[A(X_n^{(1)}, ..., X_n^{(i)}, \boxed{X_n^{(i+1)}}, Y_n^{(i+2)}, ..., Y_n^{(t(n))}) = 1] \\ - \;&\Pr[A(X_n^{(1)}, ..., X_n^{(i)}, \boxed{Y_n^{(i+1)}}, Y_n^{(i+2)}, ..., Y_n^{(t(n))}) = 1] \;>\; \frac{1}{t(n) \cdot p(n)} \end{aligned}$$

That is, for every j, we consider a hybrid distribution composed of j copies of X_n followed by $t(n) - j$ copies of Y_n. The above inequality asserts that A distinguishes the $i+1$st hybrid from the ith hybrid. (It follows by observing that the t-product of X coincides with the last (i.e. $t(n)^{\text{th}}$) hybrid, whereas the t-product of Y coincides with the 0-hybrid.) In fact, the expected gap for a random pair of neighboring hybrids is also $1/t(n) \cdot p(n)$. *Using the sampleability of both ensembles*, we convert A into an observer that distinguishes a single copy of X from a single copy of Y (in contradiction to the hypothesis).

Comment: Some students may wonder as to *why should A, which is supposed to run on product distributions, agree* (or behave nicely) on intermediate hybrids. Address the issue explicitly saying that A being merely an algorithm is not asked to agree but is rather being run on arbitrary inputs and its behavior on various distribution is well-defined. (In fact, if it behave "non-nicely" on some hybrids then even better...)

Hybrid technique – digest. The key features are defining a *polynomially-bounded number* of hybrids such that *the extreme hybrids correspond to the conclusion*, whereas *neighboring hybrids correspond to the hypothesis.* Elaborate on each of these three aspects.

4.2 Back to the definition of pseudorandom generators

Sec. 3.3

Recall that a pseudorandom generator is an *efficient* deterministic program (or algorithm) that *stretches* short random strings into long *pseudorandom* sequences, where pseudorandom ($\ell(n)$-bit long) sequences are defined as computationally indistinguishable from uniformly distributed sequences of the same length (i.e., of length $\ell(n)$).

Stretching: It is required that short seeds are stretched into longer output sequences. Specifically, n-bit long seeds are stretched into $\ell(n)$-bit long outputs, where $\ell(n) > n$ (for some polynomial ℓ). Indeed, typically we want $\ell(n) \gg n$, but in the definition we only required $\ell(n) > n$. This gap is addressed by the following result:

Theorem 4.1 *Suppose that G is a pseudorandom generator stretching seeds of length n to outputs of length $n+1$. Then, for every polynomial ℓ* (such that $\ell(n) > n$)*, there exists a pseudorandom generator G' stretching seeds of length n to outputs of length $\ell(n)$.*

The desired G' is constructed as follows. On input $s \in \{0,1\}^n$, we proceed in $\ell \stackrel{\text{def}}{=} \ell(n)$ iterations starting with $s_0 \stackrel{\text{def}}{=} s$, and outputting one bit in each iteration. In the ith iteration we output the first bit of $G(s_{i-1})$ and set s_i to equal the other n bits of $G(s_{i-1})$. (Draw a picture!)

To prove the pseudorandomness of G', we consider the following hybrids. For $i = 0, 1...., \ell(n)$, the ith hybrid outputs a uniformly distributed i-bit long string followed by the $(\ell(n) - i)$-bit prefix of $G'(U_n)$. Clearly, the 0-hybrid coincides with $G'(U_n)$, whereas the $\ell(n)$-hybrid coincides with $U_{\ell(n)}$. To utilize potential gaps between neighboring hybrids, we review the structure of the i-hybrid versus the $(i+1)$-hybrid. (Draw a picture!) For $x \in \{0,1\}^{n+1}$, let $\texttt{lead}(x)$ denote the first (leading) bit of x and $\texttt{rest}(x)$ denote the following n bits; that is, $x = \texttt{lead}(x) \cdot \texttt{rest}(x)$. View the i-hybrid as starting with a uniformly distributed i-bit long string followed by $\texttt{lead}(G(U_n))$ followed by the $(\ell(n) - i - 1)$-bit prefix of $G'(\texttt{rest}(G(U_n))$. Likewise, view $(i+1)$-hybrid as starting with a uniformly distributed i-bit long string followed by $\texttt{lead}(U_{n+1})$ followed by the $(\ell(n) - i - 1)$-bit prefix of $G'(\texttt{rest}(U_{n+1})$. Thus, distinguishing these two neighboring hybrids allows to distinguish $G(U_n)$ from U_{n+1}.

Historical Notes

The notion of computational indistinguishability was introduced in the work of Goldwasser and Micali [22] within the special context of indistinguishability of ciphertexts (as a definition of security for encryption schemes). The general notion of computational indistinguishability has first appeared in Yao's work [30], where it was used to define pseudorandom sequences (as being computationally indistinguishable from truly random sequences). Yao also proved that this notion of pseudorandom sequences is equivalent to the notion of unpredictable sequences (which was used as the notion of pseudorandomness in the work of Blum and Micali [5]).

Lecture 5

Constructing Pseudorandom Generators and Functions

Summary

In the first part of this lecture we show how to construct pseudorandom generators using any 1-1 one-way function (i.e., one-way permutation). In the second part, we define pseudorandom functions and show how they can be constructed using any pseudorandom generator.

5.1 Constructing Pseudorandom Generator

Sec. 3.4

Recall the definition of a pseudorandom generator, and that pseudorandom generators with minimal stretch suffice for constructing pseudorandom generators with arbitrary (polynomial) stretch.

Letting f be a 1-1 one-way function, and b be a corresponding hard-core predicate, we prove that $G(s) \stackrel{\text{def}}{=} f(s)b(s)$ is a pseudorandom generator. That is, we prove

Lemma 5.1 *$\{G(U_n)\}_{n\in\mathsf{N}}$ and $\{U_{n+1}\}_{n\in\mathsf{N}}$ are computationally indistinguishable.*

Define $G'(s) \stackrel{\text{def}}{=} f(s)\overline{b}(s)$, where $\overline{b}(s) \stackrel{\text{def}}{=} 1-b(s)$. The key observation is that $U_{n+1} \equiv f(U_n)U_1'$ equals $G(U_n) = f(U_n)b(U_n)$ with probability $1/2$ and equals $G'(U_n) = f(U_n)\overline{b}(U_n)$ otherwise. Thus, for every algorithm D,

$$\mathsf{Pr}[D(G(U_n)) = 1] - \mathsf{Pr}[D(U_{n+1}) = 1] \;=\; \frac{\mathsf{Pr}[D(G(U_n)) = 1] - \mathsf{Pr}[D(G'(U_n)) = 1]}{2}$$

and it suffices to show that $\{G(U_n)\}_{n\in\mathsf{N}}$ and $\{G'(U_n)\}_{n\in\mathsf{N}}$ are computationally indistinguishable. The latter is proven by contradiction; that is, (w.l.o.g.) suppose that

$$\delta(n) \stackrel{\text{def}}{=} \mathsf{Pr}[D(G(U_n)) = 1] - \mathsf{Pr}[D(G'(U_n)) = 1] \geq 0$$

then given $y = f(x)$ we predict $b(x)$ by uniformly selecting $\sigma \in \{0,1\}$ and outputting σ if $D(y\sigma) = 1$ (and $\overline{\sigma}$ otherwise). Intuitively, $D(y\sigma)$ is more likely to be 1 if $\sigma = b(x)$. Formally, we just evaluate the success probability of this prediction rule, and conclude that it is correct with probability $\frac{1}{2} + \frac{\delta(n)}{2}$.

An alternative construction: By combining the above construction with the transformation shown in the previous lecture (and mentioned above), we obtain

Theorem 5.2 *If 1-1 one-way functions exist then, for every polynomial ℓ such that $\ell(n) > n$ $(\forall n)$, there exist a pseudorandom generator with stretch function ℓ.*

A direct construction (which is in fact equivalent to the one obtained by combining the two above-mentioned constructions), follows. Let f be a 1-1 one-way function and b be a corresponding hard-core, then $G(s) \stackrel{\text{def}}{=} b(s)b(f(s))b(f^2(s))\cdots b(f^{\ell(n)-1}(s))$ is a pseudorandom generator. The proof is obtained by combining the proof ideas used to establish the validity of the two corresponding constructions.

Getting rid of the 1-1 condition: In fact, pseudorandom generators can be constructed using any one-way function (rather than only 1-1 one-way functions), alas the construction (and more so its analysis) is too complex to be presented here. One the other hand, it is easy to show that the existence of pseudorandom generators implies the existence of one-way functions (e.g., if $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a pseudorandom generator, then for $|x| = |y|$ the function $f(xy) = G(x)$ is one-way (hint: U_{2n} is unlikely to have a preimage under f)). Thus:

Theorem 5.3 *Pseudorandom generators exist if and only if one-way functions exist.*

5.2 Pseudorandom Functions

Sec. 3.6

Pseudorandom generators yield distributions with support size of at most 2^n that are computationally indistinguishable from the uniform distribution on $2^{\text{poly}(n)}$ objects. Pseudorandom functions (defined below) yield distributions with support size of at most 2^n that are computationally indistinguishable from the uniform distribution on 2^{2^n} objects. For example, a distribution of up-to 2^n many functions $\{0,1\}^n \to \{0,1\}$ is computationally indistinguishable from a random function $\{0,1\}^n \to \{0,1\}$, where computational indistinguishability means failure of any efficient test that may obtain the value of the function at inputs of its choice to distinguish the two cases.

Formalism: Here the distinguisher is an oracle machine given oracle access to a function. For example, we may consider collections $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$ satisfying the following two conditions:

1. Efficient evaluation: There exist an efficient algorithm that given a function description s and an argument x returns $f_s(x)$.

2. Pseudorandomness: For every probabilistic polynomial-time oracle machine M,

$$|\mathsf{Pr}[M^{f_{U_n}}(1^n) = 1] - \mathsf{Pr}[M^{F_n}(1^n) = 1]|$$

   is negligible, where $F_n : \{0,1\}^n \to \{0,1\}^n$ denotes a random function.

Clearly, pseudorandom functions yields pseudorandom generators (e.g., $G(s) = f_s(1)f_s(2)\cdots f_s(t)$). We show a construction establishing the converse.

Theorem 5.4 *Pseudorandom functions exist if and only if pseudorandom generators exist.*

Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a pseudorandom generator, and let $G_0(s)$ (resp., $G_1(s)$) denote the first (resp., last) $|s|$ bits of $G(s)$. For $s, x \in \{0,1\}^n$, consider

$$f_s(x) \stackrel{\text{def}}{=} s_x \stackrel{\text{def}}{=} G_{x_n} \cdots G_{x_2} G_{x_1}(s)$$

where $x = x_1 x_2 \cdots x_n$. (Draw a picture of a binary tree where the root is labeled by $s_\lambda \stackrel{\text{def}}{=} s$, and the children of the node labeled s_α are labeled $s_{\alpha 0} \stackrel{\text{def}}{=} G_0(s_\alpha)$ and $s_{\alpha 1} \stackrel{\text{def}}{=} G_1(s_\alpha)$, respectively.)

Prove the validity of the construction using a hybrid argument, where the i^{th} hybrid corresponds to a tree with random labels at the 2^i vertices of level i (and their ancestors being labeled as above). Hint: use the fact that, for any polynomial t, the t-product of $\{G(U_n)\}_{n \in \mathbb{N}}$ and the t-product of $\{U_{2n}\}_{n \in \mathbb{N}}$ are indistinguishable, and use samples from either $\{G(U_n)\}_{n \in \mathbb{N}}$ or $\{U_{2n}\}_{n \in \mathbb{N}}$ to construct either the i^{th} or $i + 1^{\text{st}}$ hybrid "on the fly".

Applications: Pseudorandom functions enable to share a random-looking function at the cost of selecting, communicating and storing a short seed. The seed is shared by the legitimate parties and a random behavior (of the function) is guaranteed even if an adversary can obtain values of the function at arguments of its choice. Immediate applications include

1. Private-key encryption schemes.
2. Message Authentication Codes.
3. Identify Friend Or Foe schemes.

The first two applications will be discussed in subsequent lectures. The third application refers to challenge–response identification protocols, where the challenger selects a random challenge and the respond should be the value of the agreed (pseudorandom) function at this challenge point. (Only legitimate group members know the agreed secret function.) Note that an efficient adversary cannot pass such a test even if prior to attempting, the adversary may play the role of the challenger (w.r.t a legitimate member) in polynomially-many instances of the protocol (and so obtain the value of the function at polynomially-many arguments of its choice).

Historical Notes

Theorem 5.2 is due to Yao [30]. The alternative proof of Theorem 5.2 coincides with the original construction of Blum and Micali [5]. Theorem 5.3 is due to Håstad, Impagliazzo, Levin and Luby [24].

Pseudorandom functions were defined and constructed (as in Theorem 5.4) by Goldreich, Goldwasser, and Micali [15].

Lecture 6

Zero-Knowledge Interactive Proofs

Summary

We first motivate the notion of zero-knowledge. Next, we define, illustrate and discuss the natural notion of interactive proof systems (which is the adequate framework for the introduction of zero-knowledge proofs). Next we use this framework to define and illustrate the notion of zero-knowledge. The illustrative examples used are the Graph Non-Isomorphism and Graph Isomorphism protocols, respectively.

6.1 Motivation

Sec. 4.1

Discuss (with or without examples) the archetypical cryptographic problem of forcing proper behavior of a party without requiring it to loss anything (in case it is honest). Note that the party's behavior depends on its secrets, which may be at least partially verifiable (via previous actions taken by the party). If the party was to reveal its secret (which is, of course, out of the question) then one could easily verify that this party acts properly. We want to be able to verify the latter fact without making the party reveal its secret (nor even partial information about the secret). That is, the verifier should learn nothing from the proof beyond the fact that the claim (of proper behavior) is valid.

The issues at hand, are

1. Presenting a (rich) framework for proofs of proper (or secret-consistent) behavior.

   This will lead us to the definition of interactive proof systems.

2. Formulating what it means to learn nothing from such proofs.

   This will lead us to the definition of zero-knowledge.

6.2 Interactive Proof Systems

Sec. 4.2

The original motivation to the definition of interactive proof systems is that the traditional notion of (written) proofs, as captured by the class $\mathcal{NP}$, does not allow zero-knowledge proofs. However, in retrospect, since proofs are going to be used inside bigger cryptographic protocols, there seems to be no reason to insist that they be uni-directional (since bi-directional communication may anyhow occur) and deterministically-verifiable (since all cryptography is anyhow "randomized").

Interactive proof systems capture the most general way in which a probabilistic polynomial-time party may be convinced of the validity of some assertion. Present the definition while relying on the intuitive notion of strategies (rather than on the cumbersome formulations of interactive Turing machines, just as one needs not define Turing machines to discuss algorithms). Indeed, uni-directional deterministically-verifiable proofs (i.e., "NP-type" proofs) are a special case.

Issues to be discussed include the *completeness* and *soundness* requirements (stated with respect to negligible error probability), error-reduction (easily established via sequential repetition but does hold *here* also via parallel repetition), and the discrepancy in computational power (of the prover versus the verifier). Stress that we will focus (e.g., in the course) on prover strategies (for the completeness condition) that are implementable in probabilistic polynomial-time when given an adequate auxiliary input (e.g., typically an NP-witness).

To illustrate the notion, present the Graph Non-Isomorphism (GNI) protocol (but avoid a detailed analysis, which refers to the automorphism group of the graphs). Stress that you show an interactive proof for a language not known to be in $\mathcal{NP}$ (i.e., not known to have an "NP-type" proof). You may further refer to the above issues in light of that protocol.

6.3 Zero-Knowledge

Sec. 4.3

The simulation paradigm is the way we capture the notion of "gaining nothing beyond something". It amounts to saying that whatever can be efficiently done with the alleged gain, can also be efficiently done with the original something (which is claimed to be the upper bound of gain). This is a general paradigm. In our context it says that whatever can be efficiently computed after (efficiently) interacting with a zero-knowledge prover, can be efficiently computed from the assertion itself (provided it is valid). Thus, the only gain from the interaction is building confidence in the validity of the assertion (and "nothing beyond" is gained).

Discuss the over-simplified formulation (of perfect zero-knowledge) and its (minor) relaxation that allows the simulator to have no output with negligible probability. Show that languages in $\mathcal{BPP}$ have a (trivial) zero-knowledge proof (in which the prover remains silent...). A non-trivial illustration of zero-knowledge can be given by presenting the Graph Isomorphism (GI) protocol, showing both the protocol and its simulator (but avoiding a detailed analysis). Stress that you show an interactive proof for a language in $\mathcal{NP}$, but that this "peculiar" proof system (unlike the "NP-type" proof) can be shown to be zero-knowledge.

You may comment that the formulation of interactive proofs is essential to the non-triviality of zero-knowledge (i.e., only languages in $\mathcal{BPP}$ have uni-directional zero-knowledge proofs). (The proof may be left as an exercise; see Theorem 4.5.1 in the book.)

Historical Notes

Interactive proofs and zero-knowledge were introduced in the seminal work of Goldwasser, Micali and Rackoff [23]. We stress that their motivation for introducing interactive proofs was to formulate the most general notion of what can be efficiently verifiable. (In contrast, Babai's motivation for introducing Arthur–Merlin games [2], which turned out to be as powerful as interactive proofs, was to "slightly extend the class $\mathcal{NP}$" such that it includes a specific computational problem that was known to be in $\mathcal{NP}$ (only) under a reasonable group-theoretic conjecture.)

The protocols for GNI and GI, illustrating the notions of interactive proofs and zero-knowledge proofs, are taken from the work of Goldreich, Micali and Wigderson [17].

Lecture 7

Constructing Zero-Knowledge Proofs

Summary

We recall the basic conditions underlying the definitional framework of zero-knowledge proofs, and present the actual definition that is typically used. We claim (and sketch a proof) that this definition is preserved under sequential composition. We define and show how to construct commitment schemes, and using the latter we show how to construct zero-knowledge proof systems for every language in $\mathcal{NP}$.

7.1 Computational Zero-knowledge

$\boxed{\text{Sec. 4.3}}$

Recall the basic three requirements from a zero-knowledge interactive proof:

1. Completeness is a viability condition referring to what happens if everybody behaves properly. In such a case, the prover only claims a valid statement and succeeds to make the verifier accept it.

2. Soundness protects the verifier from attempts of a cheating prover to prove false assertions. In such a case, the verifier will reject (with high probability) no matter what the prover does in order to cheat.

3. Zero-knowledge protects the prover from attempts of a cheating verifier to learn more than the validity of an assertions: No matter what a (feasible) verifier does in order to gain extra knowledge, it can only obtain what it could obtain by itself when assuming (or believing) that the assertion is indeed valid.

The set of valid assertions is associated with a language $L \subseteq \{0,1\}^*$.

The actual definition – first issue. We will refer to a naturally relaxed definition of zero-knowledge, which suffices for typical cryptographic purposes. Rather than asking that the simulation is perfect (i.e., distributed identically to the real interaction), we only require that the two probability ensembles be computational indistinguishable. (Stress that the ensembles are indexed by strings; typically, members of L as above.)

The actual definition – second issue. On the other hand, we strengthen the definition by requiring that zero-knowledge holds with respect to any a-priori information, which is captured by an auxiliary input to the prover. This strengthening is important because it allows to use zero-knowledge proofs inside larger protocols (i.e., the original motivation), where auxiliary input captures information obtained before the zero-knowledge proof is invoked. In particular, auxiliary inputs are important for sequential composition of zero-knowledge. (Stress that, so far, typical zero-knowledge proofs were also zero-knowledge with respect to auxiliary input; but this may change in the future following Barak's recent result [3].)

Lemma 7.1 (loosely stated): *Zero-knowledge with respect to auxiliary input is preserved under sequential composition.*

Postpone the proof of this lemma to the end of the lecture, and sketch it only if time permits. Stress that closure under parallel composition does not necessarily hold, which may be a warning to those happy to rely on unsound intuitions.

7.2 How to construct zero-knowledge proofs for NP

Sec. 4.4

Start with an abstract description (i.e., referring to locked boxes) of the 3-colorability proof system. You may actually "play" the protocol in class (using 1+6 slides[1], if you are old-fashioned like me, or a power-point...). Establish perfect completeness and "pitiful" (yet noticeable) soundness. Present (and analyze) the (abstract) simulator.

The protocol is to be (sequentially) repeated to obtain a reasonable soundness bound; stress that each repetition calls for independent coin tosses (and so although there are only 6 possible relabellings of the colors, a random one is selected independently at each repetition).

Actual (digital) implementation. This calls for a (digital) implementation of "locked boxes", which leads to the notion of commitment schemes. Define this notion (referring to "hiding" and "binding" after the commit phase), and present a simple construction based on 1-1 one-way functions. State and discuss (as time permits) the proof of the following

Theorem 7.2 (loosely stated): *Assuming the existence of commitment schemes, there exists a zero-knowledge proof system for 3-Colorability.*

Important comments:

- The proper prover (in the above proof system) can be implemented in probabilistic polynomial-time, provided it is given a 3-coloring as an auxiliary input.
- Zero-knowledge holds also with respect to auxiliary input (given to the verifier).

[1]Draw the graph on the first slide and cover the vertices (drawn as small cycles) by small removable pieces of non-transparent material. The extra 6 slides show 6 random colorings obtained from one canonical 3-partition of the graph into independent sets. Select one of these slides at random and place it behind the master slide, inviting the students to specify an edge, and revealing the colors of its endpoints (by removing the corresponding pieces).

- The fact that 3-colorability has a zero-knowledge proof does imply that so does every language in NP; but there are minor subtleties in this implication (i.e., one has to rely either on the fact that zero-knowledge holds also with respect to auxiliary input or on properties of several relevant reductions).

The above three comments are essential to the wide applicability of the construction. Stress that a typical claim regarding proper behavior of one party (w.r.t verifiable secrets) is an NP-assertion, and that the acting party knows the relevant NP-witness (and thus can play the prover role in probabilistic polynomial-time).

Historical Notes

Zero-knowledge was introduced and defined by Goldwasser, Micali and Rackoff [23]. The original definition was augmented with auxiliary inputs in [20], where the Sequential Composition Lemma (i.e., Lemma 7.1) was proven.

Theorem 7.2 (implying that (using commitment schemes) zero-knowledge proofs can be constructed for any language in $\mathcal{NP}$) was proved by Goldreich, Micali and Wigderson [17]. Our presentation follows theirs.

Lecture 8

Defining Security of Encryption Schemes

Summary

After presenting the basic syntax, we define two equivalent notions of security: semantic security and the technical definition of indistinguishability of encryptions. We prove the equivalence of the two definitions, and consider their generalization to the encryption of several plaintexts (under the same key). We discuss the inherent role of probabilistic encryption algorithms (for satisfying the definitions).

8.1 The Basic Setting

Sec. 5.1

Depending on the background of the student and on what was done in the introductory lecture, describe and/or recall the basic setting: an insecure communication channel tapped by an adversary and two parties that wish nevertheless to communicate in secrecy using that channel. Argue that the receiver must know something that the adversary does not know, and call this thing a *key*. The key is used to decrypt encrypted-messages, called *ciphertexts*, using a method that without loss of generality may be assumed to be universal (i.e., known to all). Argue that the encryption method must use a (corresponding) key, and conclude by mentioning a method of generating key-pairs. Together, these three methods (or randomized algorithms) constitute an encryption scheme (with a minimal syntactic requirement that "decryption works").

Point out that the above discussion does not mandate that the encryption-key equals the decryption-key, although in traditional schemes this is the case. In such a case, the "key distribution problem" arises (and is traditionally solved via an expensive secret channel). Introduce the notion of public-key encryption schemes, and show how it trivially solves the "key distribution problem". Stress that the difference between private-key and public-key is reflected in the definition of security, which is the main subject of the current lecture.

8.2 Definitions of Security

Sec. 5.2

8.2.1 Semantic Security and Indistinguishability of Encryptions

Introduce the definition of *semantic security* first, emphasizing its natural appeal. Comment that the definition fits into the simulation paradigm. Stress that it requires much more than merely the infeasibility of recovering the entire plaintext, and justify why the latter does not suffice. Stress that the difference between the public-key and the private-key model amounts to whether or not the (real) adversary gets the encryption-key as input.

Note that (unrestricted) partial information function h provides you with all the non-uniformity you need (for the rest of the presentation), and there is no need to consider non-uniform circuits in this definition. (The letter h stands for *history*, but some may prefer to use ℓ for *leak*.)

Advanced comment: Actually, talking in terms of non-uniform circuits will only complicate things, because you may need to require that the description of the benign adversaries (also represented by non-uniform circuits) should be efficiently computable from the description of the real adversaries. Not making this requirement yields a definition that does not seem satisfactory, because it may take much more effort to find a benign adversary than to find a real one. The same issue may arise also in case the adversaries are represented by uniform algorithms, but seem much less acute in the latter case.

Turning to *indistinguishability of encryptions*, stress that this is a technical definition, which is useful because it is equivalent to semantic security while being easier to work with. Here we explicitly use non-uniformity in the formulation (i.e., the distinguishing circuits are non-uniform). Refer again to the difference between the public-key and the private-key models.

8.2.2 Equivalence of the two definitions

The interesting (i.e., useful) direction is the fact that indistinguishability of encryptions implies semantic security. This is shown by explicitly constructing the benign adversary (which merely invokes the real adversary on an encryption of a dummy value). Stress that the benign adversary generates a key-pair by itself (and thus can work as stated also in the private-key model). Next, show that the benign adversary performs essentially as well as the real one, because indistinguishability of encryptions essentially guarantees that the real adversary cannot distinguish the real situation (when it gets an encryption of the plaintext) from the simulated one (when it gets an encryption of a dummy value). Extensive use of non-uniformity makes the implementation of this proof idea quite easy.

The other direction is essentially easier because in a sense indistinguishability of encryptions is a special case of semantic security in which the message distribution is uniform over two strings. The proof is slightly complicated by the fact that we have to accommodate non-uniform distinguishers (as in indistinguishability of encryptions) by "uniform distinguishers" (provide by semantic security): here is where the non-uniformity of h comes to our rescue (i.e., we define $h(x)$ to provide the circuit used to distinguish between the encryptions of the specific pair of $|x|$-long bit plaintexts).

8.2.3 Probabilistic Encryption

Show that, in the public-key model, a secure encryption scheme must employ a probabilistic encryption algorithm. (Use the technical definition (of indistinguishability of encryption).)

8.2.4 Security of Multiple Encryptions

Discuss the extension of the two definitions to multiple messages (using the same key), and suggest the following three exercises:

1. Providing a formulation of both extensions, and proving their equivalence.
2. Proving that in the public-key model the single-message formulation implies the multiple-message one.

   Hint: use the technical definition (of indistinguishability of encryption). Note that it may be dreadful to prove this implication while using the semantic security formulation (without passing through the technical definition).
3. Show that in the private-key model the single-message formulation is strictly weaker than the multiple-message one.

   Hint: Show that deterministic encryption algorithm suffices for secure (private-key) encryption of a single message, but cannot provide security for encryption of two messages.

We'll discuss the third exercise in the next lecture.

Historical Notes

The central role of complexity theory in cryptography can be traced to Shannon's seminal work [29]: Demonstrating the inherent limitations of the information theoretic notion of security, Shannon concluded that one should settle for a computational relaxation of the secrecy condition. That is, rather than requiring that the ciphertext yields no information on the plaintext, one has to require that such information cannot be efficiently computed from the ciphertext.

The notion of public-key encryption scheme was introduced by Diffie and Hellman [6]. First concrete candidates were suggested by Rivest, Shamir and Adleman [28] and by Merkle and Hellman [27]. However, satisfactory definitions of security were presented only a few years afterwards, by Goldwasser and Micali [22]. Indeed, the latter work is the basis of the entire rigorous approach to cryptography (presented in the current book). The paper's title ("Probabilistic Encryption") is due to the authors' realization that public-key encryption schemes in which the encryption algorithm is deterministic cannot be secure in the sense defined in their paper.

Technically speaking, the two definitions we presented are somewhat different from the two corresponding definitions in [22] (although all these variants are equivalent). Specifically, our formulation of semantic security follows the later formulations of the simulation paradigm by Goldwasser, Micali and Rackoff [23]. (The adaptation has first appeared in [7, 8].)

Bibliography

[1] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr. RSA/Rabin Functions: Certain Parts are As Hard As the Whole. *SIAM J. on Comput.*, Vol. 17, April 1988, pages 194–209.

[2] L. Babai. Trading Group Theory for Randomness. In *17th STOC*, pages 421–420, 1985.

[3] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[4] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme which hides all partial information. In *Crypto'84*, LNCS 196, Springer-Verlag, pages 289–302.

[5] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. on Comput.*, Vol. 13, pages 850–864, 1984.

Preliminary version in *23rd FOCS*, 1982.

[6] W. Diffie, and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Info. Theory*, IT-22 (Nov. 1976), pages 644–654.

[7] O. Goldreich. *Foundation of Cryptography – Class Notes.* Preprint, Spring 1989.

Superseded by the combination of [14] and [12, 13, 11].

[8] O. Goldreich. A Uniform Complexity Treatment of Encryption and Zero-Knowledge. *Journal of Cryptology*, Vol. 6, No. 1, pages 21–53, 1993.

[9] O. Goldreich. *Foundation of Cryptography – Fragments of a Book.* February 1995. Available from `http://theory.lcs.mit.edu/~oded/frag.html`

Superseded by the combination of [14] and [12].

[10] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), *Springer*, 1999.

[11] O. Goldreich. *Secure Multi-Party Computation.* In preparation, 1998. Working draft available from `http://theory.lcs.mit.edu/~oded/gmw.html`

[12] O. Goldreich. *Encryption Schemes – fragments of a chapter.* December 1999. Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`

[13] O. Goldreich. *Signature Schemes – fragments of a chapter.* May 2000. Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-book.html`

[14] O. Goldreich. *Foundation of Cryptography – Basic Tools.* Cambridge University Press, 2001.

Fragments of this book have appeared as [9].

[15] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *JACM*, Vol. 33, No. 4, pages 792–807, 1986.

[16] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st STOC*, pages 25–32, 1989.

[17] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38, No. 1, pages 691–729, 1991.

Preliminary version in *27th FOCS*, 1986.

[18] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC,* pages 218–229, 1987.

[19] O. Goldreich, N. Nisan and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[20] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Jour. of Crypto.*, Vol. 7, No. 1, pages 1–32, 1994.

[21] O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: the highly noisy case. *SIAM J. on Disc. Math.*, Vol. 13, pages 535–570, 2000.

[22] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, Vol. 28, No. 2, pages 270–299, 1984.

Preliminary version in *14th STOC*, 1982.

[23] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. on Comput.*, Vol. 18, pages 186–208, 1989.

Preliminary version in *17th STOC*, 1985.

[24] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM J. on Comput.*, Vol. 28, pages 1364–1396, 1999.

Preliminary versions by Impagliazzo et. al. in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).

[25] L.A. Levin. One-Way Function and Pseudorandom Generators. *Combinatorica*, Vol. 7, pages 357–363, 1987.

[26] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[27] R.C. Merkle and M.E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. *IEEE Trans. Inform. Theory*, Vol. 24, pages 525–530, 1978.

[28] R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *CACM*, Vol. 21, Feb. 1978, pages 120–126.

[29] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Sys. Tech. J.*, Vol. 28, pages 656–715, 1949.

[30] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.