Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Foundations of Cryptography
(Fragments of a Book)

Oded Goldreich
Department of Computer Science
and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel.

February 23, 1995

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Preface

to Dana

Why fragments?
Several years ago, Shafi Goldwasser and myself have decided to write together a book titled "Foundations of Cryptography". In a first burst of energy, I've written most of the material appearing in these fragments, but since then very little progress has been done. The chances that we will complete our original plan within a year or two seem quite slim. In fact, we even fail to commit ourselves to a date on which we will resume work on this project.

What is in these fragments?
These fragments contain a first draft for three major chapters and an introduction chapter. The three chapters are the chapters on computational difficulty (or one-way functions), pseudorandom generators and zero-knowledge. These chapters are quite complete with the exception that the zero-knowledge chapter misses the planned section on non-interactive zero-knowledge. However, none of these chapters has been carefully proofread and I expect them to be full of various mistakes ranging from spelling and grammatical mistakes to minor technical inaccuracies. I hope and believe that they are no fatal mistakes, but I cannot guarantee this either.

A major thing which is missing:
An updated list of references is indeed missing. Instead I enclose an old annotated list of references (compiled mostly in February 1989).

©1995 O. Goldreich.
All rights reserved.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

> **Author's Note:** *Text appearing in italics within indented paragraphs, such as this one, is not part of the book, but rather part of the later comments added to its fragments...*

> **Author's Note:** *The original preface should have started here:*

Revolutionary developments which took place in the previous decade have transformed cryptography from a semi-scientific discipline to a respectable field in theoretical Computer Science. In particular, concepts such as computational indistinguishability, pseudorandomness and zero-knowledge interactive proofs were introduced and classical notions as secure encryption and unforgeable signatures were placed on sound grounds.

This book attempts to present the basic concepts, definitions and results in cryptography. The emphasis is placed on the clarification of fundamental concepts and their introduction 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.

Using this book

> **Author's Note:** *Giving a course based on the material which appears in these fragments is indeed possible, but kind of strange since the basic tasks of encrypting and signing are not covered.*

- Chapters, sections, subsections, and subsubsections denoted by an asterisk (*) were intended for advanced reading.
- Historical notes and suggestions for further reading are provided at the end of each chapter.

  > **Author's Note:** *However, a corresponding list of reference is not provided. Instead, the read may try to trace the paper by using the enclosed annotated list of references (dating to 1989).*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Acknowledgements

> .... very little do we have and inclose which we can call our own in the deep sense of the word. We all have to accept and learn, either from our predecessors or from our contemporaries. Even the greatest genius would not have achieved much if he had wished to extract everything from inside himself. But there are many good people, who do not understand this, and spend half their lives wondering in darkness with their dreams of originality. I have known artists who were proud of not having followed any teacher and of owing everything only to their own genius. Such fools!
>
> [Goethe, *Conversations with Eckermann*, 17.2.1832]

First of all, I would like to thank three remarkable people who had a tremendous influence on my professional development. Shimon Even introduced me to theoretical computer science and closely guided my first steps. Silvio Micali and Shafi Goldwasser led my way in the evolving foundations of cryptography and shared with me their constant efforts of further developing these foundations.

I have collaborated with many researchers, yet I feel that my collaboration with Benny Chor and Avi Wigderson had a fundamental impact on my career and hence my development. I would like to thank them both for their indispensable contribution to our joint research, and for the excitement and pleasure I had when collaborating with them.

Leonid Levin does deserve special thanks as well. I had many interesting discussions with Lenia over the years and sometimes it took me too long to realize how helpful these discussions were.

Clearly, continuing in this pace will waste too much of the publisher's money. Hence, I confine myself to listing some of the people which had contributed significantly to my understanding of the field. These include Laszlo Babai, Mihir Bellare, Michael Ben-Or, Manuel Blum, Ran Canetti (who is an expert in Wine and Opera), Cynthia Dwork, Uri Feige, Mike Fischer, Lance Fortnow, Johan Hastad (who is a special friend), Russel Impagliazzo, Joe Kilian, Hugo Krawcyzk (who still suffers from having been my student), Mike Luby (and his goat), Moni Naor, Noam Nisan, Rafail Ostrovsky, Erez Petrank, Michael Rabin, Charlie

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Rackoff, Steven Rudich, Ron Rivest, Claus Schnorr, Mike Sipser, Adi Shamir, Andy Yao, and Moti Yung.

Author's Note: *I've probably forgot a few names and will get myself in deep trouble for it. Wouldn't it be simpler and safer just to acknowledge that such a task is infeasible?*

In addition, I would like to acknowledge helpful exchange of ideas with Ishai Ben-Aroya, Richard Chang, Ivan Damgard, Amir Herzberg, Eyal Kushilevitz (& sons), Nati Linial, Yishay Mansour, Yair Oren, Phil Rogaway, Ronen Vainish, R. Venkatesan, Yacob Yacobi, and David Zuckerman.

Author's Note: *Written in Tel-Aviv, mainly between June 1991 and November 1992.*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Contents

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Chapter 1

Introduction

In this chapter we shortly discuss the goals of cryptography. In particular, we discuss the problems of secure encryption, digital signatures, and fault-tolerant protocols. These problems lead to the notions of pseudorandom generators and zero-knowledge proofs which are discussed as well.

Our approach to cryptography is based on computational complexity. Hence, this introductory chapter contains also a section presenting the computational models used throughout the book. Likewise, the current chapter contains a section presenting some elementary backgound from probability theory, which is used extensively in the sequal.

1.1 Cryptography – Main Topics

Traditionally, cryptography has been associated with the problem of designing and analysing *encryption schemes* (i.e., schemes which provide secret communication over insecure communication media). However, nowadays, also problems such as constructing unforgeable *digital signatures* and designing *fault-tolerant protocols*, are considered as falling in the domain of cryptography. Furthermore, it turns out that notions as "*pseudorandom generators*" and "*zero-knowledge proofs*" are very related to the above problems, and hence must be treated as well in a book on cryptography. In this section we briefly discuss the above-mentioned terms.

1.1.1 Encryption Schemes

The problem of providing *secret communication over insecure media* is the most basic problem of cryptography. The setting of this problem consists of two parties communicating through a channel which is possibly tapped by an adversary. The parties wish to exchange

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

information with each other, but keep the "wiretapper" as ignorant as possible regrading the contents of this information. Loosely speaking, an encryption scheme is a protocol allowing these parties to communicate *secretly* with each other. Typically, the encryption scheme consists of a pair of algorithms. One algorithm, called *encryption*, is applied by the sender (i.e., the party sending a message), while the other algorithm, called *decryption*, is applied by the receiver. Hence, in order to send a message, the sender first applies the encryption algorithm to the message, and sends the result, called the *ciphertext*, over the channel. Upon receiving a ciphertext, the other party (i.e., the receiver) applies the decryption algorithm to it, and retrieves the original message (called the *plaintext*).

In order for the above scheme to provide secret communication, the communicating parties (at least the receiver) must know something which is not known to the wiretapper. (Otherwise, the wiretapped can decrypt the ciphertext exactly as done by the receiver.) This extra knowledge may take the form of the decryption algorithm itself, or some parameters and/or auxiliary inputs used by the decryption algorithm. We call this extra knowledge the *decryption key*. Note that, without loss of generality, we may assume that the decryption algorithm is known to the wiretapper and that the decryption algorithm needs two inputs: a ciphertext and a decryption key. We stress that the existence of a secret key, not known to the wiretapper, is merely a necessary condition for secret communication.

Evaluating the "security" of an encryption scheme is a very tricky business. A preliminary task is to understand what is "security" (i.e., to properly define what is meant by this intuitive term). Two approaches to defining security are known. The first ("classic") approach is *information theoretic*. It is concerned with the "information" about the plaintext which is "present" in the ciphertext. Loosely speaking, if the ciphertext contains information about the plaintext then the encryption scheme is considered insecure. It has been shown that such high (i.e., "perfect") level of security can be achieved only if the key in use is at least as long as the *total* length of the messages sent via the encryption scheme. The fact, that the key has to be longer than the information exchanged using it, is indeed a drastic limitation on the applicability of such encryption schemes. In particular, it is impractical to use such keys in case *huge* amounts of information need to be secretly communicated (as in computer networks).

The second ("modern") approach, followed in the current book, is based on *computational complexity*. This approach is based on the observation that it **does not matter** *whether the ciphertext contains information about the plaintext*, but rather *whether this information can be* **efficiently extracted**. In other words, instead of asking whether it is *possible* for the wiretapper to extract specific information, we ask whether it is *feasible* for the wiretapper to extract this information. It turns out that the new (i.e., "computational complexity") approach offers security even if the key is much shorter than the total length of the messages sent via the encryption scheme. For example, one may use "pseudorandom generators" (see below) which expand short keys into much longer "pseudo-keys", so that the latter are as secure as "real keys" of comparable length.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

In addition, the computational complexity approach allows the introduction of concepts and primitives which cannot exist under the information theoretic approach. A typical example is the concept of *public-key encryption schemes*. Note that in the above discussion we concentrated on the decryption algorithm and its key. It can be shown that the encryption algorithm must get, in addition to the message, an auxiliary input which depends on the decryption key. This auxiliary input is called the *encryption key*. Traditional encryption schemes, and in particular all the encryption schemes used in the millenniums until the 1980's, operate with an encryption key equal to the decryption key. Hence, the wiretapper in this schemes must be ignorant of the encryption key, and consequently the *key distribution* problem arises (i.e., how can two parties wishing to communicate over an insecure channel agree on a secret encryption/decryption key). (The traditional solution is to exchange the key through an alternative channel which is secure, though "more expensive to use", for example by a convoy.) The computational complexity approach allows the introduction of encryption schemes in which the encryption key may be given to the wiretapper without compromising the security of the scheme. Clearly, the decryption key in such schemes is different and furthermore infeasible to compute from the encryption key. Such encryption scheme, called *public-key*, have the advantage of trivially resolving the key distribution problem since the encryption key can be publicized.

In the chapter devoted to encryption schemes, we discuss private-key and public-key encryption schemes. Much attention is placed on defining the security of encryption schemes. Finally, constructions of secure encryption schemes based on various intractability assumptions are presented. Some of the constructions presented are based on pseudorandom generators, which are discussed in a prior chapter. Other constructions use specific one-way functions such as the RSA function and/or squaring modulo a composite number.

1.1.2 Pseudorandom Generators

It turns out that pseudorandom generators play a central role in the construction of encryption schemes (and related schemes). In particular, pseudorandom generators are the clue to the construction of private-key encryption schemes, and this observation is often used in practice (usually implicitly).

Although the term "pseudorandom generators" is commonly used in practice, both in the contents of cryptography and in the much wider contents of probabilistic procedures, it is important to realize that this term is seldom associated a precise meaning. We believe that using a term without knowing what it means is dangerous in general, and in particular in a delicate business as cryptography. Hence, a precise treatment of pseudorandom generators is central to cryptography.

Loosely speaking, a pseudorandom generator is a deterministic algorithm expanding short random seeds into much longer bit sequences which *appear* to be "random" (although they are not). It other words, although the output of a pseudorandom generator is not

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

really random, it is *infeasible* to tell the difference. It turns out that pseudorandomness and computational difficulty are linked even in a more fundamental manner, as pseudorandom generators can be constructed based on various intractability assumptions. Furthermore, the main result in the area asserts that pseudorandom generators exists if and only if one-way functions exists.

The chapter devoted to pseudorandom generators starts with a treatment of the concept of computational indistinguishability. Pseudorandom generators are defined next, and constructed using special types of one-way functions (defined in a prior chapter). Pseudorandom functions are defined and constructed as well.

1.1.3 Digital Signatures

A problem which did not exist in the "pre-computerized" world is that of a "digital signature". The need to discuss "digital signatures" has arise with the introduction of computer communication in business environment in which parties need to commit themselves to proposals and/or declarations they make. Discussions of "unforgeable signatures" did take place also in previous centuries, but the objects of discussion were handwritten signatures (and not digital ones), and the discussion was not perceived as related to "cryptography".

Relations between encryption and signature methods became possible with the "digitalization" of both, and the introduction of the computational complexity approach to security. Loosely speaking, a *scheme for unforgeable signatures* requires that

- each user can *efficiently generate his own signature* on documents of his choice;
- each user can *efficiently verify* whether a given string is a signature of another (specific) user on a specific document; but
- *nobody can efficiently produce signatures of other users* to documents they did not sign.

We stress that the formulation of unforgeable digital signatures provides also a clear statement of the essential ingredients of handwritten signatures. The ingredients are each person's ability to sign for himself, a universally agreed verification procedure, and the belief (or assertion) that it is infeasible (or at least hard) to forge signatures in a manner that pass the verification procedure. Clearly, it is hard to state to what extent do handwritten signatures meet these requirements. In contrast, our discussion of digital signatures will supply precise statements concerning the extend by which digital signatures meet the above requirements. Furthermore, unforgeable digital signature schemes can be constructed using the same computational assumptions as used in the construction of encryption schemes.

In the chapter devoted to signature schemes, much attention is placed on defining the security (i.e., unforgeability) of these schemes. Next, constructions of unforgeable signature

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

schemes based on various intractability assumptions are presented. In addition, we treat the related problem of message authentication.

Message authentication

Message authentication is a task related to the setting considered for encryption schemes, i.e., communication over an insecure channel. This time, we consider an active adversary which is monitoring the channel and may alter the messages sent on it. The parties communicating through this insecure channel wish to authenticate the messages they send so their counterpart can tell an original message (sent by the sender) from a modified one (i.e., modified by the adversary). Loosely speaking, a *scheme for message authentication* requires that

- each of the communicating parties can *efficiently generate an authentication tag* to any message of his choice;
- each of the communicating parties can *efficiently verify* whether a given string is an authentication tag of a given message; but
- *no external adversary* (i.e., a party other than the communicating parties) *can efficiently produce authentication tags* to messages not sent by the communicating parties.

In some sense "message authentication" is similar to digital signatures. The difference between the two is that in the setting of message authentication the adversary is not required to be able to verify the validity of authentication tags produced by the legitimate users, whereas in the setting of signature schemes the adversary is required to be able to verify the validity of signatures produced by other users. Hence, digital signatures provide a solution to the message authentication problem. On the other hand, message authentication schemes do not necessarily constitute a digital signature scheme.

Signatures widen the scope of cryptography

Considering the problem of digital signatures as belonging to cryptography, widens the scope of this area from the specific "secret communication problem" to a variety of problems concerned with limiting the "gain" obtained by "dishonest" behaviour of parties (that are either internal or external to the system). Specifically

- In the "secret communication problem" (solved by use of encryption schemes) one wishes to reduce as much as possible the information that a potential wiretapper may extract from the communication between two (legitimate) users. In this case, the legitimate system consists of the two communicating parties, and the wiretapper is considered as an external ("dishonest") party.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

- In the "message authentication problem" one aims at prohibiting an (external) wire-tapper from modifying the communication between two (legitimate) users.
- In the "signature problem" one aims at supplying all users of a system with a way of making self-binding statements so that other users may not make statements that bind somebody else. In this case, the legitimate system consists of the set of all users and a potential forger is considered as an internal yet dishonest user.

Hence, in the wide sense, *cryptography is concerned with any problem in which one wishes to limit the affect of dishonest users.* A general treatment of such problems is captured by the treatment of "fault-tolerant" (or cryptographic) protocols.

1.1.4 Fault-Tolerant Protocols and Zero-Knowledge Proofs

A discussion of signature schemes naturally leads to a discussion of cryptographic protocols, since it is of natural concern to ask under what circumstances should a party send his signature to another party. In particular, problems like mutual simultaneous commitment (e.g., contract signing), arise naturally. Another type of problems, which are motivated by the use of computer communication in the business environment, consists of "secure implementation" of protocols (e.g., implementing secret and incorruptible voting).

Simultaneity problems

A typical example of a simultaneity problem is the problem of simultaneous exchange of secrets, of which contract signing is a special case. The setting in a simultaneous exchange of secrets consists of two parties, each holding a "secret". The goal is to execute a protocol so that if both parties follow it correctly then at termination each holds its counterpart's secret, and in any case (even if one party "cheats") the first party "holds" the second party's secret if and only if the second party "holds" the first party's secret. Simultaneous exchange of secrets can be achieved only when assuming the existence of third parties which are trusted to some extend.

Simultaneous exchange of secrets can be easily achieved using the active participation of a trusted third party. Each party sends its secret to the trusted party (using a secure channel), who once receiving both secrets send both of them to both parties. There are two problems with this solution

1. The solution requires *active* participation of an "external" party in all cases (i.e., also in case both parties are honest). We note that other solutions requiring milder forms of participation (of external parties) do exist, yet further discussion is postponed to the chapter devoted to cryptographic protocols.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

2. The solution requires the existence of a *totally trusted* entity. In some applications such an entity does not exist. Nevertheless, in the sequel we discuss the problem of implementing a trusted third party by a set of users with an honest majority (even if the identity of the honest users is not known).

Secure implementation of protocols and trusted parties

A different type of protocol problems are the problems concerned with the secure implementation of protocols. To be more specific, we discuss the problem of evaluating a function of local inputs each held by a different user. An illustrative and motivating example is *voting*, in which the function is majority and the local input held by user A is a single bit representing the vote of user A (e.g., "Pro" or "Con"). We say that a protocol implements a secure evaluation of a specific function if it satisfies

- *privacy*: No party "gains information" on the input of other parties, beyond what is deduced from the value of the function; and
- *robustness*: No party can "influence" the value of the function, beyond the influence obtained by selecting its own input.

It is sometimes required that the above conditions hold with respect to "small" (e.g., minority) coalitions of parties (instead of single parties).

Clearly, if one of the users is known to be totally trusted then there exist a simple solution to the problem of secure evaluation of any function. Each user just sends its input to the trusted party (using a secure channel), who once receiving all inputs, computes the function, sends te outcome to all users, and erase all intermediate computations (including the inputs received) from its memory. Certainly, it is unrealistic to assume that a party can be trusted to such an extend (e.g. that it erases voluntarily what it has "learnt"). Nevertheless, we have seen that the problem of implementing secure function evaluation reduces to the problem of implementing a trusted party. It turns out that a trusted party can be implemented by a set of users with an honest majority (even if the identity of the honest users is not known). This is indeed a major result in the area.

Zero-knowledge as a paradigm

A major tool in the construction of cryptographic protocols is the concept of *zero-knowledge* proof systems, and the fact that zero-knowledge proof systems exist for all languages in $\mathcal{NP}$ (provided that one-way functions exist). Loosely speaking, zero-knowledge proofs yield nothing but the validity of the assertion. Zero-knowledge proofs provide a tool for "forcing" parties to follow a given protocol properly.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

To illustrate the role zero-knowledge proofs, consider a setting in which a party upon receiving an encrypted message should answer with the least significant bit of the message. Clearly, if the party just sends the (least significant) bit (of the message) then there is no way to guarantee that it did not cheat. The party may prove that it did not cheat by revealing the entire message as well as its decryption key, but this would yield information beyond what has been required. A much better idea is to let the party augment the bit it sends by a zero-knowledge proof that this bit is indeed the least significant bit of the message. We stress that the above statement is of the "NP-type" (since the proof specified above can be efficiently verified), and therefore the existence of zero-knowledge proofs for NP-statements implies that the above statement can be proven without revealing anything beyond its validity.

1.2 Some Background from Probability Theory

Probability plays a central role in cryptography. In particular, probability is essential in order to allow a discussion of information or lack of information (i.e., secrecy). We assume that the reader is familiar with the basic notions of probability theory. In this section, we merely present the probabilistic notations that are used in throughout the book, and three useful probabilistic inequalities.

1.2.1 Notational Conventions

Throughout the entire book we will refer only to *discrete* probability distributions. Traditionally, a *random variable* is defined as a function from the sample space into the reals (or integers). In this book we use the term *random variable* also when referring to functions mapping the sample space into the set of binary strings. For example, we may say that X is a random variable assigned values in the set of all strings so that $\text{Prob}(X=00) = \frac{1}{3}$ and $\text{Prob}(X=111) = \frac{2}{3}$. This is indeed a non-standard convention, but a useful one. Also, we will refer directly to the random variables without specifying the probability space on which they are defined. In most cases the probability space consists of all strings of a particular length.

How to read probabilistic statements

All our probabilistic statements refer to functions of random variables which are defined beforehand. Typically, we may write $\text{Prob}(f(X)=1)$, where X is a random variable defined beforehand (and f is a function). An important convention is that *all occurrences of the same symbol in a probabilistic statement refer to the same (unique) random variable*. Hence, if $E(\cdot,\cdot)$ is an expression depending on two variables and X is a random variable then

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

$\text{Prob}(E(X,X))$ denotes the probability that $E(x,x)$ holds when x is chosen with probability $\text{Prob}(X=x)$. Namely,

$$\text{Prob}(E(X,X)) = \sum_x \text{Prob}(X=x) \cdot \text{val}(E(x,x))$$

where $\text{val}(E(x,x))$ equals 1 if $E(x,x)$ holds and equals 0 otherwise. For example, for every random variable X, we have $\text{Prob}(X=X)=1$. We stress that if one wishes to discuss the probability that $E(x,y)$ holds when x and y are chosen independently with identical probability distribution the one needs to define *two* independent random variables each with the same probability distribution. Hence, if X and Y are two independent random variables then $\text{Prob}(E(X,Y))$ denotes the probability that $E(x,y)$ holds when the pair (x,y) is chosen with probability $\text{Prob}(X=x) \cdot \text{Prob}(Y=y)$. Namely,

$$\text{Prob}(E(X,Y)) = \sum_{x,y} \text{Prob}(X=x) \cdot \text{Prob}(Y=y) \cdot \text{val}(E(x,y))$$

For example, for every two independent random variables, X and Y, we have $\text{Prob}(X=Y)=1$ only if both X and Y are trivial (i.e., assign the entire probability mass to a single string).

Typical random variables

Throughout the entire book, U_n denotes a random variable uniformly distributed over the set of strings of length n. Namely, $\text{Prob}(U_n=\alpha)$ equals 2^{-n} if $\alpha \in \{0,1\}^n$ and equals 0 otherwise. In addition, we will occasionally use random variables (arbitrarily) distributed over $\{0,1\}^n$ or $\{0,1\}^{l(n)}$, for some function $l : \mathbb{N} \mapsto \mathbb{N}$. Such random variables are typically denoted by X_n, Y_n, Z_n, etc. We stress that in some cases X_n is distributed over $\{0,1\}^n$ whereas in others it is distributed over $\{0,1\}^{l(n)}$, for some function $l(\cdot)$, typically a polynomial. Another type of random variable, the output of a randomized algorithm on a fixed input, is discussed in the next section.

1.2.2 Three Inequalities

The following probabilistic inequalities will be very useful in course of the book. All inequalities refer to random variables which are assigned real values. The most basic inequality is *Markov Inequality* which asserts that, for random variables assigned values in some interval, some relation must exist between the deviation of a value from the expectation of the random variable and the probability that the random variable is assigned this value. Specifically,

Markov Inequality: Let X be a non-negative random variable and v a real number. Then

$$\text{Prob}\,(X \geq v) < \frac{\text{Exp}(X)}{v}$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Equivalently, $\text{Prob}(X \geq r \cdot \text{Exp}(X)) < \frac{1}{r}$.

Proof:

$$\begin{aligned}
\text{Exp}(X) &= \sum_x \text{Prob}(X=x) \cdot x \\
&> \sum_{x<v} \text{Prob}(X=x) \cdot 0 + \sum_{x \geq v} \text{Prob}(X=x) \cdot v \\
&= \text{Prob}(X \geq v) \cdot v
\end{aligned}$$

The claim follows. ■

Markov inequality is typically used in cases one knows very little about the distribution of the random variable. It suffices to know its expectation and at least one bound on the range of its values.

Exercise 1:

1. Let X be a random variable such that $\text{Exp}(X) = \mu$ and $X \leq 2\mu$. Give an upper bound on $\text{Prob}(X < \frac{\mu}{2})$.
2. Let $0 < \epsilon, \delta < 1$, and Y be a random variable ranging in the interval $[0,1]$ such that $\text{Exp}(Y) = \delta + \epsilon$. Give a lower bound on $\text{Prob}(Y \geq \delta + \frac{\epsilon}{2})$.

Using Markov's inequality, one gets a "possibly stronger" bound for the deviation of a random variable from its expectation. This bound, called Chebyshev's inequality, is useful provided one has additional knowledge concerning the random variable (specifically a good upper bound on its variance).

Chebyshev's Inequality: Let X be a random variable, and $\delta > 0$. Then

$$\text{Prob}\left(|X - \text{Exp}(X)| > \delta\right) < \frac{\text{Var}(X)}{\delta^2}$$

Proof: We define a random variable $Y \stackrel{\text{def}}{=} (X - \text{Exp}(X))^2$, and apply Markov inequality. We get

$$\begin{aligned}
\text{Prob}\left(|X - \text{Exp}(X)| > \delta\right) &= \text{Prob}\left((X - \text{Exp}(X))^2 > \delta^2\right) \\
&< \frac{\text{Exp}((X - \text{Exp}(X))^2)}{\delta^2}
\end{aligned}$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

and the claim follows. ■

Chebyshev's inequality is particularly useful in the analysis of the error probability of approximation via repeated sampling. It suffices to assume that the samples are picked in a pairwise independent manner.

Corollary (*Pairwise Independent Sampling*): Let $X_1, X_2, ..., X_n$ be pairwise independent random variables with the identical expectation, denoted μ, and identical variance, denoted σ^2. Then

$$\mathrm{Prob}\left(\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| > \delta\right) < \frac{\sigma^2}{\delta^2 n}$$

The X_i's are *pairwise independent* of for every $i \neq j$ and all a, b, it holds that $\mathrm{Prob}(X_i = a \wedge X_j = b)$ equals $\mathrm{Prob}(X_i = a) \cdot \mathrm{Prob}(X_j = b)$.

Proof: Define the random variables $\overline{X}_i \stackrel{\mathrm{def}}{=} X_i - \mathrm{Exp}(X_i)$. Note that the $\overline{X}_i$'s are pairwise independent, and each has zero expectation. Applying Chebyshev's inequality to the random variable defined by the sum $\sum_{i=1}^n \frac{X_i}{n}$, and using the linearity of the expectation operator, we get

$$\begin{aligned}\mathrm{Prob}\left(|\sum_{i=1}^n \frac{X_i}{n} - \mu| > \delta\right) &< \frac{\mathrm{Var}\left(\sum_{i=1}^n \frac{X_i}{n}\right)}{\delta^2} \\ &= \frac{\mathrm{Exp}\left(\left(\sum_{i=1}^n \overline{X}_i\right)^2\right)}{\delta^2 \cdot n^2}\end{aligned}$$

Now (again using the linearity of Exp)

$$\mathrm{Exp}\left(\left(\sum_{i=1}^n \overline{X}_i\right)^2\right) = \sum_{i=1}^n \mathrm{Exp}\left(\overline{X}_i^2\right) + \sum_{1 \le i \neq j \le n} \mathrm{Exp}\left(\overline{X}_i \overline{X}_j\right)$$

By the pairwise independence of the $\overline{X}_i$'s, we get $\mathrm{Exp}(\overline{X}_i \overline{X}_j) = \mathrm{Exp}(\overline{X}_i) \cdot \mathrm{Exp}(\overline{X}_j)$, and using $\mathrm{Exp}(\overline{X}_i) = 0$, we get

$$\mathrm{Exp}\left(\left(\sum_{i=1}^n \overline{X}_i\right)^2\right) = n \cdot \sigma^2$$

The corollary follows. ■

Using pairwise independent sampling, the error probability in the approximation is decreasing linearly with the number of sample points. Using totally independent sampling

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

points, the error probability in the approximation can be shown to decrease exponentially with the number of sample points. (The random variables $X_1, X_2, ..., X_n$ are said to be *totally independent* if for every sequence $a_1, a_2, ..., a_n$ it folds that $\text{Prob}(\wedge_{i=1}^n X_i = a_i)$ equals $\prod_{i=1}^n \text{Prob}(X_i = a_i)$.)

The bounds quote below are (weakenings of) a special case of the *Martingale Tail Inequality* which suffices for our purposes. The first bound, commonly referred to as *Chernoff Bound*, concerns 0-1 random variables (i.e., random variables which are assigned as values either 0 or 1).

Chernoff Bound: Let $p \leq \frac{1}{2}$, and $X_1, X_2, ..., X_n$ be independent 0-1 random variables so that $\text{Prob}(X_i = 1) = p$, for each i. Then for all δ, $0 < \delta \leq p(1-p)$, we have

$$\text{Prob}\left(\left|\frac{\sum_{i=1}^n X_i}{n} - p\right| > \delta\right) < 2 \cdot e^{-\frac{\delta^2}{2p(1-p)} \cdot n}$$

We will usually apply the bound with a constant $p \approx \frac{1}{2}$. In this case, n independent samples give an approximation which deviates by δ from the expectation with probability ϵ which is exponentially decreasing with $\delta^2 n$. Such an approximation is called an (ϵ, δ)-*approximation*, and can be achieved using $n = O(\delta^{-2} \cdot \log(1/\epsilon))$ sample points. It is important to remember that the sufficient number of sample points is polynomially related to δ^{-1} and logarithmically related to ϵ^{-1}. So using $\text{poly}(n)$ many samples the error probability (i.e. ϵ) can be made negligible (as a function in n), but the accuracy of the estimation can be bounded above by any fixed polynomial fraction (but cannot be made negligible).

A more general bound, useful in the approximations of the expectation of a general random variable (not necessarily 0-1), is given below.

Hoefding Inequality: Let $X_1, X_2, ..., X_n$ be n independent random variables with identical probability distribution, each ranging over the (real) interval $[a, b]$, and let μ denote the expected value of each of these variables. Then,

$$\text{Prob}\left(\left|\frac{\sum_{i=1}^n X_i}{n} - p\right| > \delta\right) < 2 \cdot e^{-\frac{2\delta^2}{b-a} \cdot n}$$

Hoefding Inequality is useful in estimating the average value of a function defined over a large set of values. It can be applied provided we can efficiently sample the set and have a bound on the possible values (of the function).

Exercise 2: Let $f : \{0,1\}^* \mapsto [0,1]$ be a polynomial-time computable function, and let $F(n)$ denote the average value of f over $\{0,1\}^n$. Namely,

$$F(n) \stackrel{\text{def}}{=} \frac{\sum_{x \in \{0,1\}^n} f(x)}{2^n}$$

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Let $p(\cdot)$ be a polynomial. Present a probabilistic polynomial-time algorithm that on input 1^n outputs an estimate to $F(n)$, denoted $A(n)$, such that

$$\text{Prob}\left(|F(n) - A(n)| > \frac{1}{p(n)}\right) < 2^{-n}$$

Guidance: The algorithm selects at random polynomially many (how many?) sample points $s_i \in \{0,1\}^n$. These points are selected independently and with uniform probability distribution (why?). The algorithm outputs the average value taken over this sample. Analyze the performance of the algorithm using Hoefding Inequality (hint: define random variables $X_i = f(s_i)$).

1.3 The Computational Model

Our approach to cryptography is heavily based on computational complexity. Thus, some background on computational complexity is required for our discussion of cryptography. In this section, we briefly recall the definitions of the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{BPP}$, non-uniform $\mathcal{P}$ (i.e., $\mathcal{P}$/poly), and the concept of oracle machines. In addition, we discuss the type of intractability assumptions used throughout the rest of the book.

1.3.1 P, NP, and NP-completeness

A conservative approach to computing devices associates efficient computations with the complexity class $\mathcal{P}$. Jumping ahead, we note that the approach taken in this book is a more liberal one in that it allows the computing devices to use coin tosses.

Definition 1.1 *$\mathcal{P}$ is the class of languages which can be recognized by a (deterministic) polynomial-time machine (algorithm). Language L is* recognizable in polynomial-time *if there exists a (deterministic) Turing machine M and a polynomial $p(\cdot)$ such that*

- *On input a string x, machine M halts after at most $p(|x|)$ steps.*
- *$M(x) = 1$ if and only if $x \in L$.*

Likewise, the complexity class $\mathcal{NP}$ is associated with computational problems having solutions that, once given, can be efficiently tested for validity. It is customary to define $\mathcal{NP}$ as the class of languages which can be recognized by a non-deterministic polynomial-time machine. A more fundamental interpretation of $\mathcal{NP}$ is given by the following equivalent definition.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Definition 1.2 *A language L is in $\mathcal{NP}$, if there exists a Boolean relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(\cdot)$ such that R_L can be recognized in (deterministic) polynomial-time and $x \in L$ if and only if there exists a y such that $|y| \leq p(|x|)$ and $(x,y) \in R_L$. Such a y is called a* witness for membership of $x \in L$.

Thus, $\mathcal{NP}$ consists of the set of languages for which there exist short proofs of membership that can be efficiently verified. It is widely believed that $\mathcal{P} \neq \mathcal{NP}$, and settling this conjecture is certainly the most intriguing open problem in Theoretical Computer Science. If indeed $\mathcal{P} \neq \mathcal{NP}$ then there exists a language $L \in \mathcal{NP}$ so that for every algorithm recognizing L has super-polynomial running-time *in the worst-case*. Certainly, all $\mathcal{NP}$-complete languages (see definition below) will have super-polynomial time complexity *in the worst-case*.

Definition 1.3 *A language is $\mathcal{NP}$-*complete *if it is in $\mathcal{NP}$ and every language in $\mathcal{NP}$ is polynomially-reducible to it. A language L is* polynomially-reducible *to a language L' if there exist a polynomial-time computable function f so that $x \in L$ if and only if $f(x) \in L'$.*

Among the languages known to be $\mathcal{NP}$-complete are *Satisfiablity* (of propositional formulae), and *Graph Colorability*.

1.3.2 Probabilistic Polynomial-Time

The basic thesis underlying our discussion is the association of "efficient" computations with probabilistic polynomial-time computations. Namely, we will consider as efficient only randomized algorithms (i.e., probabilistic Turing machines) whose running time is bounded by a polynomial in the length of the input. Such algorithms (machines) can be viewed in two equivalent ways.

One way of viewing randomized algorithms is to allow the algorithm to make random moves ("toss coins"). Formally this can be modeled by a Turing machine in which the transition function maps pairs of the form $(\langle\text{state}\rangle, \langle\text{symbol}\rangle)$ to two possible triples of the form $(\langle\text{state}\rangle, \langle\text{symbol}\rangle, \langle\text{direction}\rangle)$. The next step of such a machine is determined by a random choice of one of these triples. Namely, to make a step, the machine chooses at random (with probability one half for each possibility) either the first triple or the second one, and then acts accordingly. These random choices are called the *internal coin tosses* of the machine. The output of a probabilistic machine, M, on input x is not a string but rather a random variable assuming strings as possible values. This random variable, denoted $M(x)$, is induced by the internal coin tosses of M. By $\text{Prob}(M(x)=y)$ we mean the probability that machine M on input x outputs y. The probability space is that of all possible outcomes for the internal coin taken with uniform probability distribution. The last sentence is slightly more problematic than it seems. The simple case is when, on input

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

x, machine M always makes the same number of internal coin tosses (independent of their outcome). Since, we only consider polynomial-time machines, we may assume without loss of generality, that the number of coin tosses made by M on input x is independent of their outcome, and is denoted by $t_M(x)$. We denote by $M_r(x)$ the output of M on input x when r is the outcome of its internal coin tosses. Then, $\text{Prob}(M(x)=y)$ is merely the fraction of $r \in \{0,1\}^{t_M(x)}$ for which $M_r(x) = y$. Namely,

$$\text{Prob}\left(M(x)=y\right) = \frac{|\{r \in \{0,1\}^{t_M(x)} : M_r(x)=y\}|}{2^{t_M(x)}}$$

The second way of looking at randomized algorithms is to view the outcome of the internal coin tosses of the machine as an auxiliary input. Namely, we consider deterministic machines with two inputs. The first input plays the role of the "real input" (i.e. x) of the first approach, while the second input plays the role of a possible outcome for a sequence of internal coin tosses. Thus, the notation $M(x,r)$ corresponds to the notation $M_r(x)$ used above. In the second approach one considers the probability distribution of $M(x,r)$, for any *fixed* x and a uniformly chosen $r \in \{0,1\}^{t_M(x)}$. Pictorially, here the coin tosses are not "internal" but rather supplied to the machine by an "external" coin tossing device.

Before continuing, let me remark that one should not confuse the fictitious model of "non-deterministic" machines with the model of probabilistic machines. The first is an unrealistic model which is useful for talking about search problems the solutions to which can be efficiently verified (e.g., the definition of $\mathcal{NP}$), while the second is a realistic model of computation.

In the sequel, unless otherwise stated, a *probabilistic polynomial-time Turing machine* means a probabilistic machine that always (i.e., independently of the outcome of its internal coin tosses) halts after a polynomial (in the length of the input) number of steps. It follows that the number of coin tosses of a probabilistic polynomial-time machine M is bounded by a polynomial, denoted T_M, in its input length. Finally, without loss of generality, we assume that on input x the machine always makes $T_M(|x|)$ coin tosses.

Thesis: *Efficient computations correspond to computations that can be carried out by probabilistic polynomial-time Turing machines.*

A complexity class capturing these computations is the class, denoted $\mathcal{BPP}$, of languages recognizable (with high probability) by probabilistic polynomial-time machines. The probability refers to the event "the machine makes correct verdict on string x".

Definition 1.4 (Bounded-Probability Polynomial-time — $\mathcal{BPP}$): *$\mathcal{BPP}$ is the class of languages which can be recognized by a probabilistic polynomial-time machine (i.e., randomized algorithm). We say that L is* recognized *by the probabilistic polynomial-time machine M if*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

- *For every* $x \in L$ *it holds that* $\text{Prob}(M(x)=1) \geq \frac{2}{3}$.
- *For every* $x \notin L$ *it holds that* $\text{Prob}(M(x)=0) \geq \frac{2}{3}$.

The phrase "bounded-probability" indicates that the success probability is bounded away from $\frac{1}{2}$. In fact, substituting in Definition 1.4 the constant $\frac{2}{3}$ by any other constant greater than $\frac{1}{2}$ does not change the class defined. More generally:

Exercise 1: Prove that Definition 1.4 is robust under the substitution of $\frac{2}{3}$ by $\frac{1}{2} + \frac{1}{p(|x|)}$, for every polynomial $p(\cdot)$. Namely, that $L \in \mathcal{BPP}$ if there exists a polynomial $p(\cdot)$ and a probabilistic polynomial-time machine, M, such that

- For every $x \in L$ it holds that $\text{Prob}(M(x)=1) \geq \frac{1}{2} + \frac{1}{p(|x|)}$.
- For every $x \notin L$ it holds that $\text{Prob}(M(x)=0) \geq \frac{1}{2} + \frac{1}{p(|x|)}$.

Guidance: Given a probabilistic polynomial-time machine M satisfying the above condition, construct a probabilistic polynomial-time machine M' as follows. On input x, machine M', runs $O(p(|x|))$ many copies of M, on the same input x, and rules by majority. Use Chebyshev's inequality (see Sec. 1.2) to show that M' is correct with probability $> \frac{2}{3}$.

Exercise 2: Prove that Definition 1.4 is robust under the substitution of $\frac{2}{3}$ by $1 - 2^{-|x|}$.
Guidance: Similar to Exercise 1, except that you have to use a stronger probabilistic inequality (namely Chernoff bound — see Sec. 1.2).

We conclude that languages in $\mathcal{BPP}$ can be recognized by probabilistic polynomial-time machines with a negligible error probability. By *negligible* we call any function which decreases faster than one over any polynomial. Namely,

Definition 1.5 (negligible): *We call a function* $\mu : \mathbb{N} \mapsto \mathbb{N}$ negligible *if for every polynomial* $p(\cdot)$ *there exists an* N *such that for all* $n > N$

$$\mu(n) < \frac{1}{p(n)}$$

For example, the functions $2^{-\sqrt{n}}$ and $n^{-\log_2 n}$, are negligible (as functions in n). Negligible function stay this way when multiplied by any fixed polynomial. Namely, for every negligible function μ and any polynomial p, the function $\mu'(n) \stackrel{\text{def}}{=} p(n) \cdot \mu(n)$ is negligible. It follows that an event which occurs with negligible probability is highly unlikely to occur even if we repeat the experiment polynomially many times.

Convention: In Definition 1.5 we used the phrase "there exists an N such that for all $n > N$". In the future we will use the shorter and less tedious phrase "for all sufficiently large n". This makes one quantifier (i.e., the $\exists N$) implicit, and is particularly beneficial in statements that contain several (more essential) quantifiers.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

1.3.3 Non-Uniform Polynomial-Time

A stronger model of efficient computation is that of non-uniform polynomial-time. This model will be used only in the negative way; namely, for saying that even such machines cannot do something.

A *non-uniform polynomial-time "machine"* is a pair $(M, \overline{a})$, where M is a two-input polynomial-time machine and $\overline{a} = a_1, a_2, ...$ is an infinite sequence such that $|a_n| = \text{poly}(n)$. For every x, we consider the computation of machine M on the input pair $(x, a_{|x|})$. Intuitively, a_n may be thought as an extra "advice" supplied from the "outside" (together with the input $x \in \{0,1\}^n$). We stress that machine M gets the same advice (i.e., a_n) on all inputs of the same length (i.e., n). Intuitively, the advice a_n may be useful in some cases (i.e., for some computations on inputs of length n), but it is unlikely to encode enough information to be useful for all 2^n possible inputs.

Another way of looking at non-uniform polynomial-time "machines" is to consider an infinite sequence of machines, $M_1, M_2, ...$ so that both the length of the description of M_n and its running time on inputs of length n are bounded by polynomial in n (fixed for the entire sequence). Machine M_n is used only on inputs of length n. Note the correspondence between the two ways of looking at non-uniform polynomial-time. The pair $(M, (a_1, a_2, ...))$ (of the first definition) gives rise to an infinite sequence of machines $M_{a_1}, M_{a_2}, ...$, where $M_{a_{|x|}}(x) \stackrel{\text{def}}{=} M(x, a_{|x|})$. On the other hand, a sequence $M_1, M_2, ...$ (as in the second definition) gives rise to the pair $(U, (\langle M_1 \rangle, \langle M_2 \rangle, ...))$, where U is the universal Turing machine and $\langle M_n \rangle$ is the description of machine M_n (i.e., $U(x, \langle M_{|x|} \rangle) = M_{|x|}(x)$).

In the first sentence of the current subsection, non-uniform polynomial-time has been referred to as a stronger model than probabilistic polynomial-time. This statement is valid in many contexts (e.g., language recognition as in Theorem 1 below). In particular it will be valid in all contexts we discuss in this book. So we have the following informal "meta-theorem"

Meta-Theorem: Whatever can be achieved by probabilistic polynomial-time machines can be achieved by non-uniform polynomial-time "machines".

The meta-theorem is clearly wrong if one thinks of the task of tossing coins... So the meta-theorem should not be understood literally. It is merely an indication of real theorems that can be proven in reasonable cases. Let's consider the context of language recognition.

Definition 1.6 *The complexity class* non-uniform polynomial-time *(denoted* $\mathcal{P}/\text{poly}$*) is the class of languages L which can be recognized by a non-uniform (sequence) polynomial-time "machine". Namely, $L \in \mathcal{P}/\text{poly}$ if there exists an infinite sequence of machines $M_1, M_2, ...$ satisfying*

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

1. *There exists a polynomial $p(\cdot)$ such that, for every n, the description of machine M_n has length bounded above by $p(n)$.*
2. *There exists a polynomial $q(\cdot)$ such that, for every n, the running time of machine M_n on each input of length n is bounded above by $q(n)$. has length $\leq p(n)$.*
3. *For every n and every $x \in \{0,1\}^n$, machine M_n accepts x if and only if $x \in L$.*

Note that the non-uniformity is implicit in the lack of a requirement concerning the construction of the machines in the sequence. It is only required that these machines exist. In contrast, if one augments Definition 1.6 by requiring the existence of a polynomial-time algorithm that on input 1^n (n presented in unary) outputs the description of M_n then one gets a cumbersome way of defining $\mathcal{P}$. On the other hand, it is obvious that $\mathcal{P} \subseteq \mathcal{P}/\text{poly}$ (in fact strict containment can be proven by considering non-recursive unary languages). Furthermore,

Theorem 1: $\mathcal{BPP} \subseteq \mathcal{P}/\text{poly}$.

Proof: Let M be a probabilistic machine recognizing $L \in \mathcal{BPP}$. Let $\xi_L(x) \stackrel{\text{def}}{=} 1$ if $x \in L$ and $\xi_L(x) = 0$ otherwise. Then, for every $x \in \{0,1\}^*$,

$$\text{Prob}(M(x) = \xi_L(x)) \geq \frac{2}{3}$$

Assume, without loss of generality, that on each input of length n, machine M uses the same number, $m = \text{poly}(n)$, of coin tosses. Let $x \in \{0,1\}^n$. Clearly, we can find for each $x \in \{0,1\}^n$ a sequence of coin tosses $r \in \{0,1\}^m$ such that $M_r(x) = \xi_L(x)$ (in fact most sequences r have this property). But can one sequence $r \in \{0,1\}^m$ fit all $x \in \{0,1\}^n$? Probably not (provide an example!). Nevertheless, we can find a sequence $r \in \{0,1\}^n$ which fits $\frac{2}{3}$ of all the x's of length n. This is done by a counting argument (which asserts that if $\frac{2}{3}$ of the r's are good for each x then there is an r which is good for at least $\frac{2}{3}$ of the x's). However, this does not give us an r which is good for all $x \in \{0,1\}^n$. To get such an r we have to apply the above argument on a machine M' with exponentially vanishing error probability. Such a machine is guaranteed by Exercise 2. Namely, for every $x \in \{0,1\}^*$,

$$\text{Prob}(M'(x) = \xi_L(x)) > 1 - 2^{-|x|}$$

Applying the argument now we conclude that there exists an $r \in \{0,1\}^m$, denoted r_n, which is good for *more than* a $1 - 2^{-n}$ fraction of the $x \in \{0,1\}^n$. It follows that r_n is good for all the 2^n inputs of length n. Machine M' (viewed as a deterministic two-input machine) together with the infinite sequence $r_1, r_2, \ldots$ constructed as above, demonstrates that L is in $\mathcal{P}/\text{poly}$. ■

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

Finally, let me mention a more convenient way of viewing non-uniform polynomial-time. This is via (non-uniform) families of polynomial-size Boolean circuits. A *Boolean circuit* is a directed acyclic graph with internal nodes marked by elements in $\{\wedge, \vee, \neg\}$ Nodes with no ingoing edges are called *input nodes*, and nodes with no outgoing edges are called *output nodes*, A node mark $\neg$ may have only one child. Computation in the circuit begins with placing input bits on the input nodes (one bit per node) and proceeds as follows. If the children of a node (of indegree d) marked $\wedge$ have values $v_1, v_2, ..., v_d$ then the node gets the value $\wedge_{i=1}^{d} v_i$. Similarly for nodes marked $\vee$ and $\neg$. The output of the circuit is read from its output nodes. The *size* of a circuit is the number of its edges. A *polynomial-size circuit family* is an infinite sequence of Boolean circuits, $C_1, C_2, ...$ such that, for every n, the circuit C_n has n input nodes and size $p(n)$, where $p(\cdot)$ is a polynomial (fixed for the entire family). Clearly, the computation of a Turing machine M on inputs of length n can be simulated by a single circuit (with n input nodes) having size $O((|\langle M \rangle| + n + t(n))^2)$, where $t(n)$ is a bound on the running time of M on inputs of length n. Thus, a non-uniform sequence of polynomial-time machines can be simulated by a non-uniform family of polynomial-size circuits. The converse is also true as machines with polynomial description length can incorporate polynomial-size circuits and simulate their computations in polynomial-time. The thing which is nice about the circuit formulation is that there is no need to repeat the polynomiality requirement twice (once for size and once for time) as in the first formulation.

1.3.4 Intractability Assumptions

We will consider as *intractable* those tasks which cannot be performed by probabilistic polynomial-time machines. However, the adverserial tasks in which we will be interested (e.g., "breaking an encryption scheme", "forging signatures", etc.) can be performed by non-deterministic polynomial-time machines (since the solutions, once found, can be easily tested for validity). Thus, the computational approach to cryptography (and in particular most of the material in this book) is *interesting* only if $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (which certainly implies $\mathcal{P} \neq \mathcal{NP}$). We use the phrase "not interesting" (rather than "not valid") since all our statements will be of the form "if $\langle$`intractability assumption`$\rangle$ then $\langle$`useful consequence`$\rangle$". The statement remains valid even if $\mathcal{P} = \mathcal{NP}$ (or just $\langle$`intractability assumption`$\rangle$ which is never weaker than $\mathcal{P} \neq \mathcal{NP}$ is wrong), but in such a case the implication is of little interest (since everything is implied by a fallacy).

In most places where we state that "if $\langle$`intractability assumption`$\rangle$ then $\langle$`useful consequence`$\rangle$" it will be the case that $\langle$`useful consequence`$\rangle$ either implies $\langle$`intractability assumption`$\rangle$ or some weaker form of it, which in turn implies $\mathcal{NP} - \mathcal{BPP} \neq \emptyset$. Thus, in light of the current state of knowledge in complexity theory, one cannot hope for asserting $\langle$`useful consequence`$\rangle$ without any intractability assumption.

In few cases an assumption concerning the limitations of probabilistic polynomial-time machines (e.g., $\mathcal{BPP}$ does not contain $\mathcal{NP}$) will not suffice, and we will use instead an

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

assumption concerning the limitations of non-uniform polynomial-time machines. Such an assumption is of course stronger. But also the consequences in such a case will be stronger as they will also be phrased in terms of non-uniform complexity. However, since all our proofs are obtained by reductions, an implication stated in terms of probabilistic polynomial-time is stronger (than one stated in terms of non-uniform polynomial-time), and will be preferred unless it is either not known or too complicated. This is the case since a probabilistic polynomial-time reduction (proving implication in its probabilistic formalization) always implies a non-uniform polynomial-time reduction (proving the statement in its non-uniform formalization), but the converse is not always true. (The current paragraph may be better understood in the future after seeing some concrete examples.)

Finally, we mention that intractability assumptions concerning worst-case complexity (e.g., $\mathcal{P} \neq \mathcal{NP}$) will not suffice, because we will *not be satisfied* with their corresponding consequences. Cryptographic schemes which are guaranteed to be *hard to break in the* **worst-case** are useless. A cryptographic scheme must be unbreakable on "most cases" (i.e., "typical case") which implies that it is *hard to break on the average*. It follows that, since we are not able to prove that "worst-case intractability" imply analogous "intractability for average case" (such a result would be considered a breakthrough in complexity theory), our intractability assumption must concern average-case complexity.

1.3.5 Oracle Machines

The original utility of oracle machines in complexity theory is to capture notions of reducibility. In this book we use oracle machines for a different purpose altogether. We use an oracle machine to model an adversary which may use a cryptosystem in course of its attempt to break it.

Definition 1.7 *A (deterministic/probabilistic)* oracle machine *is a (deterministic/probabilistic) Turing machine with an additional tape, called the* oracle tape*, and two special states, called* oracle invocation *and* oracle appeared*. The* computation of the deterministic oracle machine M on input x and access to the oracle $f : \{0,1\}^* \mapsto \{0,1\}^*$ *is defined by the* successive configuration relation*. For configurations with state different from "oracle invocation" the next configuration is defined as usual. Let γ be a configuration in which the state is "oracle invocation" and the contents of the oracle tape is q. Then the configuration following γ is identical to γ, except that the state is "oracle appeared" and the contents of the oracle tape is $f(q)$. The string q is called M's* query *and $f(q)$ is called thee* oracle reply*. The computation of a probabilistic oracle machine is defined analogously.*

We stress that the running time of an oracle machine is the number of steps made during its computation, and that the oracle's reply on each query is obtained in a single step.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

1.4 Motivation to the Formal Treatment

It is indeed unfortunate that our *formal treatment* of the field of cryptography requires justification. Nevertheless, we prefer to address this (unjustified) requirement rather than ignore it. In the rest of this section we address three related issues

1. the mere need for a formal treatment of the field;
2. the practical meaning and/or consequences of the formal treatment;
3. the "conservative" tendencies of the treatment.

Parts of this section may become more clear after reading any of the chapters 3–7.

1.4.1 The Need to Formalize Intuition

An abstract justification

We believe that one of the roles of science is to formulate our intuition about reality so that this intuition can be carefully examined, and consequently either be justified as sound or be rejected as false. Notably, there are many cases in which our initial intuition turns out to be correct, as well as many cases in which our initial intuition turns out to be wrong. The more we understand the discipline the better our intuition becomes. At this stage in history it would be very presumptuous to claim that we have good intuition about the nature of efficient computation. In particular, we even don't know the answer to a basis question as whether $\mathcal{P}$ is strictly contained in $\mathcal{NP}$, let alone having an understanding what makes one computation problem hard while a seemingly related computational problem is easy. Consequently, we should be extremely careful when making assertions about what can or cannot be efficiently computed. Unfortunately, making assertions about what can or cannot be efficiently computed is exactly what cryptography is all about... Not to mention that may of the problems of cryptography have a much more cumbersome and delicate description than what is usually standard in complexity theory. Hence, not only that there is a need to formalize "intuition" in general, but the need to formalize "intuition" is particularly required in a sensitive field as cryptography.

A concrete justification

Cryptography, as a discipline, is well-motivated. Consequently, cryptographic issues are being discussed by many researchers, engineers, and students. Unfortunately, most of these discussions are carried out without a precise definition of their subject matter. Instead it is implicitly assumed that the basic concepts of cryptography (e.g., secure encryption)

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

are self-evident (since they are so intuitive), and that there is no need to present adequate definitions. The fallacy of this assumption is demonstrated by the abandon of papers (not to mention private discussion) which derive and/or jump into wrong conclusions concerning security. In most cases these wrong conclusions can be traced back into implicit misconceptions regarding security, which could not have escaped the eyes of the authors if made explicitly. We avoid listing all these cases here for several obvious reasons. Nevertheless, we mention one well-known example.

In around 1979, Ron Rivest claimed that no signature scheme that is "proven secure assuming the intractability of factoring" can resist a "chosen message attack". His argument was based on an implicit (and unjustified) assumption concerning the nature of a "proof of security (which assumes the intractability of factoring)". Consequently, for several years it was believe that one has to choose between having a signature scheme "proven to be unforgeable under the intractability of factoring" and having a signature scheme which resist a "chosen message attack". However, in 1984 Goldwasser, Micali and Rivest (himself) pointed out the fallacy on which Rivest's argument (of 1979) was based, and furthermore presented signature schemes which resist a "chosen message attack", under general assumptions. In particular, the intractability of factoring suffices for proving that there exists a signature scheme which resist "forgery", even under a "chosen message attack".

To summary, the basic concepts of cryptography indeed very intuitive, yet they are *not* are self-evident and/or well-understood. Hence, we do not understand these issues well enough yet to be able to discuss them *correctly* without using precise definitions.

1.4.2 The Practical Consequences of the Formal Treatment

As customary in complexity theory, our treatment is presented in terms of asymptotic analysis of algorithms. This makes the statement of the results somewhat less cumbersome, but is *not* essential to the underlying ideas. Hence, the results, although stated in an "abstract manner", lend themselves to concrete interpolations. To clarify the above statement we consider a generic example.

A typical result presented in this book relates two computational problems. The first problem is a simple computational problem which is assumed to be intractable (e.g., intractability of factoring), whereas the second problem consists of "breaking" a specific implementation of a useful cryptographic primitive (e.g., a specific encryption scheme). The abstract statement may assert that if integer factoring cannot be performed in polynomial-time then the encryption scheme is secure in the sense that it cannot be "broken" in polynomial-time. Typically, the statement is proven by a fixed polynomial-time reduction of integer factorization to the problem of breaking the encryption scheme. Hence, by working out the constants one can derive a statement of the following type: *if factoring integers of* `X` (say 300) *decimal digits is infeasible in practice then the encryption scheme is secure in practice provided one uses a key of length* `Y` (say 500) *decimal digits.* Actually, the statement will

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.

have to be more cumbersome so that it includes also the computing power of the real machines. Namely, *if factoring integers of 300 decimal digits cannot be done using 1000 years of a Cray then the encryption scheme cannot be broken in 10 years by a Cray, provided one uses a key of length 500 decimal digits.* We stress that the relation between the four parameters mentioned above can be derived from the reduction (used to prove the abstract statement). For most results these reduction yield a reasonable relation between the various parameters. Consequently, all cryptographic primitives considered in this book (i.e., public and private-key encryption, signatures, zero-knowledge, pseudorandom generators, fault-tolerant protocols) can be implemented in practice based on reasonable intractability assumptions (such as the unfeasibility of factoring 500 digit integers).

In few cases, the reductions *currently known* do not yield practical consequences, since the "security parameter" (e.g., key length) in the derived cryptographic primitive has to be too large. In all these cases, the "impracticality" of the result is explicitly stated, and the reader is encouraged to try to provide a more efficient reduction that would have practical consequences. Hence, we do not consider these few cases as indicating a deficiency in our approach, but rather as important open problems.

1.4.3 The Tendency to be Conservative

When reaching the chapters in which cryptographic primitives are defined (specifically in Chapters 3 through 7), the reader may notice that we are unrealistically "conservative" in our definitions of security. In other words, we are unrealistically liberal in our definition of insecurity. Technically speaking, this tendency raises no problems since our primitives which are secure in a very strong sense are certainly secure also in the (more restricted) reasonable sense. Furthermore, we are able to implement such (strongly secure) primitives using reasonable intractability assumptions, and in most cases one can show that such assumptions are necessary even for much weaker (and in fact less than minimal) notions of security. Yet the reader may wonder why we choose to present definitions which seem stronger than what is required in practice.

The reason to our tendency to be conservative, when defining security, is that it is extremely difficult to capture what is exactly require in practice. Furthermore, a certain level in security may be required in one application, whereas another level is required in a different application. In seems impossible to cover whatever can be required in all applications without taking our conservative approach. In the sequel we shall see how one can define security in a way covering all possible practical applications.

Extracted from a working draft of Goldreich's FOUNDATIONS OF CRYPTOGRAPHY. See copyright notice.